IODD

IO-Link Device Description

Specification

related to
IO-Link Communication Specification V1.0
and
IODD Schemas V1.0.1

Version 1.0.1
March 2010

Order No: 10.012

File name: IOL-Device-Desc-Spec_10012_V101_100304.doc

Prepared, approved and released by the IO-Link Consortium

Any comments, proposals, requests on this document are appreciated. Please use www.io-link-projects.com for your entries and provide name and email address.
Login: *IO-Link-DD*
Password: *Report*

Important notes:

NOTE 1 The IO-Link Consortium Rules shall be considered prior to the development and marketing of IO-Link products. The document can be downloaded from the www.io-link.com portal.

NOTE 2 Any IO-Link device shall provide an associated IODD file. Easy access to the file and potential updates shall be possible. It is the responsibility of the IO-Link device manufacturer to test the IODD file with the help of the IODD-Checker tool available per download from www.io-link.com.

NOTE 3 Any IO-Link device shall provide an associated manufacturer declaration on the conformity of the device with the IO-Link Communication Specification and its related IODD, and test documents, available per download from www.io-link.com.

Disclaimer:

The attention of adopters is directed to the possibility that compliance with or adoption of IO-Link Consortium specifications may require use of an invention covered by patent rights. The IO-Link Consortium shall not be responsible for identifying patents for which a license may be required by any IO-Link Consortium specification, or for conducting legal inquiries into the legal validity or scope of those patents that are brought to its attention. IO-Link Consortium specifications are prospective and advisory only. Prospective users are responsible for protecting themselves against liability for infringement of patents.

The information contained in this document is subject to change without notice. The material in this document details an IO-Link Consortium specification in accordance with the license and notices set forth on this page. This document does not represent a commitment to implement any portion of this specification in any company's products.

WHILE THE INFORMATION IN THIS PUBLICATION IS BELIEVED TO BE ACCURATE, THE IO-LINK CONSORTIUM MAKES NO WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, WITH REGARD TO THIS MATERIAL INCLUDING, BUT NOT LIMITED TO ANY WARRANTY OF TITLE OR OWNERSHIP, IMPLIED WARRANTY OF MERCHANTABILITY OR WARRANTY OF FITNESS FOR PARTICULAR PURPOSE OR USE.

In no event shall the IO-Link Consortium be liable for errors contained herein or for indirect, incidental, special, consequential, reliance or cover damages, including loss of profits, revenue, data or use, incurred by any user or any third party. Compliance with this specification does not absolve manufacturers of IO-Link equipment, from the requirements of safety and regulatory agencies (TÜV, BIA, UL, CSA, etc.).

IO-Link ® is registered trade mark. The use is restricted for members of the IO-Link Consortium. More detailed terms for the use can be found in the IO-Link Consortium Rules on www.io-link.com.

Conventions:

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

Publisher:
IO-Link Consortium
Haid-und-Neu-Str. 7
76131 Karlsruhe
Germany
Phone: +49 721 / 96 58 590
Fax: +49 721 / 96 58 589
E-mail: info@io-link.com
Web site: www.io-link.com

Content

Figures

Tables

Revision Log

Version	Originator	Date	Change Note / History / Reason
1.0	Team Technology	18-Sep-09	Final release of IODD specification
1.0	Team Technology	18-Jan-09	Formal adjustments and errata
1.0.1	Team Technology	04-Mar-10	Incorporated errata document and schema changes due to clarification of data transfer

1 Introduction

An IODD (IO-Link Device Description) is a set of files that formally describes an IO-Link Device.

The IODD is created by the IO-Link Device vendor and shall be sufficient for engineering tools to identify, communicate, parameterize and diagnose the IO-Link Device.

The set of files consists of the main IODD file, optional language files and optional picture files.

An IODD is mandatory for each IO-Link Device. This specification defines the IODD for IO-Link Devices that conform to the IO-Link Communication Specification Version 1.0.

2 Related documents and references

2.1 References

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

IO-Link Communication Specification Version 1.0, January 2009, Order No: 10.002

ANSI/IEEE Std 754-1985, *IEEE Standard for Binary Floating-Point Arithmetic*

IETF RFC 2083, *PNG (Portable Network Graphics) Specification Version 1.0*, available at http://tools.ietf.org/html/rfc2083

ISO 639-1:2002, *Codes for the representation of names of languages – Part 1: Alpha-2 code*

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

ISO 15745-1:2003, *Industrial automation systems and integration – Open systems application integration framework – Part 1: Generic reference description*

ISO/IEC 61158-6-10:2007, *Industrial communication networks – Fieldbus specifications – Part 6-10: Application layer protocol specification – Type 10 elements*

The Unicode Standard, V5.0, available at http://www.unicode.org/

ITU-T recommendation V.42 (03/2002), *Error-correcting procedures for DCEs using asynchronous-to-synchronous conversion*, available at http://www.itu.int/rec/T-REC-V.42-200203-I/en

REC-xml-20081126, *Extensible Markup Language (XML) 1.0 Fifth Edition – W3C Recommendation 26 November 2008,* available at http://www.w3.org/TR/xml/

REC-xmlschema-1-20041028, *XML Schema Part 1: Structures – W3C Recommendation 28 October 2004,* available at http://www.w3.org/TR/xmlschema-1/

REC-xmlschema-2-20041028, *XML Schema Part 2: Datatypes – W3C Recommendation 28 October 2004,* available at http://www.w3.org/TR/xmlschema-2/

2.2 Related documents

ANSI INCITS 4-1986 (R2007), *Information Systems – Coded Character Sets – 7-Bit American National Standard Code for Information Interchange (7-Bit ASCII)* (predecessor of ISO/IEC 646)

IEC 60559:1989, *Binary floating-point arithmetic for microprocessor systems*

IETF RFC 1305, *Network Time Protocol (Version 3) Specification, Implementation and Analysis*, available at http://tools.ietf.org/html/rfc1305

IETF RFC 3629, *UTF-8, a transformation format of ISO 10646*, available at http://tools.ietf.org/html/rfc3629

ISO/IEC 3309:1993, *Information technology – Telecommunications and information exchange between systems – High-level data link control (HDLC) procedures – Frame structure* (withdrawn)

ISO 8601:2004, *Data elements and interchange formats – Information interchange – Representation of dates and times*

ISO/IEC 10646:2003/Amd 6:2009, *Information technology – Universal Multiple-Octet Coded Character Set (UCS)*

ISO/IEC 15948:2004, *Information technology – Computer graphics and image processing – Portable Network Graphics (PNG): Functional specification*

3 Definitions and abbreviations

3.1 Definitions

For the purposes of this document, the terms and definitions given in ISO 15745-1:2003 apply.

3.2 Abbreviated terms

ANSI	American National Standards Institute (http://www.ansi.org/)
ASCII	American Standard Code for Information Interchange (see ANSI INCITS 4-1986 (R2007) and the US variant of ISO/IEC 646:1991)
BIPM	Bureau International des Poids et Mesures (http://www.bipm.org/)
CRC	Cyclic Redundancy Check
IEC	International Electrotechnical Commision (http://www.iec.ch/)
IEEE	Institute of Electrical and Electronics Engineers (http://www.ieee.org/)
IETF	Internet Engineering Task Force (http://www.ietf.org/)
IO or I/O	Input / Output
IODD	IO-Link Device Description
ISO	International Standardization Organisation (http://www.iso.org/)
ITU	International Telecommunication Union (http://www.itu.int/)
MSXML	Microsoft XML Core Services (see http://msdn.microsoft.com/en-us/library/ms763742%28VS.85%29.aspx)
PDU	Protocol Data Unit
PLC	Programmable Logic Controller
PNG	Portable Network Graphics (see RFC 2083 and ISO/IEC 15948:2004)
RFC	Request for Comments

SPDU	Service PDU
UCS	Universal Multiple-Octet Coded Character Set (see *The Unicode Standard* or ISO/IEC 10646:2003/Amd 6:2009)
UTC	Continuous Universal Time (Temps Universel Coordonné) (coordinated by the BIPM)
UTF	UCS Transformation Format (see *The Unicode Standard* or ISO/IEC 10646:2003/Amd 6:2009)
W3C	World Wide Web Consortium (http://www.w3.org/)
XML	Extensible Markup Language (see REC-xml-20081126)
XSD	XML Schema Definition (see REC-xmlschema-1-20041028 and REC-xmlschema-2-20041028)

4 Basic structure

The following figure shows the basic structure of the main IODD file. It follows the ISO 15745-1:2003 standard regarding the device profile and communication network profile. It consists of the elements DocumentInfo, ProfileHeader, ProfileBody, CommNetworkProfile, ExternalTextCollection and the Stamp.

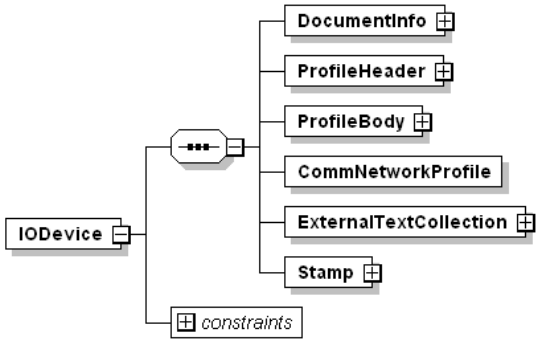


Figure 1 – Structure of main IODD file following ISO 15745-1

5 Files

Conceptionally, the IO-Link Device Description consists of the set of files created by the device vendor, and the set of standard definition files which are part of this specification. Engineering tools combine informations from both sets of files to get the complete device description.

All IODD XML files shall use the name space http://www.w3.org/2001/XMLSchema-instance with the prefix “xsi”and the name space http://www.io-link.com/IODD/2009/11 with the prefix “iodd”. A schemaLocation for the name space http://www.io-link.com/IODD/2009/11 to the required schema shall be given. For the main IODD file, this is IODD1.0.1.xsd, and for the language files this is IODD-Primitives1.0.1.xsd. The schema file name shall be given without any path prefix.

All XMLs generated by the vendor shall be checked by the IODD Checker software before delivery. This Checker is a tool available from the IO-Link web site (http://www.io-link.com/). It checks the content of the device description and if no errors were found writes a checksum over the file contents into the element Stamp at the end of the XML-file.

Engineering tools shall compare the checksum in the Stamp with the checksum calculated from the file contents. It is recommended to reject the IODD if there is a mismatch. Tools may then omit schema validation and additional checks.

Tools must not evaluate the file name; they always evaluate the file’s content. The device-specific file name is only intended for better legibility.

Adherence to the name guidelines makes it possible that all IODDs can be put in a single directory.

File names must not only be different in upper and lower case. Case sensitivity of default parts of file names shall be adhered to.

The following special characters are permitted in vendor name and device name: _, #, -

All files of the set of files belonging to a specific IODD shall have the same <vendor name> part in their file names. The <vendor name> should be the same for all IODDs of the same vendor.

5.1 Main IODD file

The file name shall follow the following rule:

<vendor name>**-**<device name>**-**<date of file creation>**-IODD**<schema version>**.xml**

e.g. VendorX-DeviceY-20080414-IODD1.0.1.xml

Contains information (in XML) about the identification of the device, communication characteristics, parameters, process data, and diagnosis data.

The IODD must always entirely contain texts in the PrimaryLanguage (English). The IODD may contain texts in further languages.

A style sheet for the vendor-specific description of IO-Link devices for a certain browser (optional); this style sheet shall not be referenced from the IODD file.

e.g. VendorX-IODD1.0.1.xsl

5.2 Language files (optional)

To add support for additional languages after an IODD has been released, separate language files (in XML) may be created. Their file name must exactly match name of the main IODD file, except that there is an additional language designation before the file name extension:

<vendor name>**-**<device name>**-**<date of file creation>**-IODD**<schema version>**-**<language>**.xml**

The “language” part follows ISO 639-1:2002. The “language” part shall correspond to the value of the xml:lang attribute inside the language file. There shall be no additional language file for languages already covered in the main IODD file.

The “Text” elements contained in the additional language file shall follow the same rules as specified for the “Text” elements in additional languages inside the main IODD.

e.g. VendorX-DeviceY-20080414-IODD1.0.1-ru.xml

Additional language file containing texts in Russian.

A tool or interpreter shall select the appropriate language from the main IODD file or the accompanying language files according to its user interface language settings.

5.3 Image files (optional)

The file format shall be PNG (file extension .png, see RFC 2083 or ISO/IEC 15948:2004). The same rules for permitted characters apply as in section ‘Files’ (see above).

<vendor name>**-logo.png**
Vendor logo. 160 x 90 pixel, landscape format. The background of the logo should be transparent.

<vendor name>**-**<picture name>**-icon.png**
Device icon. 48 x 48 pixel.

<vendor name>**-**<picture name>**-pic.png**
Device picture. Min. 160 x 160 pixel, max. 320 x 320, square.

The device icon and device picture are referenced from each device variant.

5.4 Standard definitions files

IODD-StandardDefinitions1.0.1.xml
This file contains the definition of standardized variables (see IO-Link Communication Specification) plus English language texts.

IODD-StandardDefinitions1.0.1-de.xml
Additional language file containing texts in German.

IODD-StandardUnitDefinitions1.0.1.xml
This file contains the definitions of all available unit codes plus English language texts.

IODD-StandardUnitDefinitions1.0.1-de.xml
Additional language file containing texts in German.

Those files are part of the standard and must not be changed. Vendors of tools should use those files instead of hard-coding standardized things.

Additional language files for standard definitions files will be provided by the IODD subteam when needed on the IO-Link website.

5.5 Schema files

Schema files are needed to validate the structure of XML-files and to aid in editing.

IODD1.0.1.xsd
IODD-schema; includes the following sub-schemas:

IODD-Primitives1.0.1.xsd
includes basic schema elements

IODD-Datatypes1.0.1.xsd
includes schema elements for the definition of data types

IODD-Events1.0.1.xsd
includes schema elements for the definition of events

IODD-Variables1.0.1.xsd
includes schema elements for the definition of variables

IODD-UserInterface1.0.1.xsd
includes schema elements for the definition of the user interface

IODD-Communication1.0.1.xsd
includes schema elements for the definition of the communication network profile

IODD-StandardDefinitions1.0.1.xsd
schema for the definition of system-specific elements used to validate the file IODD-StandardDefinitions1.0.1.xml and IODD-StandardUnitDefinitions1.0.1.xml.

6 Description mechanisms

6.1 Names of elements and attributes

Following one common pattern, the names of the elements begin with an uppercase letter while the names of the attributes begin with a lowercase letter. When names consist of several words, each word (except for the first in case of an attribute) starts with an uppercase letter. No separator character (like _) is used.

6.2 Ids

The values of the attribute “id” shall follow the regular expression pattern:

“[A-Za-z][A-Za-z0-9 _-]*[A-Za-z0-9]”.

6.3 Referencing

Each element that can be referenced within the IODD contains an explicit attribute “id”. The referencing element contains a type-dependent attribute with the following composition: <type>Id

Examples: textId, datatypeId, menuId, variableId

6.4 Text localization

All text components of the different languages which are referenced in the IODD are allocated in the ExternalTextCollection (for further information see “Language-Dependent Description Texts”).

The text components of the different languages are referenced in the relevant location according to a key (textId).

Further languages can be added in an appropriate file (see chapter 5.2).

The PrimaryLanguage in the IODD must be completely available. If there is a further language added in the IODD or in a separate language file, not all entries must be given. In this case, the interpreter has to go back to the entry of the PrimaryLanguage.

7 Device Description

Notation

(m) Mandatory attribute or element
(o) Optional attribute or element
(c) Use depends on … (see description)

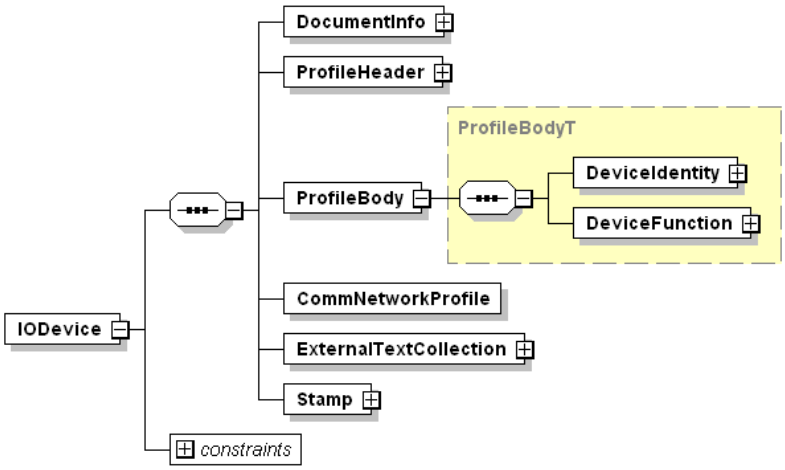


Figure 2 – Basic structure of main IODD file

The graph above shows the basic structure of an IO-Device in a device description.

7.1 Metainformation

7.1.1 DocumentInfo (m)

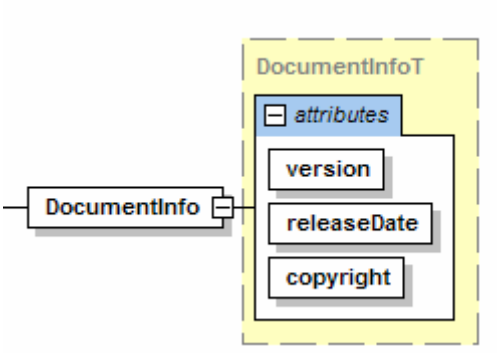


Figure 3 – DocumentInfo element

Here the vendor inserts the information for the IODD.

version (m)
The version attribute contains the version of the concrete instance and not the version of the IODD specification. The vendor shall increase this version for each official release of the IODD for a particular device.

releaseDate (m)
The date information in the IODD file name shall correspond to the releaseDate attribute in the DocumentInfo element. There shall be no more than one official release of the IODD for a particular device per day. Engineering tools shall rely on this date for determining the newest version of the IODD for a device.

copyright (m)
Vendor-specific copyright text.

e.g.
File name: IO-Link-SampleDevice-20090127-IODD1.0.1.xml

DocumentInfo:
<DocumentInfo version="V5.17" releaseDate="2009-01-27" copyright="IO-Link Consortium"/>

7.1.2 ProfileHeader (m)

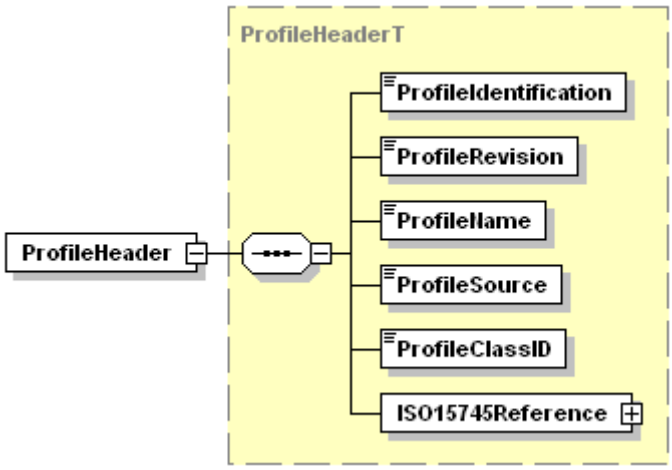


Figure 4 – ProfileHeader element

Within this element, the vendor is obliged to give the following constant information in plain text.

ProfileIdentification (m)
„IO-Link Device Profile“

ProfileRevision (m)
„1.00“

ProfileName (m)
„Device Profile for IO-Link Devices“

ProfileSource (m)
„IO-Link Consortium“

ProfileClassID (m)
„Device”

ISO15745Reference (m)
Information about the underlying ISO standard

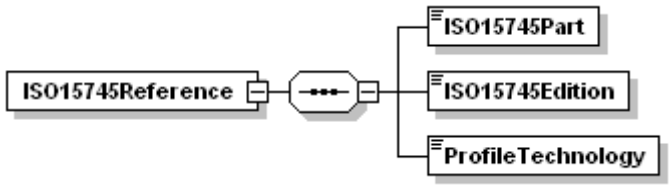


Figure 5 – ISO15745Reference element

ISO15745Part (m)
„1”

ISO15745Edition (m)
„1”

ProfileTechnology (m)
„IODD”

7.1.3 ProfileBody (m)

The ProfileBody contains the description of identity and functionality of the device.

7.1.4 File validation

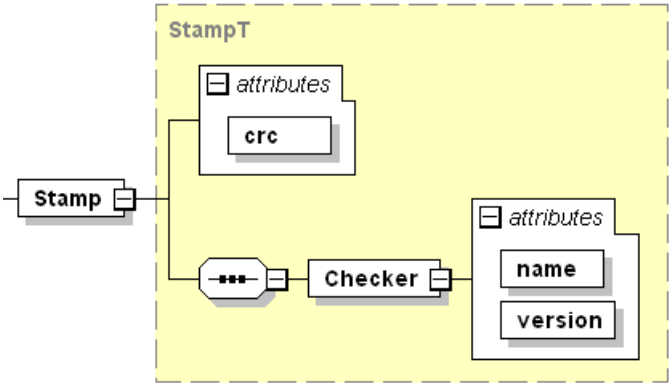


Figure 6 – Stamp element

There is a tool called “IODD Checker” that intensely tests the IODD if it is conformant to this specification. The Checker features a *check* and a *stamp* mode. In *check* mode, errors detected during the checking process are reported, but the file remains unchanged. In *stamp* mode, the Stamp element is always rewritten.

crc (m)
If no errors are detected during the checking process, the crc attribute is set to a CRC value calculated from the file contents. Otherwise, the crc attribute is set to an invalid value. By checking the CRC, an interpreter can find out whether the IODD has been altered since the last successful check. In this case, the IODD should be rejected by the interpreter.

For the CRC, the CRC-32 algorithm is used (see section 8.1.1.6.2 of ITU-T recommendation V.42 (03/2002) or ISO/IEC 3309:1993). Before the actual calculation, the crc attribute is set to an empty string and the checker inserts its name and version into the appropriate attributes. The generated CRC is then inserted into the crc attribute.

External text files are stamped in the same way, but the CRC of the main IODD file is also included in the calculation. Before the actual calculation, the crc attribute is set to an empty string and the checker inserts its name and version into the appropriate attributes. Only for CRC calculation the checker adds the CRC of the main IODD to the end of the external text file. The generated CRC is then inserted into the crc attribute.

Checker (m)
Identification of the IODD Checker version used to check and stamp this file. If there is a severe bug in a specific Checker version, or the method of calculating the CRC must be modified in the future, engineering tools are able to adapt to this based on the Checker name and version.

name (m)
The name of the IODD Checker.

version (m)
The version of the IODD Checker.

When writing a new IODD, before applying the IODD checker on it for the first time, it is recommended to set the attributes to the following values:

Stamp/@crc = "0"
Stamp/Checker/@name = "" (empty string)
Stamp/Checker/@version = "V0.0.0"

7.2 Device identity

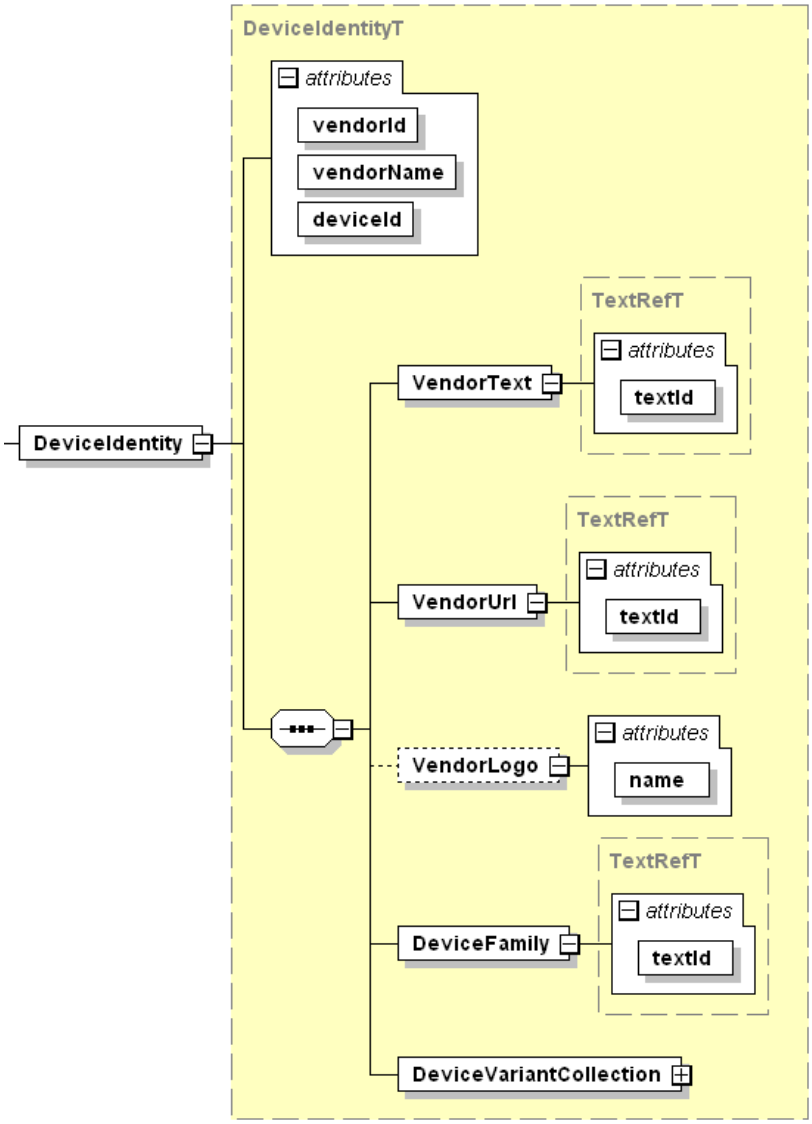


Figure 7 – DeviceIdentity element

vendorId (m)
Text; explicit identification of the vendor worldwide; a tool shall display this id in decimal notation

vendorName (m)
Name of the vendor of the device

deviceId (m)
Vendor-internal explicit identification of the device; a tool shall display this id in decimal notation

VendorText (m)
The vendor can use this text to describe himself

VendorUrl (m)
The vendor's URL

VendorLogo (o)
File name of the vendor's logo; in PNG format, 160 x 90 pixels

DeviceFamily (m)
Vendor-specific classification of the devices

7.2.1 Device variants

The different device variants are listed here. There shall be at least one device variant. Variants are uniquely identified by their productId.

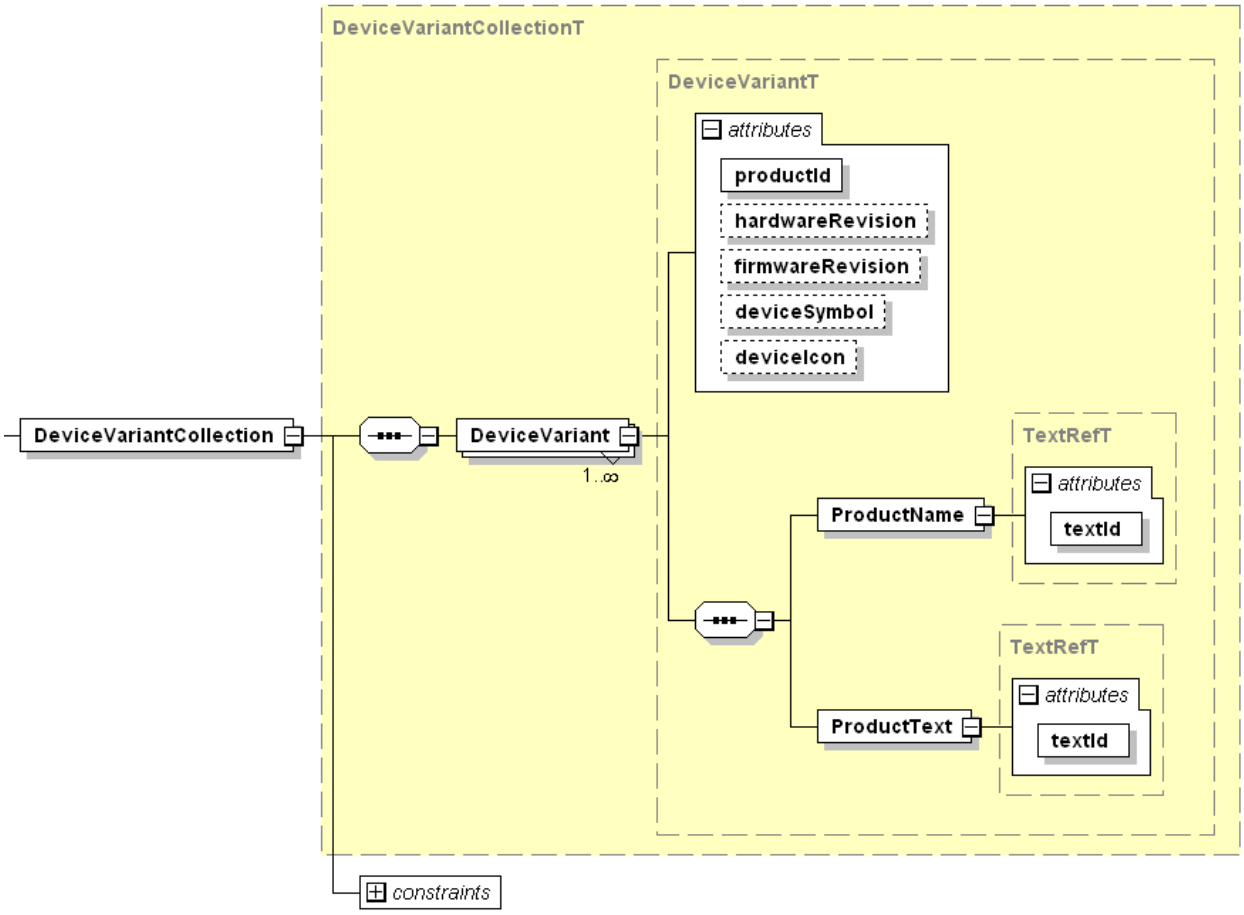


Figure 8 – DeviceVariantCollection element

productId (m)
Text; explicit item number within the description file

hardwareRevision (o)
Indicates that the description file was created for this device hardware version; only informative; it does not indicate functional changes

firmwareRevision (o)
Indicates that the description file was created for this device firmware version; only informative; it does not indicate functional changes

deviceSymbol (o)
File name of the device symbol

deviceIcon (o)
File name of the device icon

ProductName (m)
Text, uniquely identifies the product within the DeviceVariants. It shall correspond to the product name in the vendor's catalogue or to the name which is labelled on the product.

ProductText (m)
Descriptive text of the device

7.3 Device function

The entire functionality of the device is collected here. Parameters, process data, and events are defined. Their significances, addresses, and data fields are identified as well as a grouping of the views in menus is defined.

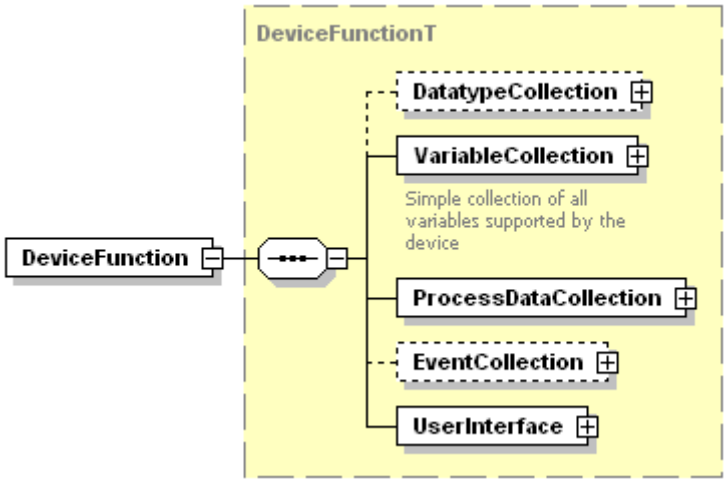


Figure 9 – DeviceFunction element

7.3.1 Declarations of data types

The DatatypeCollection incorporates all declarations for the reuse of data types (especially useful for Records). There shall be no unreferenced Datatype elements. Standardized data types are described in the schema IODD-Datatypes1.0.1.xsd.

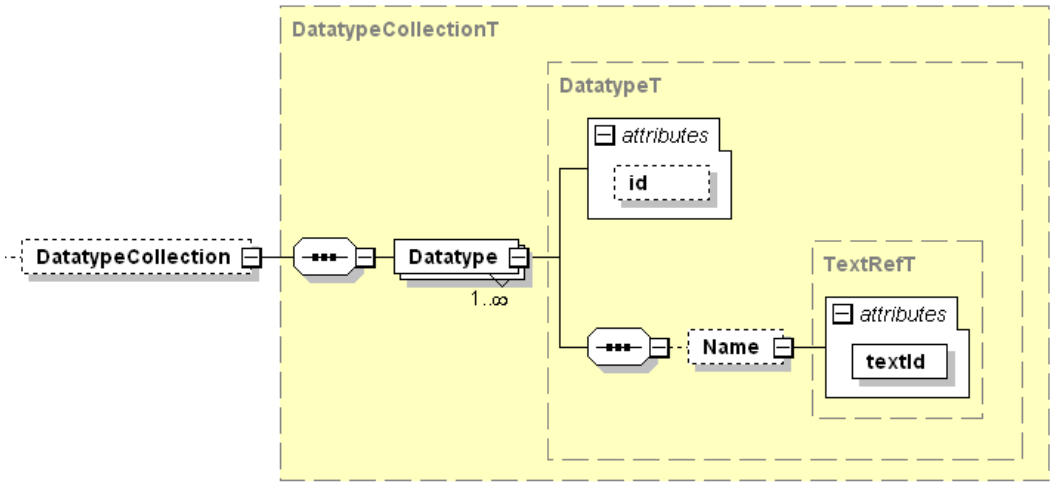


Figure 10 – DatatypeCollection element

For the Datatype element, this figure only shows the elements and attributes common to all data types. The actual selected data type needs additional elements and attributes. See chapter 8 for details.

For Datatypes in the DatatypeCollection, the attribute id shall be specified.

7.3.2 Variables

All parameters of the device are included here. Standard parameters are defined in IODD-StandardDefinitions1.0.1.xml and are referenced by StdVariableRef. StdDirectParameterRef

allows defining a Record which is being layed over the DirectParameterPage 2 (DirectParameters 16 – 31). All other device-specific variables are named under “Variable”.

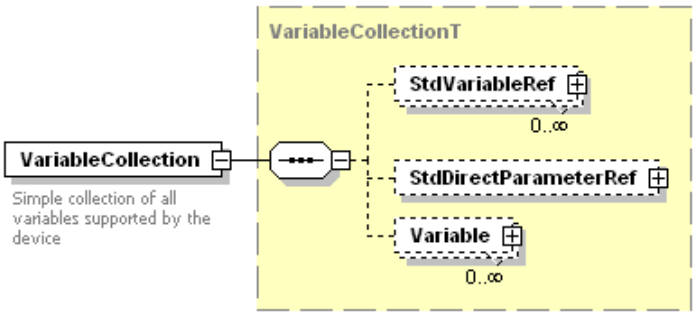


Figure 11 – VariableCollection element

7.3.2.1 StdVariableRef

Here it is described, which of the standard variables are used. They are referenced here by an explicit key. Because direct parameters are mandatory, the variables V_DirectParameters_1 and V_DirectParameters_2 shall always be referenced. All standard SPDU variables marked with the attribute mandatory=”true” in the IODD-StandardDefinitions shall be referenced if the device supports SPDU access.

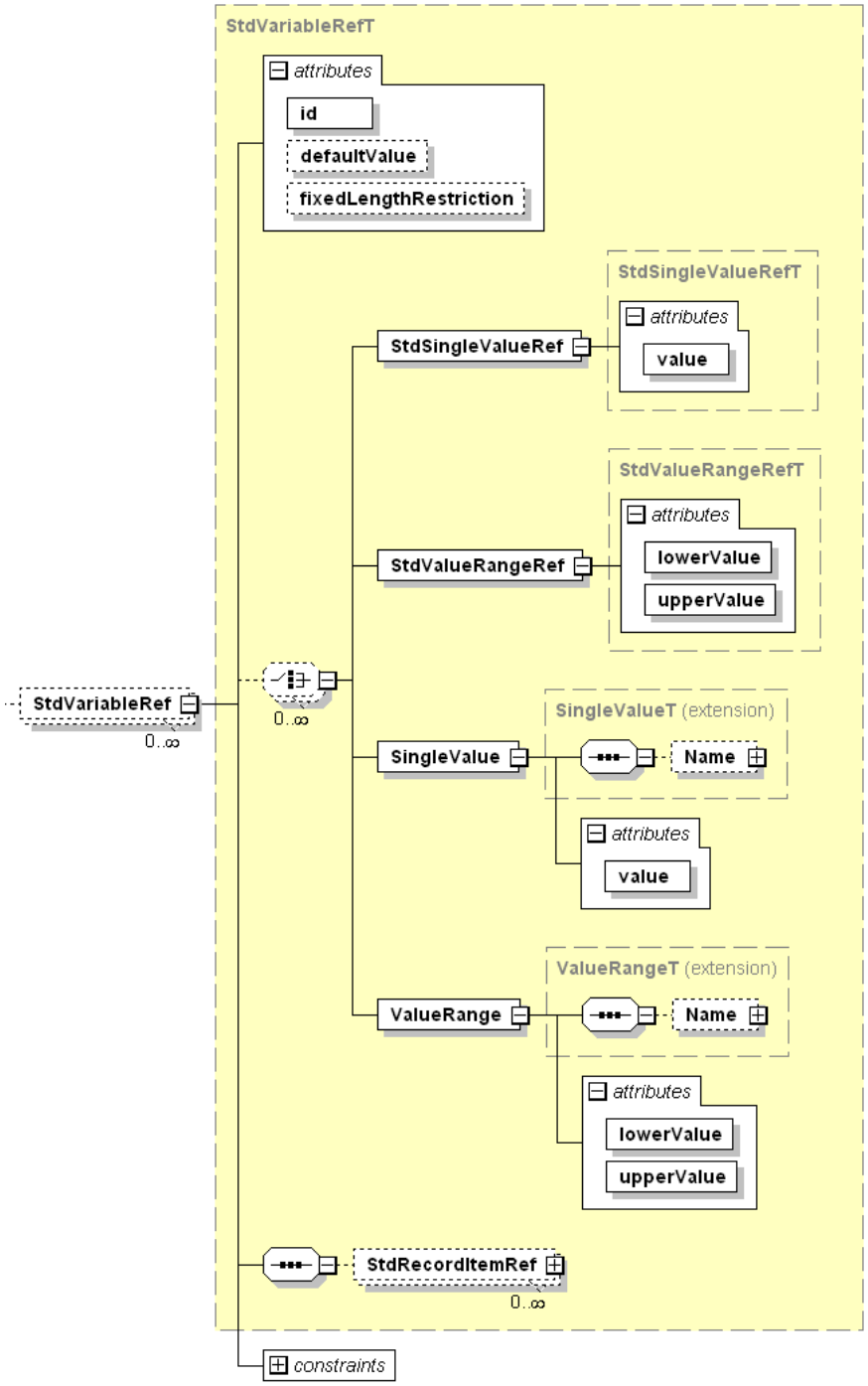


Figure 12 – StdVariableRef element

Id (m)
This id is special since it can be both starting and end point of a referencing process. As end point of the referencing process, it contains the key of those variables within the IODD. As starting point, it references to a standard variable.

defaultValue (o)
Contains the default value of the variable. For references to V_ProcessDataIn or V_Process-DataOut this attribute shall not be specified.

fixedLengthRestriction (o)
Contains the size limit of the variables

StdSingleValueRef (o)
Contains a reference to acceptable values the StdVariable can take on

StdValueRangeRef (o)
Contains a reference to the acceptable value range of the StdVariable

SingleValue (o)
Contains acceptable values with an assigned name (for more information see “ValueRange”)

ValueRange (o)
Specification of assigned ranges with name, lowerValue, and upperValue

- For each variable, both SingleValues and ValueRanges are permitted at the same time.
- Several ValueRanges must not overlap.
- SingleValues must not be located within a ValueRange.

StdRecordItemRef (o)
Same structure as StdVariableRef; the record items within a Record can be addressed via the subindex.

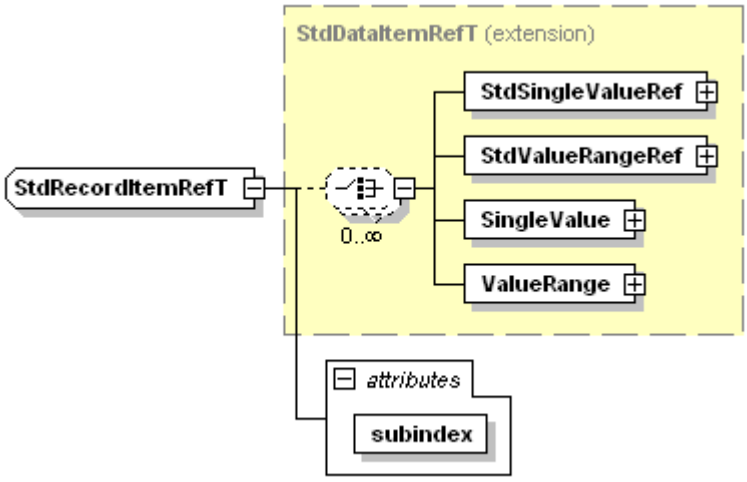


Figure 13 – StdRecordItemRef element

7.3.2.2 StdDirectParameterRef

This element corresponds to the device-specific data within the DirectParameter page.

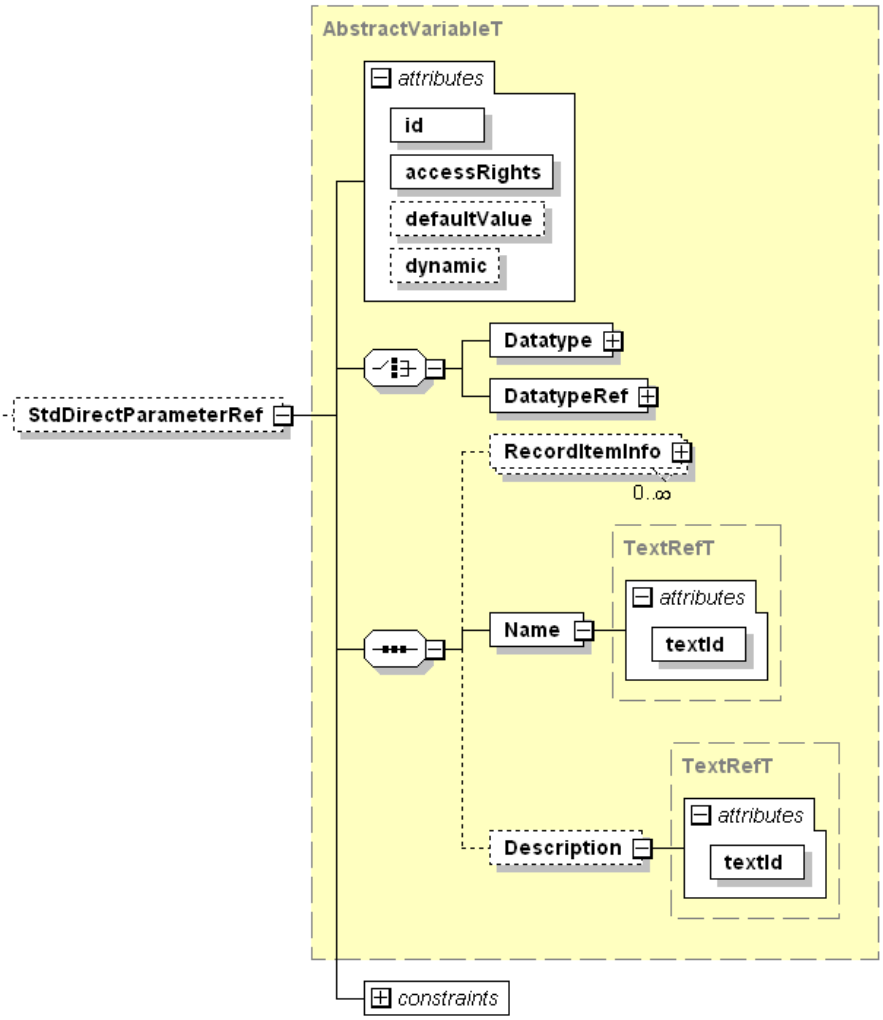


Figure 14 – StdDirectParameterRef element

id (m)
As the end point of a referencing process, it contains the key of the variable within the IODD

accessRights (m)
“ro” read-only,
“wo”, write-only,
“rw”, read-write

defaultValue (o)
This attribute shall not be used because the data type is fixed to record. Use RecordItemInfo element(s) to specify default values for individual record items.

dynamic (o)
Serves as information, whether the variable is autonomously changed by the device; the value range is true or false respectively

Datatype (c)
Directly given data type (see Note below)

DatatypeRef (c)
Reference to a data type that was defined in the DatatypeCollection (see Note below)

RecordItemInfo (o)
Contains a default value for a record item addressed by the subindex

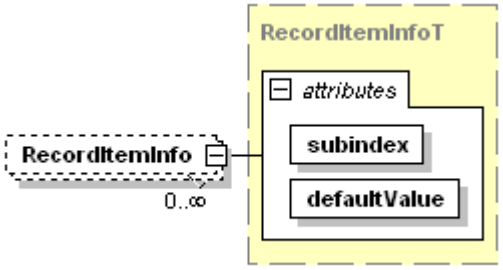


Figure 15 – RecordItemInfo element

Name (m)
Contains the name of the variable

Description (o)
Contains a description of the variable (e.g. information text, help, etc.)

Note: The data type shall be a record with a minimum length of 4 bytes and a maximum length of 16 bytes. The last byte of this record is mapped to the last byte of the direct parameter page 2. The last four bytes shall not be used by record items, because they are reserved. The first valid bitOffset is 32.

Example

RecordItem	Subindex	Datentyp	bitLength	bitOffset
1	1	UIntegerT	16	48
2	2	UIntegerT	16	32

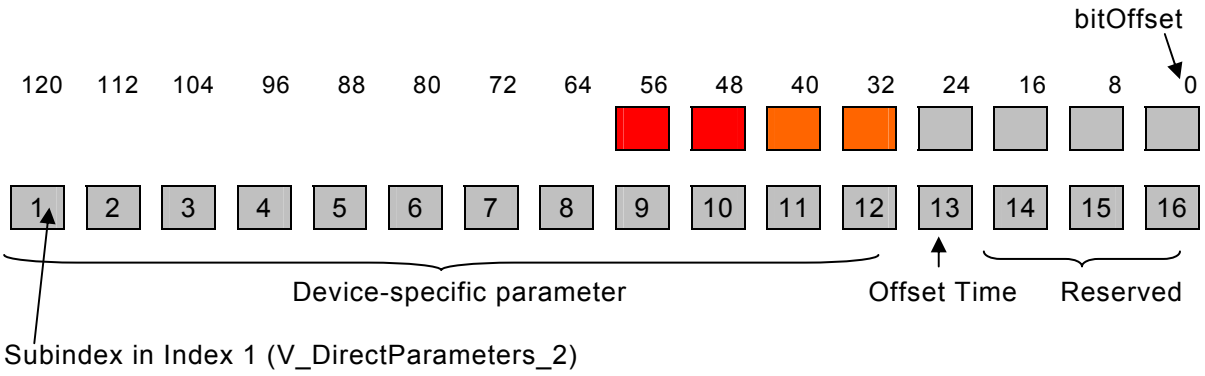


Figure 16 – Direct parameter overlay

Note: The communication of direct parameters is byte oriented.

- For record items, which cross a byte boundary the consistency cannot be guaranteed.
- If a byte contains more than one record item, the subindex access will influence all contained record items or parts.
- For record items, which cross a byte boundary, the device cannot rely on the order of the single accesses. This means, the device shall tolerate intermediate values that may exceed the allowed value range.

Recommendation: Use StdDirectParameterRef only for devices that do not support SPDU access.

7.3.2.3 Variable

Contains the description of a device parameter.

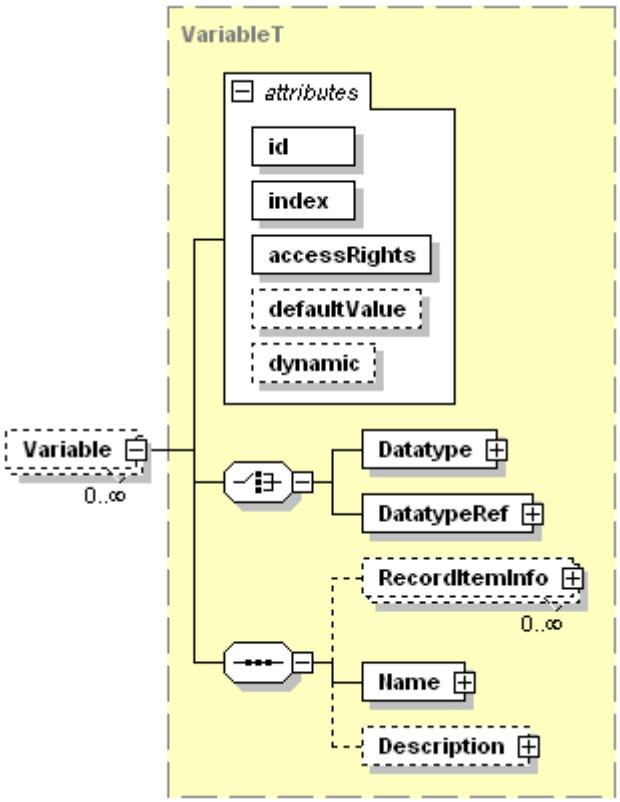


Figure 17 – Variable element

id (m)
As the end point of a referencing process, it contains the key of the variable within the IODD

index (m)
Index for the addressing of a variable

accessRights (m)
“ro” read-only,
“wo”, write-only,
“rw”, read-write

defaultValue (o)
Offline default value; it always refers to the complete variable. If the variable is a record, use RecordItemInfo element(s) to specify default values for individual record items. On a variable of type array, the specified defaultValue shall be applied to all array members.

dynamic (o)
Serves as information, whether the variable is autonomously changed by the device; the value range is true or false respectively

Datatype (c)
Directly given data type

DatatypeRef (c)
Reference to a data type that was defined in the DatatypeCollection

RecordItemInfo (o)
Only applicable if the variable is of type record. Contains a default value for a record item addressed by the subindex. See Figure 15.

Name (m)
Contains the name of the variable

Description (o)
Contains a description of the variable (e.g. information text, help, etc.)

7.3.3 Process data

Contains all process data of the device

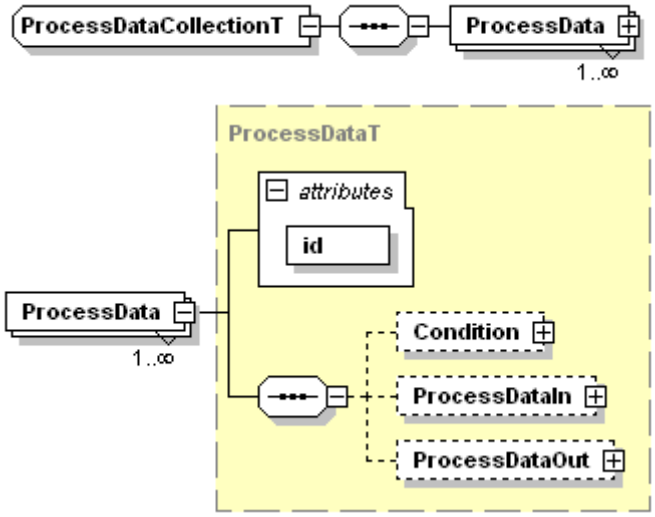


Figure 18 – ProcessDataCollection element

The element ProcessData consists of the elements Condition, ProcessDataIn, and ProcessDataOut. It can occur n times in the collection. If ProcessData occurs more than once, the individual ProcessData elements can be distinguished by to the Condition element.

The attribute ‘bitLength’ shall represent the underlying ProcesssDataIn (-Out) datatype in a bit granular manner. For record data types this bitLength shall equal the bitLength attribute of the record.

The value of the DirectParameterPage 1, Address 5 (Process Data In), shall be calculated from the bitLength-attribute value by the following formula:

```
If bitLength <= 16 then
        ProcessDataIn = bitLength
Else
        ProcessDataIn = bitLength rounded up to the next multiple of 8
End If
```

The same issue is valid for Process Data Out (DirectParameterPage 1, Address 6). The attribute “id” shall be unique within the elements ProcessData, ProcessDataIn and ProcessDataOut.

id (m)
Explicit id of the ProcessData

Condition (o)
Serves to switch between different ProcessData; Condition references the id of a variable. The assigned ProcessDataIn and ProcessDataOut descriptions are only valid if the variable takes on the value given in the attribute “value”. There shall only be exactly one variable used for the switching of process data. The referenced variable must contain a default value. The process data length (of ProcessDataIn and ProcessDataOut respectively) must be the same for all ProcessData.

Conditions shall only be of datatype IntegerT, UIntegerT and BooleanT.

The attribute Condition/@value can only hold values 0 – 255, thus limiting the possible IntegerT and UIntegerT values. Also, BooleanT condition values shall be entered as 0 for false and 1 for true.

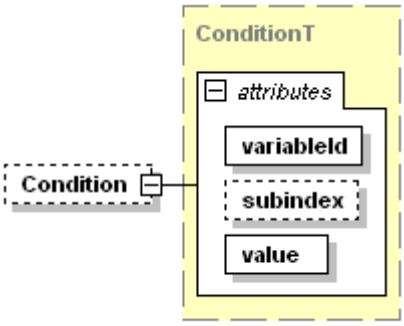


Figure 19 – Condition element

ProcessDataIn (o)
Description of the input process data

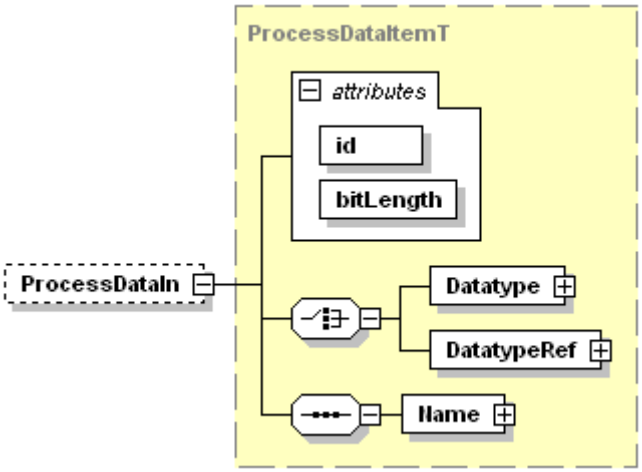


Figure 20 – ProcessDataIn element

id (m)
Explicit id of the ProcessDataIn description

bitLength (m)
Length of the input process data (in bits)

Name (m)
Name specification of the input process data

ProcessDataOut (o)
Description of the output process data

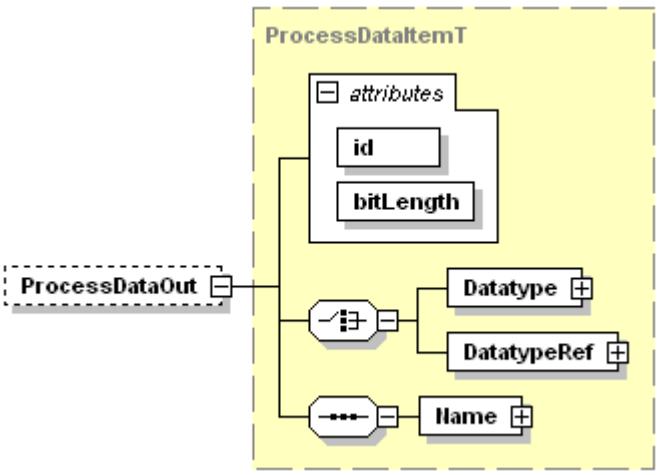


Figure 21 – ProcessDataOut element

id (m)
Explicit id of the ProcessDataOut description

bitLength (m)
Length of the output process data (in bits)

Name (m)
Name specification of the output process data

7.3.4 Events

Contains the event collection of the device

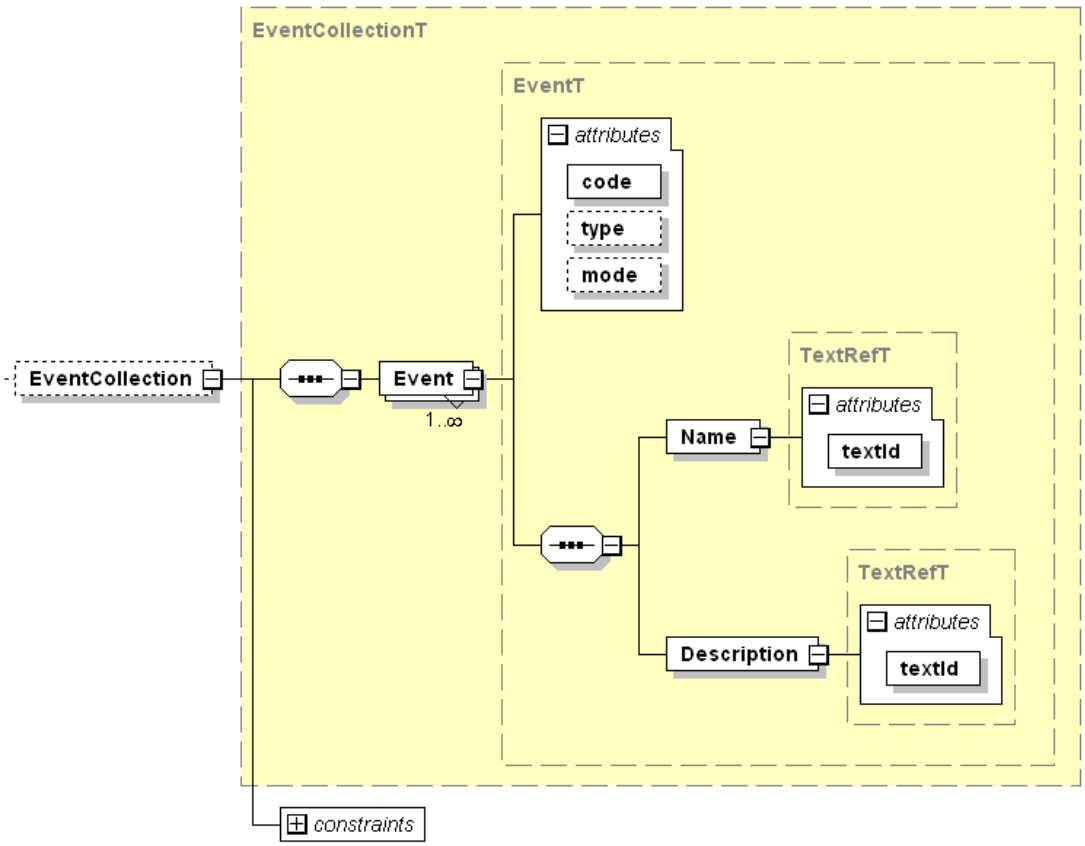


Figure 22 – EventCollection element

code (m)
type (o)
mode (o)
See IO-Link Communication Specification Annex B

Name (m)
Event name

Description (m)
Event description

7.3.5 User interface

Contains the menus of the device

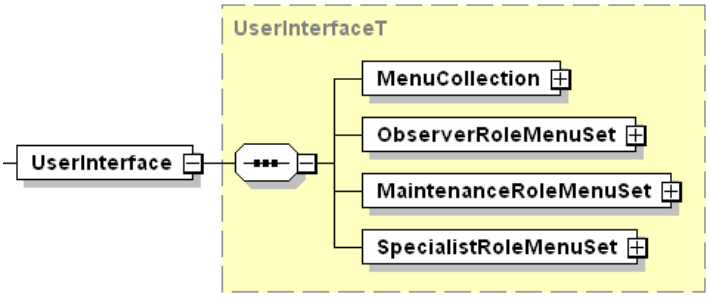


Figure 23 – UserInterface element

Roles
A UserInterface must always be divided into three roles. It is up to the vendor how the roles are organized. The tool must assign the entered UserLevel to the respective menu. At most three menu levels below the role assignment are acceptable.

Example:

ObservationRoleMenuSet
->IdentificationMenu
-> Menu1
-> MenuRef1
-> Menu2
-> MenuRef1
MaintenanceRoleMenuSet
->ObservationMenu
-> MenuX
-> MenuRefY

ObserverRoleMenuSet (m)
This menu is designed for users who may not carry out any modifications on the device.

MaintenanceRoleMenuSet (m)
This menu is designed for observers and to undertake “uncritical” editing. It is up to the vendor to assess that.

SpecialistRoleMenuSet (m)
If the user is logged in as a specialist, he/she has total access to the device. Again, the vendor can decide which parameters may be edited.

For each role, there is a definite set of menus given.

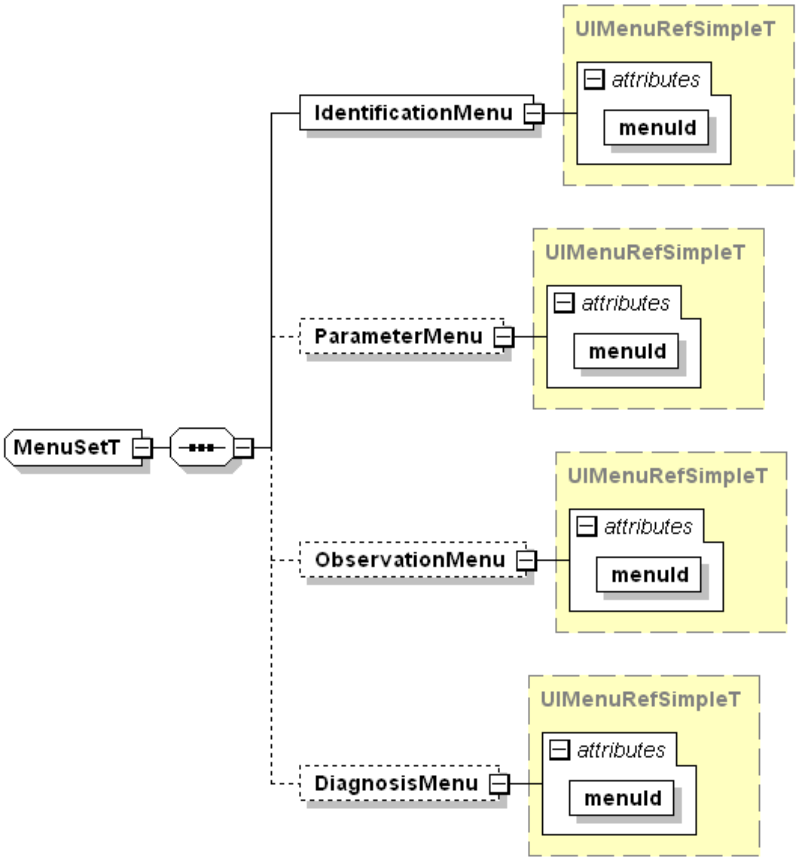


Figure 24 – <Role>MenuSet element

IdentificationMenu (m)
The attribute menuid references a menu from the MenuCollection. This menu should contain variables which serve the identification of the device.

ParameterMenu (o)
The attribute menuid references a menu from the MenuCollection. This menu should contain variables which serve the parameterization of the device.

ObservationMenu (o)
The attribute menuid references a menu from the MenuCollection. This menu should contain variables which serve the observation of the device (process data, dynamic variables, etc.).

DiagnosisMenu (o)
The attribute menuid references a menu from the MenuCollection. This menu should contain variables which serve the diagnosis of the device (events, etc.).

7.3.5.1 Menus

The names of top level menus, like IdentificationMenu, ParameterMenu, ObservationMenu or DiagnosisMenu are given from tooling. If a name is specified, it shall be ignored by tooling.

In underlying menus, a menu name shall be given by IODD.

MenuCollection (m)
All menu entries of the device are collected in the MenuCollection. These menu entries may be referenced by different roles (ObserverRole, MaintenanceRole, and SpecialistRole). There shall be no unreferenced Menu elements.

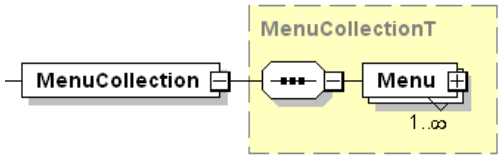


Figure 25 – MenuCollection element

Menu (m)
Variables, RecordItems and other menus may be referenced here.

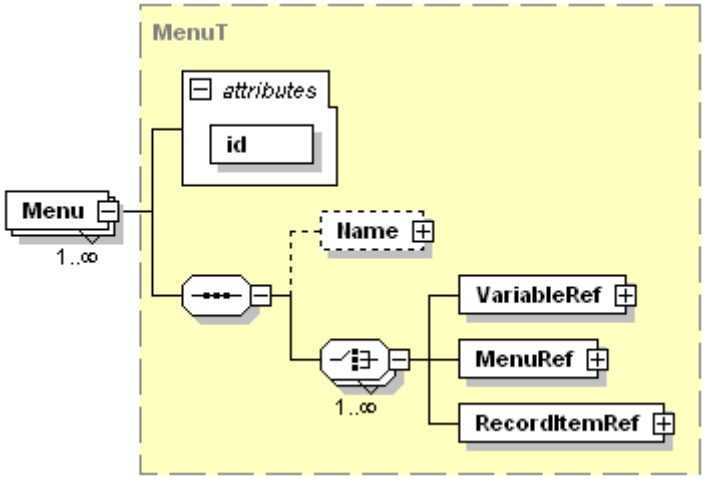


Figure 26 – Menu element

id (m)
Explicit id of the menu

Name (c)
Name of the menu. Top-level menus (i.e. those referenced from one of the MenuSets) may have a Name element, but it shall be ignored by engineering tools. Instead, hard-coded names shall be used by engineering tools. Nested menus shall have a Name element which is shown by engineering tools.

7.3.5.2 VariableRef

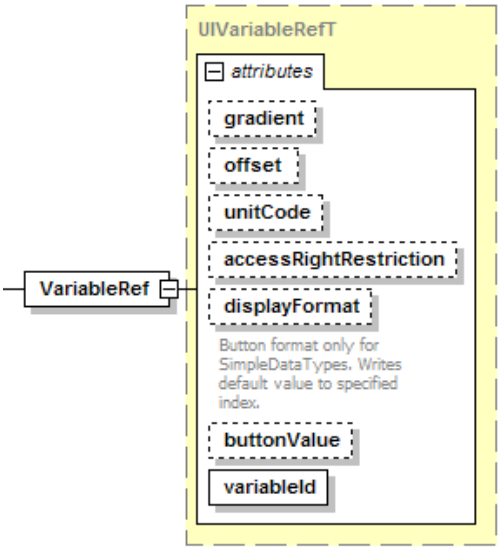


Figure 27 – VariableRef element

Regardless of the type of the referenced Variable or RecordItem, gradient and offset shall be specified as floating point values.

Displayed value = (value read from the IO-Link Device x gradient) + offset

When applying gradient and / or offset to convert the Variable or RecordItem value to the displayed value, the value will be implicitely converted to a floating point value. Consequently, the only allowed displayFormat on such values shall be “Dec”. (The displayFormat “Hex”, “Bin”, … does not force a conversion back to the original type of the Variable or RecordItem.)

When applying gradient and / or offset to convert an entered value back to the new value of a Variable or RecordItem, the resulting floating point value will be rounded to the nearest possible value of the type of the Variable or RecordItem.

Single array members can’t be referenced with RecordItemRef. If you need to access a single member, you have to define a record instead of an array.

A variable of type ArrayT can only be referenced as a whole, i.e. with VariableRef. All the attributes in VariableRef (gradient, offset, unitCode, accessRightRestriction, displayFormat and buttonValue) apply to each of the array members.

gradient (o)
Gradient of the indicated variables. When offset is specified and gradient is not specified, a value of 1.0 shall be used.

offset (o)
Zero-offset of the indicated variables. When gradient is specified and offset is not specified, a value of 0.0 shall be used.

unitCode (o)
Unit code to which the indicated variable refers. For valid unit codes see IODD-StandardUnitDefinitions1.0.1.xml.

accessRightRestriction (o)
For certrain UserRoles, the access rights may be limited here.

displayFormat (o)
Refers to the representation of variables; possible values are: Bin, Dec, Hex, Button, Event, MinCycleTime, MasterCycleTime. A tool or interpreter evaluates these specifications.

Definitions of the tool:

Dec:	Decimal notation without postfix, e.g. 23205
Hex:	Hexadecimal notation with postfix “h”, e.g. 5AA5h
Bin:	Binary notation with postfix “b”, e.g. 0101 1010 1010 0101b
Button:	Operation of the button sends the value of buttonValue to the device.
Event:	The value is to be interpreted as event, i.e. the text for the respective event qualifier and code is shown.
MinCycleTime:	The tool is able to calculate the correct time value from this byte
MasterCycleTime:	The tool is able to calculate the correct time value from this byte

A variable referenced as “Event” shall be a record consisting of an event qualifier and an event code. I.e. it shall have the following data type:

```
<Datatype xsi:type="RecordT" bitLength="24">
        <RecordItem subindex="1" bitOffset="16">
                <SimpleDatatype xsi:type="UIntegerT" bitLength="8"/>
                <Name textId="…"/>
        </RecordItem>
        <RecordItem subindex="2" bitOffset="0">
                <SimpleDatatype xsi:type="UIntegerT" bitLength="16"/>
                <Name textId="…"/>
```

```
            </RecordItem>
        </Datatype>
```

Note: The standard variable V_LastEvent has this data type.

The value of the ‘buttonValue’-attribute shall be defined as a ‘SingleValue’ of the Variable / RecordItem.

The label of the button shall be the name of the ‘SingleValue’.

A variable referenced as “Button”
shall have accessRights “wo”
shall only be displayed as a button
shall not be used as a condition variable, to switch menus or processdata.

The buttonValue
will be sent to the device immediately by pushing the button.
shall not be part of the block-download sequence.

buttonValue (o)
If the displayFormat is “Button”, the value indicated here is sent to the device when the button is clicked.

variableId (m)
Referenced variable

7.3.5.3 RecordItemRef

Corresponds to VariableRef with an additional subindex. The variable referenced by variableId shall be of type record. For displayFormat=”Button”, the referenced variable shall support subindex access.

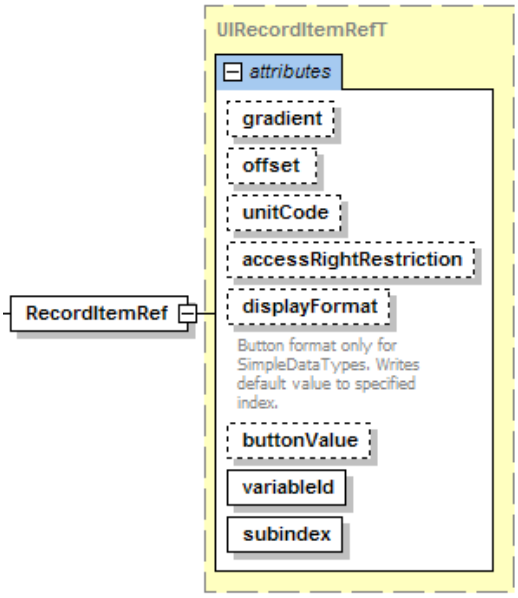


Figure 28 – RecordItemRef element

subindex (m)
Addresses the record item of a variable of type record

7.3.5.4 MenuRef

Reference to a (sub)menu nested inside this menu

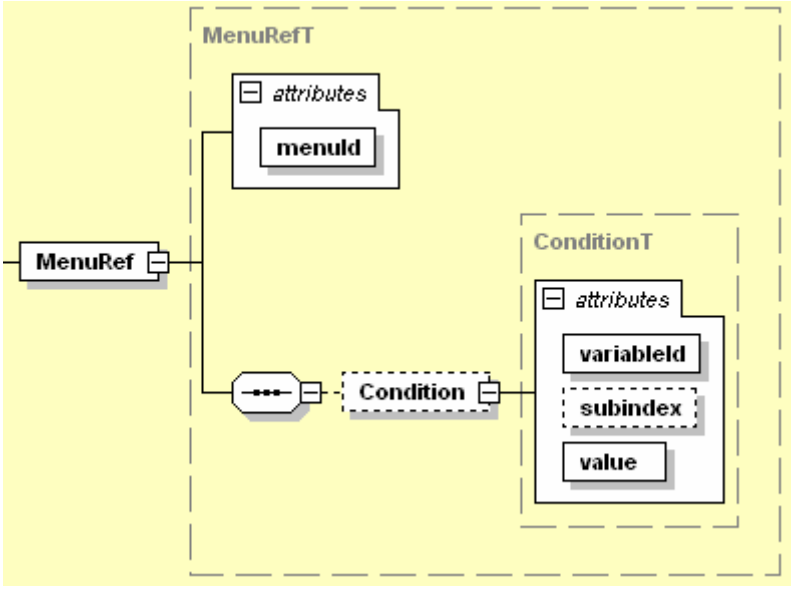


Figure 29 – MenuRef element

menuId (m)
References the (sub)menu from the MenuCollection

Condition (o)
Condition for the display of this menu; a tool or interpreter only shows the referenced menu if the variable with the reference *variableId* has the value *value*. If *variableId* contains a record, *subindex* shall be given additionally.

Conditions shall only be of datatype IntegerT, UIntegerT and BooleanT.

The attribute Condition/@value can only hold values 0 – 255, thus limiting the possible IntegerT and UIntegerT values. Also, BooleanT condition values shall be entered as 0 for false and 1 for true.

Conditions can be used in all menu levels.

If there is more than one ProcessData element, selected by conditions, and the variable V_ProcessDataIn or V_ProcessDataOut is referenced in a menu, one of the following shall hold:

- The type of reference (VariableRef / RecordItemRef) and the gradient, offset, unitCode and displayFormat fit to each of the ProcessData elements.
- The menu is conditioned in the same way as one of the ProcessData elements, and the type of reference (VariableRef / RecordItemRef) and the gradient, offset, unitCode and displayFormat fit to this particular ProcessData element.

“Conditioned in the same way” means that this or one of the parent menus has the same condition (same variable, same subindex, same value).

7.4 Communication characteristics

Figure 30 – CommNetworkProfile element

Excursion on XML schema *abstract types*:
An abstract type can’t be used itself. Only non-abstract types which are derived from an abstract type can be used. The instance selects the desired derived type with xsi:type=”*name of the derived type*”.

This technique is used here with the 'CommNetworkProfile' element to adapt the XML structure to the requirements of the specific communication. This allows easy extension of the IODD to non-IO-Link devices with different communication characteristics as long as the applicative concept remains the same (i.e. addressing via index/subindex, standardized variables).

For IO-Link, the following derived type IOLinkCommNetworkProfileT describes the communication characteristics of an IO-Link interface.

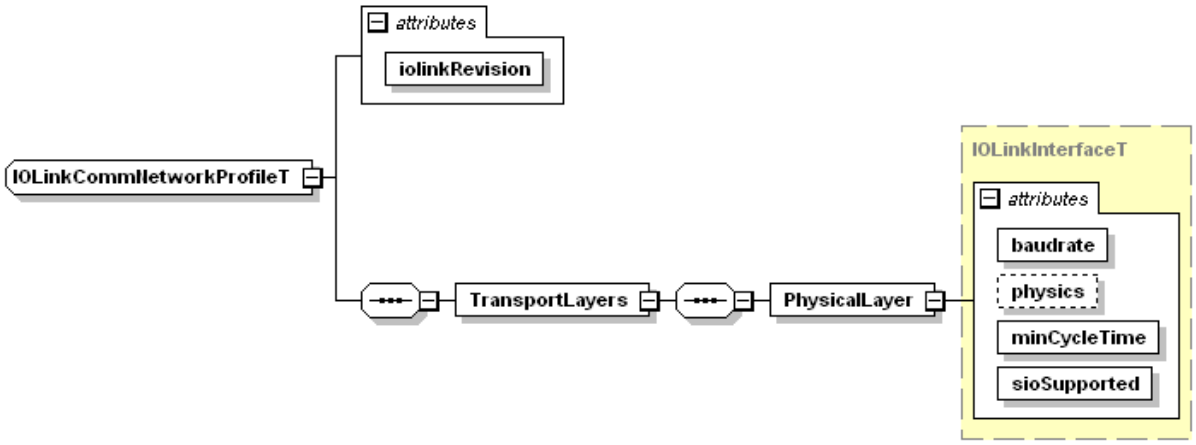


Figure 31 – CommNetworkProfile element – IO-Link variant

iolinkRevision (m)
Identifies the implemented protocol version in the format "Vx.y"

TransportLayers (m)

PhysicalLayer (m)

baudrate (m)
"COM1", "COM2" or "COM3"

physics (o)
2

minCycleTime (m)
The minimum cycle time of the slave; specified in 1 µs units. E.g. 2300 represents 2.3 ms. For the allowed values, refer to chapter 'Min Cycle Time' in the IO-Link Communication Specification.

sioSupported (m)
Whether the fall-back to SIO mode is supported.

7.5 Language dependent description texts

All text components of the different languages are given in the ExternalTextCollection. There may be one or more languages deposited. Additional languages may be stored in separate files.

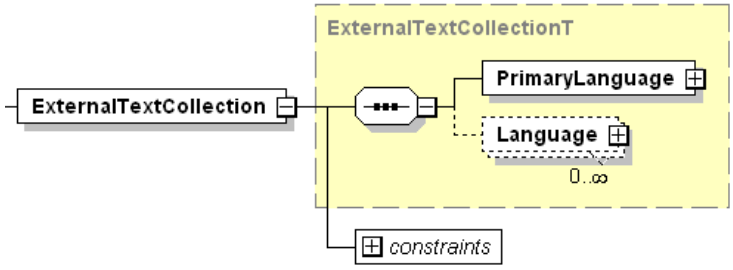


Figure 32 – ExternalTextCollection element

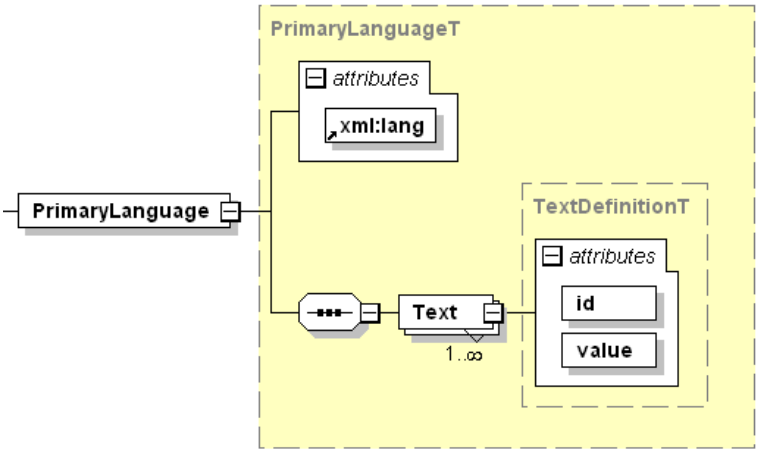


Figure 33 – PrimaryLanguage element

PrimaryLanguage (m)
Shall be in English (the attribute xml:lang shall have the value “en”).

Text (m)
Language dependent text which is referenced by its id.

id (m)
Shall be referenced by other elements via their textId attribute (there shall be no unreferenced Text elements)

value (m)
text in the denoted language

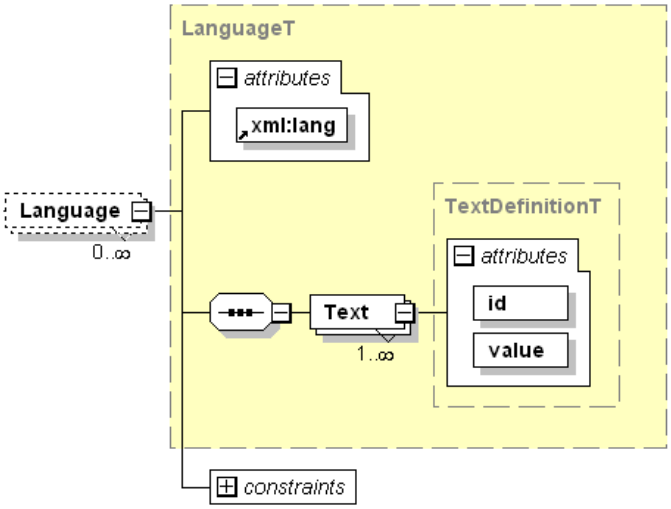


Figure 34 – Language element

Language (o)
Optional specification of texts in another language. The attribute xml:lang specifies the language (see ISO 639-1:2002). The structure of this element corresponds to the structure of the element PrimaryLanguage.

8 Data types

8.1 Declaration of data types

Excursion on XML schema *abstract types*:
An abstract type can’t be used itself. Only non-abstract types which are derived from an abstract

type can be used. The instance selects the desired derived type with xsi:type="*name of the derived type*".

This technique is used here with the 'Datatype' element to adapt the XML structure to the requirements of the specific data type.

Examples:

For an array declaration, one needs a base data type and a count.

For an integer data type, one needs the number of bits and a list of SingleValues and ValueRanges.

The IODD-Datatypes1.0.1.xsd schema provides derived types for all possible data types. The presence and type of required elements and attributes is checked by this schema. This is a much better approach than to have a single definition in the schema that lists all possible elements and attributes as optional and then let the checker find out whether the right elements/attributes are used with each data type.

Actually, the data types form the following hierarchy:

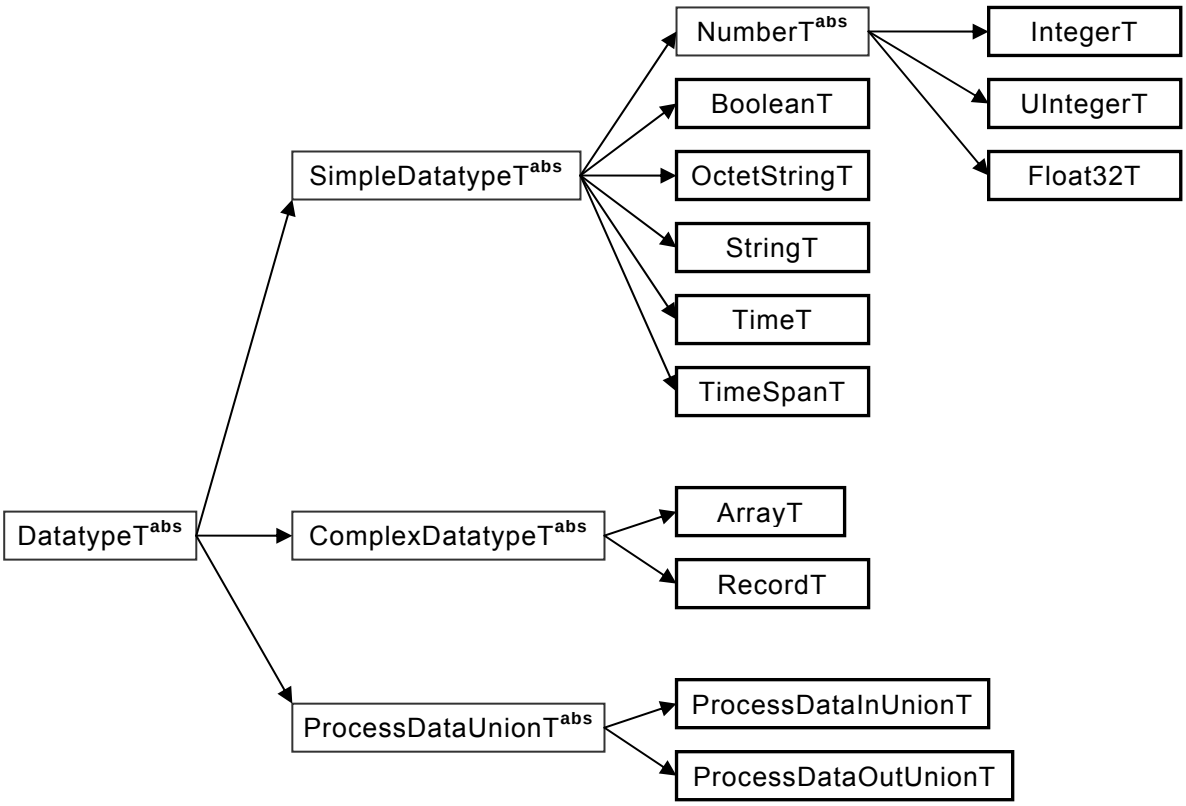


Figure 35 – Data type hierarchy

Each derivation adds elements and/or attributes appropriately.

The types ProcessDataInUnionT and ProcessDataOutUnionT are restricted to the description of the process data standard variables (Index 40 and 41) in IODD-StandardDefinitions1.0.1.xml and thus are not allowed in a normal IODD. The engineering tool shall take the data type of the appropriate ProcessDataIn / ProcessDataOut element. If more than one ProcessDataIn /

ProcessDataOut element is given, it is necessary to select the currently valid element by evaluating the Condition elements.

SingleValue and ValueRange elements are also strongly typed.

Where SingleValue and / or ValueRange elements are permitted, the following shall hold:

- When neither SingleValue nor ValueRange elements are given, the complete value range of the data type is allowed. When SingleValue(s) or ValueRange(s) are given, only these values are allowed.
- In ValueRanges, both the lowerValue and the upperValue are included in the range of allowed values.
- SingleValues and ValueRanges shall not overlap.

Where referencing standard variables with StdVariableRef, the following shall hold:

- When neither SingleValue nor ValueRange nor StdSingleValueRef nor StdValueRangeRef elements are given, the standard variable's value range as defined in IODD-StandardDefinitions1.0.1.xml is taken.
- When SingleValue(s) or ValueRange(s) or StdSingleValueRef(s) or StdValueRangeRef(s) are given, only these values are allowed.
- For each StdSingleValueRef there shall exist a SingleValue with the same value at the standard variable.
- For each StdValueRangeRef there shall exist a ValueRange with the same lowerValue and upperValue at the standard variable.
- SingleValues and ValueRanges shall not overlap with each other.
- SingleValues and ValueRanges shall not overlap with SingleValues and ValueRanges at the standard variable, no matter whether these are referenced by StdSingleValueRef and StdValueRangeRef or not.

When the Datatype appears inside the DatatypeCollection, the attribute 'id' shall be present. Otherwise, the attribute 'id' shall not be present.

The element 'Name' can be used to assign a human readable name to a data type. This name may be displayed additionally by the engineering system.

8.2 Simple data types

8.2.1 General

The coding of simple data types is shown only for singular use which is characterized by

- Process data consisting of one simple data type
- Parameter consisting of one simple data type
- Subindex (>0) access on individual data items of parameters of complex data types (arrays, records)

8.2.2 BooleanT

A BooleanT is representing a data type that can have only two different values i.e. TRUE and FALSE. The data type is defined in Table 1. For singular use the coding is shown in Table 2. A sender shall always use 0xFF for 'TRUE' or 0x00 for 'FALSE'. A receiver can interpret the range from 0x01 through 0xFF for 'TRUE' and shall interpret 0x00 for 'FALSE' to simplify implementations. The packed form is demonstrated in the examples in chapter 8.3.

Table 1 – BooleanT

Data type name	Value range	Resolution	Length
BooleanT	true / false	-	1 octet

Table 2 – BooleanT coding

Bit	7	6	5	4	3	2	1	0	Values
true	1	1	1	1	1	1	1	1	0xFF
false	0	0	0	0	0	0	0	0	0x00

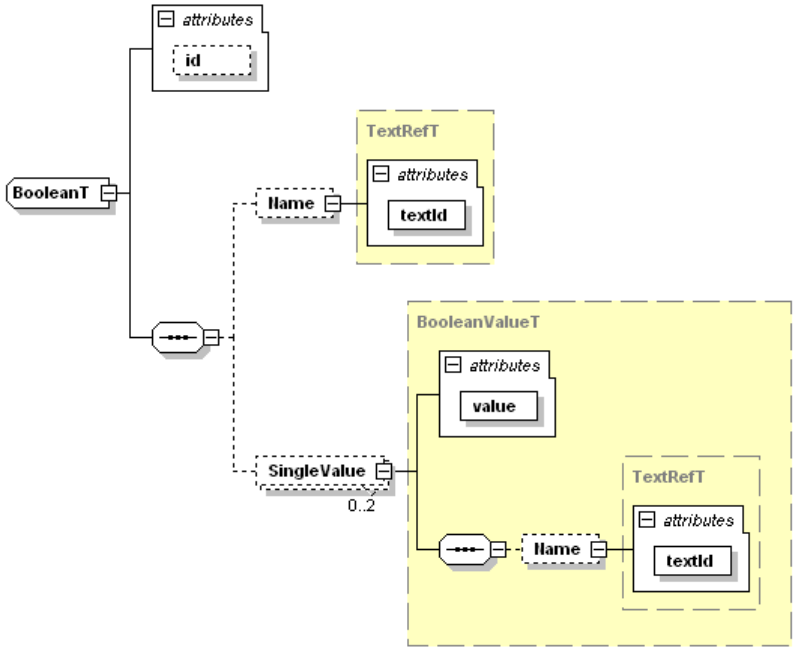


Figure 36 – BooleanT

Example:

```
<Datatype xsi:type="BooleanT">
        <SingleValue xsi:type="BooleanValueT" value="false">
                <Name textId="TI_Inversion_Off"/>
        </SingleValue>
        <SingleValue xsi:type="BooleanValueT" value="true">
                <Name textId="TI_Inversion_On"/>
        </SingleValue>
</Datatype>
```

Lexical representation:

Conforms to the representation of “boolean” in XML Schema, see http://www.w3.org/TR/xmlschema-2/#boolean

Regular expression pattern: “true|false|1|0”

8.2.3 UIntegerT

A UIntegerT is representing an unsigned number depicted by 2 up to 64 bits ("enumerated"). The number is accommodated and right-aligned within the following permitted octet containers: 1, 2, 4, or 8. High order padding bits are filled with "0". Coding examples are shown in Figure 37.

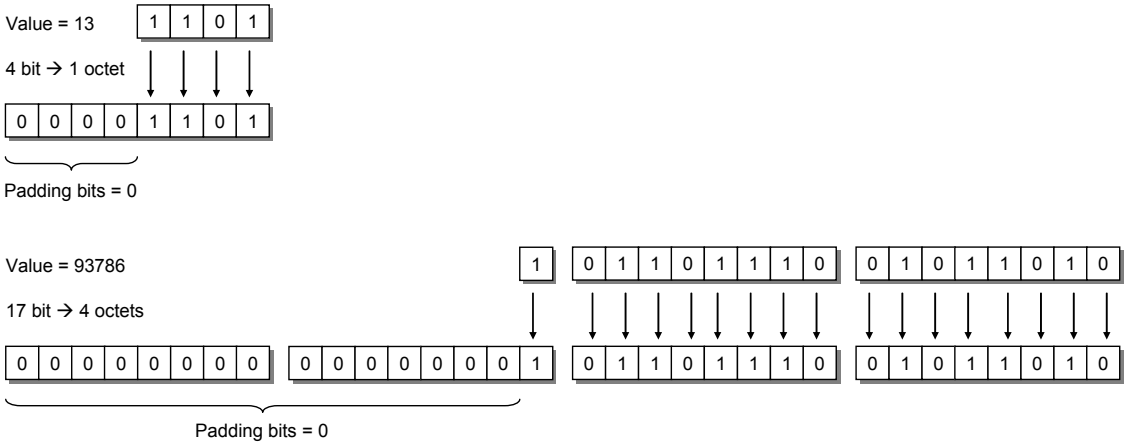


Figure 37 – Coding examples of UIntegerT

The data type UIntegerT is defined in Table 3 for singular use.

Table 3 – UIntegerT

Data type name	Value range	Resolution	Length
UIntegerT	$0 \ldots 2^{bitLength} - 1$	1	1 octet, or 2 octets, or 4 octets, or 8 octets
NOTE 1 High order padding bits are filled with "0" NOTE 2 Most significant octet (MSO) sent first			

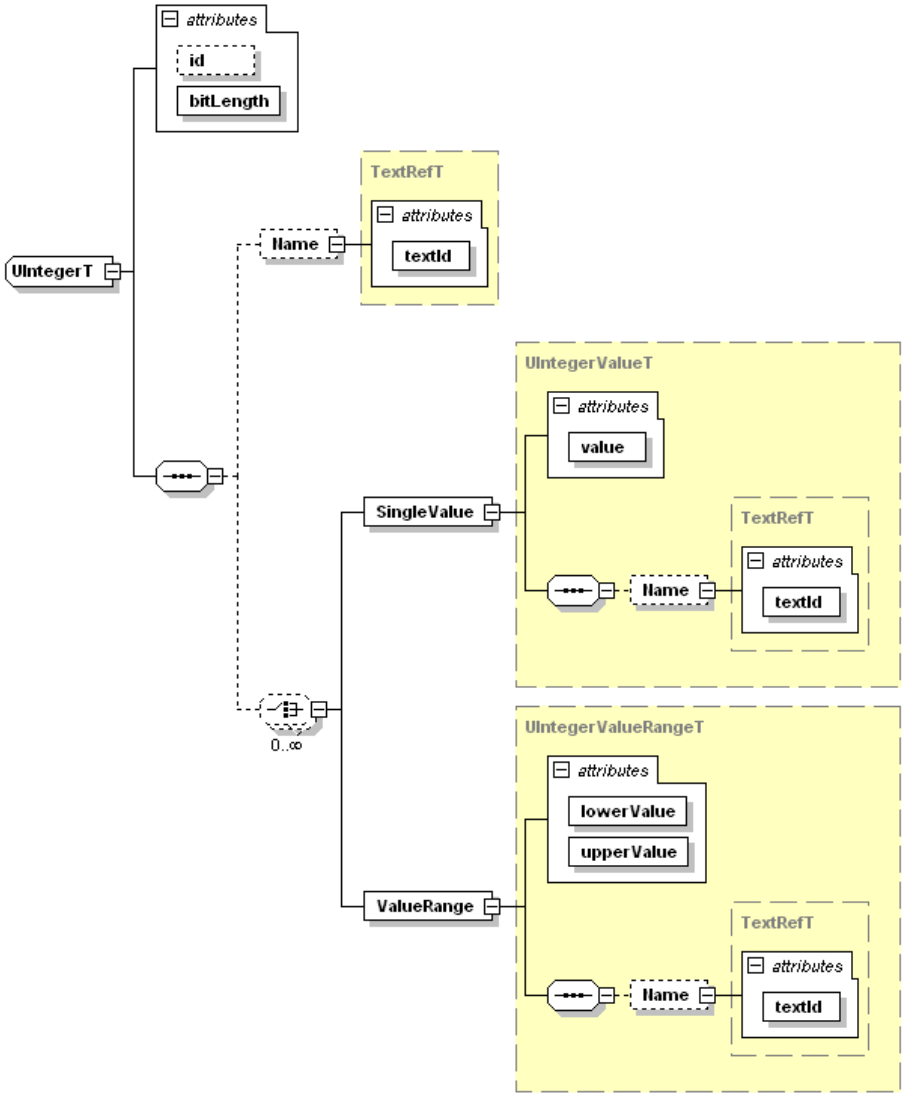


Figure 38 – UIntegerT

The attribute bitLength determines the size of the unsigned integer. Allowable values for bitLength are 2 – 64.

Example:

```
<Datatype xsi:type="UIntegerT" bitLength="8">
        <SingleValue xsi:type="UIntegerValueT" value="96">
                <Name textId="TI_System"/>
        </SingleValue>
</Datatype>
```

Lexical representation:

Conforms to the representation of “unsignedLong” in XML Schema, see http://www.w3.org/TR/xmlschema-2/#unsignedLong

Regular expression pattern: “+?\d+”

8.2.4 IntegerT

An IntegerT is representing a signed number depicted by 2 up to 64 bits. The number is accommodated within the following permitted octet containers: 1, 2, 4, or 8 and right-aligned and extended correctly signed to the chosen number of bits. The data type is defined in Table 4 for singular use. SN represents the sign with "0" for all positive numbers and zero, and "1" for all negative numbers. Padding bits are filled with the content of the sign bit (SN).

Table 4 – IntegerT

Data type name	Value range	Resolution	Length
IntegerT	$-2^{bitLength\ -1}$ ... $2^{bitLength\ -1} - 1$	1	1 octet, or 2 octets, or 4 octets, or 8 octets
NOTE 1 High order padding bits are filled with the value of the sign bit (SN) NOTE 2 Most significant octet (MSO) sent first (lowest respective octet number in Table 5)			

The 4 coding possibilities in containers are shown in Table 5 through Table 8.

Table 5 – IntegerT coding (8 octets)

Bit	7	6	5	4	3	2	1	0	Container
Octet 1	SN	2^{62}	2^{61}	2^{60}	2^{59}	2^{58}	2^{57}	2^{56}	8 octets
Octet 2	2^{55}	2^{54}	2^{53}	2^{52}	2^{51}	2^{50}	2^{49}	2^{48}	
Octet 3	2^{47}	2^{46}	2^{45}	2^{44}	2^{43}	2^{42}	2^{41}	2^{40}	
Octet 4	2^{39}	2^{38}	2^{37}	2^{36}	2^{35}	2^{34}	2^{33}	2^{32}	
Octet 5	2^{31}	2^{30}	2^{29}	2^{28}	2^{27}	2^{26}	2^{25}	2^{24}	
Octet 6	2^{23}	2^{22}	2^{21}	2^{20}	2^{19}	2^{18}	2^{17}	2^{16}	
Octet 7	2^{15}	2^{14}	2^{13}	2^{12}	2^{11}	2^{10}	2^{9}	2^{8}	
Octet 8	2^{7}	2^{6}	2^{5}	2^{4}	2^{3}	2^{2}	2^{1}	2^{0}	

Table 6 – IntegerT coding (4 octets)

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

Table 7 – IntegerT coding (2 octets)

Bit	7	6	5	4	3	2	1	0	Container
Octet 1	SN	2^{14}	2^{13}	2^{12}	2^{11}	2^{10}	2^{9}	2^{8}	2 octets
Octet 2	2^{7}	2^{6}	2^{5}	2^{4}	2^{3}	2^{2}	2^{1}	2^{0}	

Table 8 – IntegerT coding (1 octet)

Bit	7	6	5	4	3	2	1	0	Container
Octet 1	SN	2^{6}	2^{5}	2^{4}	2^{3}	2^{2}	2^{1}	2^{0}	1 octet

Coding examples within containers are shown in chapter 8.3.

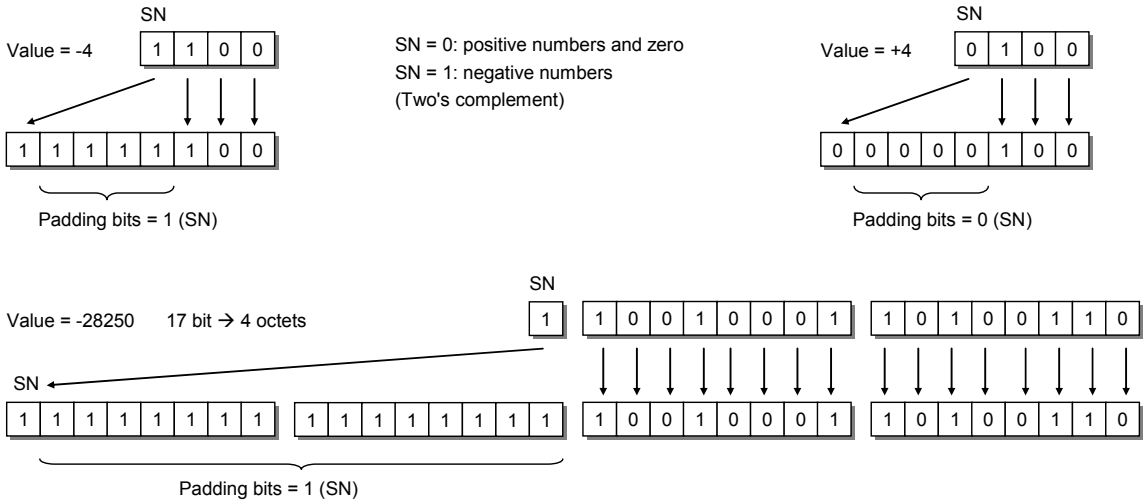


Figure 39 – Coding examples of IntegerT

For the representation in the IODD and an example see 8.2.3.

Lexical representation:

Conforms to the representation of “long” in XML Schema, see http://www.w3.org/TR/xmlschema-2/#long

Regular expression pattern: “[+-]?\d+”

8.2.5 Float32T

A Float32 is representing a number defined by ANSI/IEEE Std 754-1985 as single precision (32 bit). Table 9 shows the definition and Table 10 the coding. SN represents the sign with "0" for all positive numbers and zero, and "1" for all negative numbers.

Table 9 – Float32T

Data type name	Value range	Resolution	Length
Float32T	Refer to ANSI/IEEE Std 754-1985	Refer to ANSI/IEEE Std 754-1985	4 octets

Table 10 – Coding of Float32T

Bits	7	6	5	4	3	2	1	0
Octet 1	SN	Exponent (E)						
	2^0	2^7	2^6	2^5	2^4	2^3	2^2	2^1
Octet 2	(E)	Fraction (F)						
	2^0	2^{-1}	2^{-2}	2^{-3}	2^{-4}	2^{-5}	2^{-6}	2^{-7}
Octet 3	Fraction (F)							
	2^{-8}	2^{-9}	2^{-10}	2^{-11}	2^{-12}	2^{-13}	2^{-14}	2^{-15}
Octet 4	Fraction (F)							
	2^{-16}	2^{-17}	2^{-18}	2^{-19}	2^{-20}	2^{-21}	2^{-22}	2^{-23}

In order to realize negative exponent values a special exponent encoding mechanism is set in place as follows:

The Float32T exponent (E) is encoded using an offset binary representation, with the zero offset being 127; also known as exponent bias in the ANSI/IEEE Std 754-1985 standard.

E_{min} = 0x01 – 0x7F = –126

E_{max} = 0xFE – 0x7F = 127

Exponent bias = 0x7F = 127

Thus, as defined by the offset binary representation, in order to get the true exponent the offset of 127 has to be subtracted from the stored exponent.

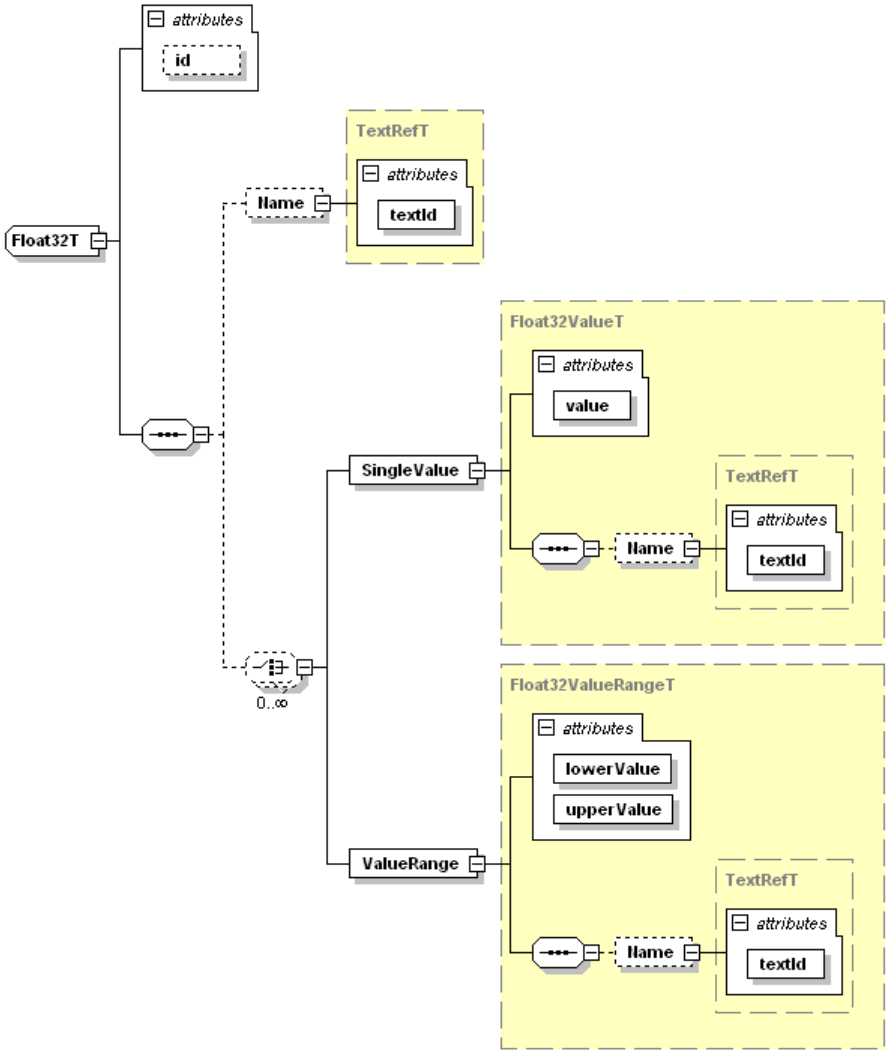


Figure 40 – Float32T

Example:

```
<Datatype xsi:type="Float32T">
        <SingleValue xsi:type="Float32ValueT" value="0.0">
                <Name textId="TI_Zero"/>
        </SingleValue>
        <ValueRange xsi:type="Float32ValueRangeT" lowerValue="1.0" upperValue="1000.0">
                <Name textId="TI_Valid"/>
        </SingleValue>
</Datatype>
```

Lexical representation:

Conforms to the representation of “float” in XML Schema, see
http://www.w3.org/TR/xmlschema-2/#float

Regular expression pattern: “[+-]?\d+(\.\d+)?([eE][+-]?\d+)?|-?INF|NaN”

8.2.6 StringT

A StringT is representing an ordered sequence of symbols (characters) with a variable or fixed length of octets (maximum of 232 octets) coded in US-ASCII (7 bit) or UTF-8. UTF-8 uses one

octet for all ASCII characters and up to 4 octets for other characters. 0x00 is not permitted as a character. Table 11 shows the definition.

Table 11 – StringT

Data type name	Encoding	Standards	Length
StringT	US-ASCII	ANSI INCITS 4-1986 (R2007), corresponding to ISO/IEC 646:1991 (national variant US)	Any length of character string with a maximum of 232 octets. The length shall be defined within a device's IODD via the attribute 'fixedLength'.
	UTF-8	*The Unicode Standard*	

An instance of StringT can be shorter than defined by the IODD attribute 'fixedLength'. 0x00 shall be used for the padding of unused octets. Character strings are transmitted in their actual length in case of singular access (Figure 41). Optimization for transmission is possible by omitting the padding information if the IODD attribute 'fixedLengthRestriction' is not set. The receiver can deduce the original length from the length of the Service PDU.

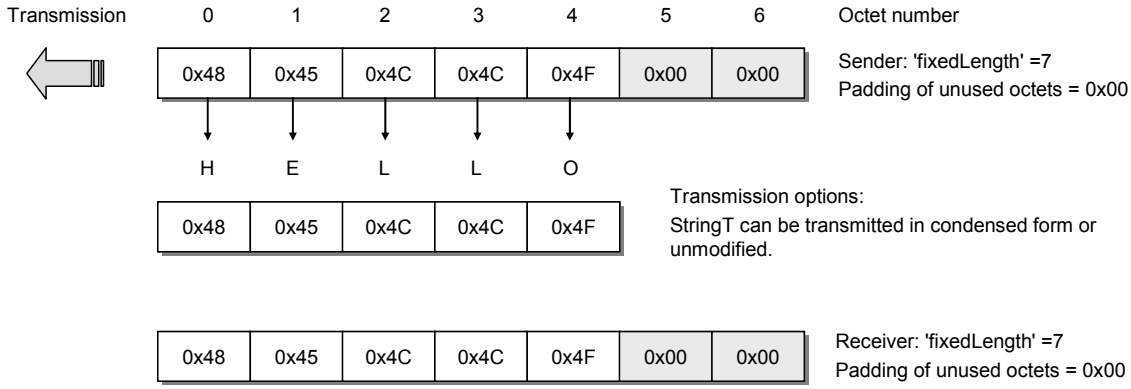


Figure 41 – Singular access of StringT

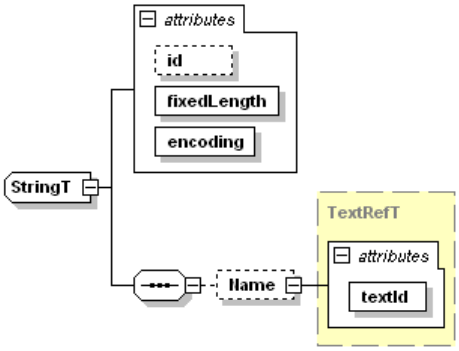


Figure 42 – StringT

The attribute ‘fixedLength’ specifies the length of the string in bytes.

For the attribute ‘encoding’, the values “UTF-8” and “US-ASCII” are allowed. Note that US-ASCII consists of 7-bit characters only.

Example:
<Datatype xsi:type="StringT" fixedLength="64" encoding="UTF-8"/>

Lexical representation:

Conforms to the representation of “string” in XML Schema, see http://www.w3.org/TR/xmlschema-2/#string

Regular expression pattern: “.*” (No restriction, just the string.)

Special characters shall be coded according to the XML syntax. See REC-xml-20081126, chapter 2.4 Character Data and Markup.

& → &

’ → '

> → >

< → <

“ → "

LF →
 LF = linefeed

Only the linefeed is allowed for formatting the text.

8.2.7 OctetStringT

An OctetStringT is representing an ordered sequence of octets with a fixed length (maximum of 232 octets). Table 12 shows the definition and Figure 43 a coding example for a fixed length of 7.

Table 12 – OctetStringT

Data type name	Value range	Standards	Length
OctetStringT	0x00 … 0xFF per octet	-	Fixed length with a maximum of 232 octets
NOTE The length is defined within a device's IODD via the attribute 'fixedLength'.			

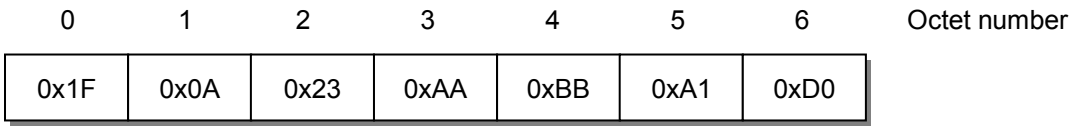


Figure 43 – Coding example of OctetStringT

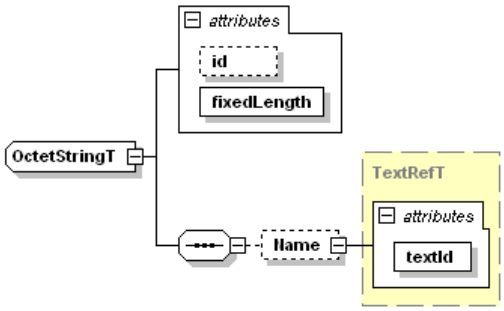


Figure 44 – OctetStringT

The attribute ‘fixedLength’ specifies the length of the octet string in bytes.

Example:

```
<Datatype xsi:type="OctetStringT" fixedLength="10"/>
```

Lexical representation:

Regular expression pattern: “(0x[0-9a-fA-F][0-9a-fA-F],)*0x[0-9a-fA-F][0-9a-fA-F]”

8.2.8 TimeT

A TimeT corresponds to the data type NetworkTime in ISO/IEC 61158-6-10:2007 (see “Encoding of a Network Time value”). It is based on RFC 1305 and composed of two unsigned values that express the network time related to a particular date. Its semantic has changed from RFC 1305 in ISO/IEC 61158-6-10:2007 according to Figure 45. Table 13 shows the definition and Table 14 the coding of TimeT.

The first element is an unsigned integer value that provides the network time in seconds since 1900-01-01 0.00,00(UTC) or since 2036-02-07 6.28,16(UTC) for time values less than 0x9DFF4400, which represents the 1984-01-01 0:00,00(UTC). The second element is an unsigned integer value that provides the fractional portion of seconds in $1/2^{32}$ s. Rollovers after 136 years are not automatically detectable and are to be maintained by the application.

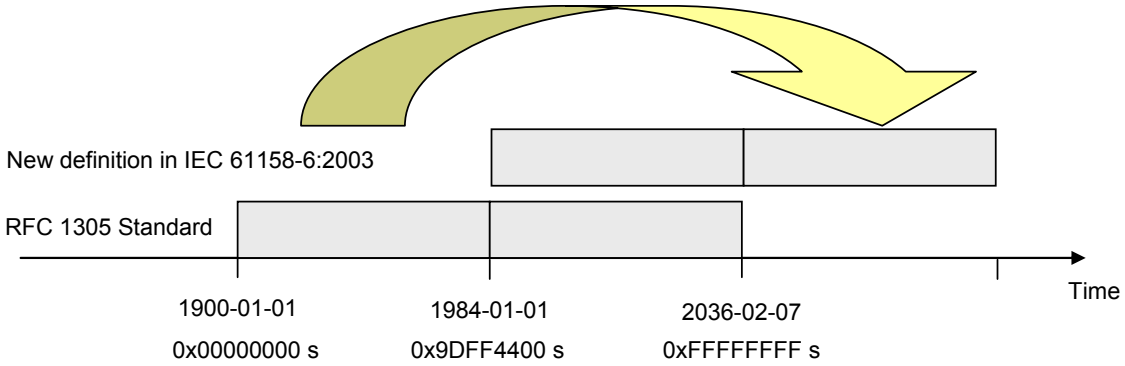


Figure 45 – New definition of NetworkTime in ISO/IEC 61158-6-10:2007

Table 13 – TimeT

Data type name	Value range	Resolution	Length
TimeT	Octet 1 to 4 (see Table 14): $0 \le i \le (2^{32}-1)$	s (Seconds)	8 Octets (32 bit unsigned integer) + 32 bit unsigned integer)
	Octet 5 to 8 (see Table 14): $0 \le i \le (2^{32}-1)$	$(1/2^{32})$ s	
NOTE 32 bit unsigned integer are normal computer science data types			

Table 14 – Coding of TimeT

Bit	7	6	5	4	3	2	1	0	Definitions
Octet 1	2^{31}	2^{30}	2^{29}	2^{28}	2^{27}	2^{26}	2^{25}	2^{24}	Seconds since 1900-01-01 0.00,00 or since 2036-02-07 6.28,16 when time value less than 0x9DFF4400.00000000
Octet 2	2^{23}	2^{22}	2^{21}	2^{20}	2^{19}	2^{18}	2^{17}	2^{16}	
Octet 3	2^{15}	2^{14}	2^{13}	2^{12}	2^{11}	2^{10}	2^{9}	2^{8}	
Octet 4	2^{7}	2^{6}	2^{5}	2^{4}	2^{3}	2^{2}	2^{1}	2^{0}	
Octet 5	2^{31}	2^{30}	2^{29}	2^{28}	2^{27}	2^{26}	2^{25}	2^{24}	Fractional part of seconds. One unit is $1/(2^{32})$ s
Octet 6	2^{23}	2^{22}	2^{21}	2^{20}	2^{19}	2^{18}	2^{17}	2^{16}	
Octet 7	2^{15}	2^{14}	2^{13}	2^{12}	2^{11}	2^{10}	2^{9}	2^{8}	
Octet 8	2^{7}	2^{6}	2^{5}	2^{4}	2^{3}	2^{2}	2^{1}	2^{0}	
	MSB							LSB	MSB = Most significant bit LSB = Least significant bit

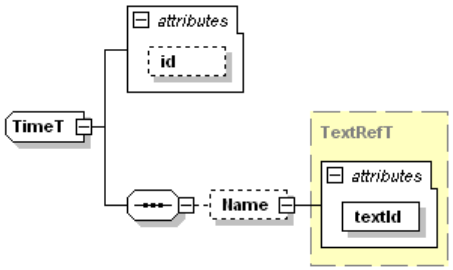


Figure 46 – TimeT

Example:

```
<Datatype xsi:type="TimeT"/>
```

Lexical representation:

Follows the representation of “dateTime” in XML Schema, see http://www.w3.org/TR/xmlschema-2/#dateTime, but is stricter:

Regular expression pattern: “\d{4}\-\d{2}\-\d{2}(T\d{2}:\d{2}:\d{2}(\.\d{1,3})?)?”

(yyyy-mm-dd[Thh:mm:ss[.fff]] where fff = fraction of a second, up to millisecond)

8.2.9 TimeSpanT

A TimeSpanT corresponds to the data type NetworkTimeDifference in ISO/IEC 61158-6-10:2007 (see “Encoding of a Network Time Difference value”). It is composed of an integer value that provides the network time difference in seconds and of an unsigned integer value that provides the fractional portion of seconds in $1/2^{32}$ seconds. Table 15 shows the definition and Table 16 the coding of TimeSpanT.

Table 15 – TimeSpanT

Data type name	Value range	Resolution	Length
TimeSpanT	Octet 1 to 4 (see Table 16): $-2^{31} \le i \le (2^{31}-1)$	s (Seconds)	8 octets (32 bit integer + 32 bit unsigned integer)
	Octet 5 to 8 (see Table 16): $0 \le i \le (2^{32}-1)$	$(1/2^{32})$ s	
NOTE 32 bit integer and 32 bit unsigned integer are normal computer science data types			

Table 16 – Coding of TimeSpanT

Bit	7	6	5	4	3	2	1	0	Definitions
Octet 1	2^{31}	2^{30}	2^{29}	2^{28}	2^{27}	2^{26}	2^{25}	2^{24}	Seconds (s) as 32 bit integer
Octet 2	2^{23}	2^{22}	2^{21}	2^{20}	2^{19}	2^{18}	2^{17}	2^{16}	
Octet 3	2^{15}	2^{14}	2^{13}	2^{12}	2^{11}	2^{10}	2^{9}	2^{8}	
Octet 4	2^{7}	2^{6}	2^{5}	2^{4}	2^{3}	2^{2}	2^{1}	2^{0}	
Octet 5	2^{31}	2^{30}	2^{29}	2^{28}	2^{27}	2^{26}	2^{25}	2^{24}	Fractional part of seconds as 32 bit unsigned integer. One unit is $1/(2^{32})$ s.
Octet 6	2^{23}	2^{22}	2^{21}	2^{20}	2^{19}	2^{18}	2^{17}	2^{16}	
Octet 7	2^{15}	2^{14}	2^{13}	2^{12}	2^{11}	2^{10}	2^{9}	2^{8}	
Octet 8	2^{7}	2^{6}	2^{5}	2^{4}	2^{3}	2^{2}	2^{1}	2^{0}	
	MSB							LSB	MSB = Most significant bit LSB = Least significant bit

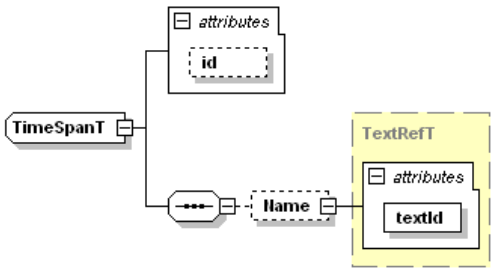


Figure 47 – TimeSpanT

Example:

<Datatype xsi:type="TimeSpanT"/>

Lexical representation:

Follows the representation of “duration” in XML Schema, see http://www.w3.org/TR/xmlschema-2/#duration, but is much stricter:

Regular expression pattern: “[+-]?PT\d+(\.\d{1,3})?S”

8.3 Complex data types

8.3.1 General

Complex data types are combinations of simple data types. Complex data types consist of several simple data types in a packed manner within a sequence of octets. Unused bit space shall be padded with "0".

The coding of simple data types within complex data types shall be the same as for singular use specified in chapter 8.2, except for:

BooleanT

The coding of BooleanT is only 1 bit wide. A value of 0 indicates false and a value of 1 indicates true. There is no padding to a byte.

UIntegerT and IntegerT

The coding of UIntegerT and IntegerT is as wide as indicated by the attribute bitLength. There is no padding to 1 / 2 / 4 / 8 byte.

8.3.2 Arrays

An array addressed by an Index, is a data structure with data items of the same data type. The individual data items are addressable by the Subindex. Subindex = 0 addresses the whole array within the Index space. The structuring rules for arrays are:

1. The Subindex data items are packed in a row without gaps describing an octet sequence
2. The highest Subindex data item n starts right-aligned within the octet sequence
3. UIntegerT and IntegerT with a length of ≥ 58 bit and < 64 bit are not permitted

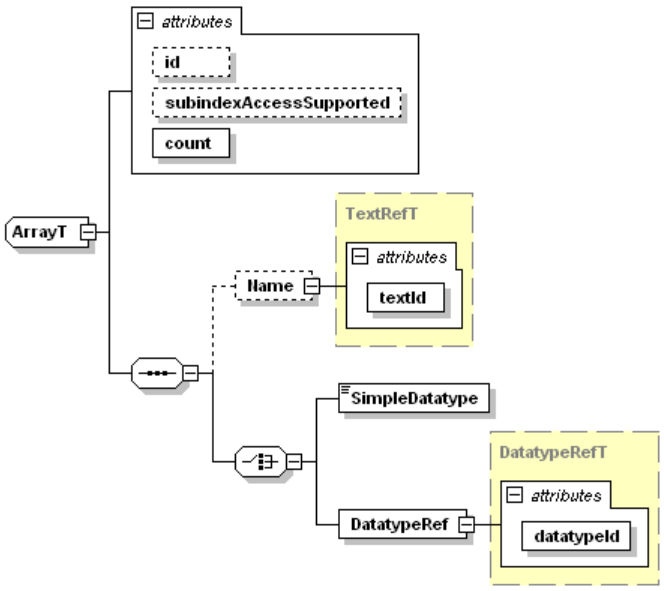


Figure 48 – ArrayT

The SimpleDatatype element allows any of the types derived from SimpleDatatypeT. Instead of defining the simple data type inside the array definition, it is also possible to reference the definition of a simple data type from the DatatypeCollection with DatatypeRef/@datatypeId.

The attribute ‘count’ specifies the fixed number of data items in the array.

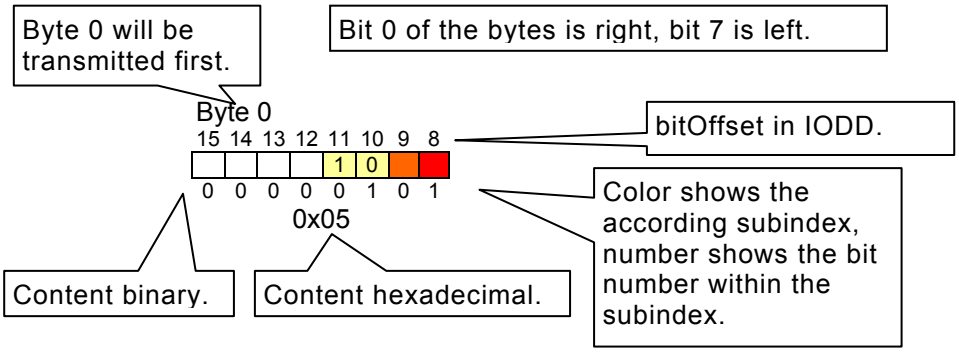
If the attribute ‘subindexAccessSupported’ is present and set to false, individual data items of the array cannot be accessed via their subindex. It is only possible to access the complete array via subindex 0.

Lexical representation:

There is no lexical representation for a value of type ArrayT.

Examples

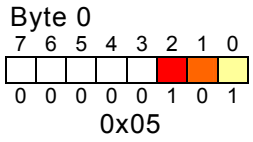
Notation:



Bit array

```
<Datatype xsi:type="ArrayT" count="3">
        <SimpleDatatype xsi:type="BooleanT"/>
</Datatype>
```

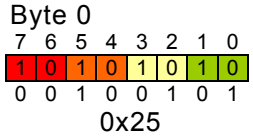
Subindex	Value
1	true
2	false
3	true



Integer array

```
<Datatype xsi:type="ArrayT" count="4">
        <SimpleDatatype xsi:type="IntegerT" bitLength="2"/>
</Datatype>
```

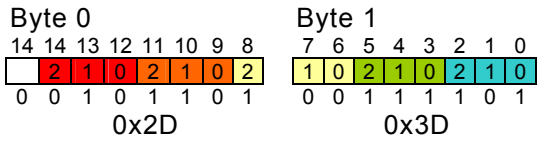
Subindex	Value
1	0
2	-2
3	1
4	1



Integer array

```
<Datatype xsi:type="ArrayT" count="5">
        <SimpleDatatype xsi:type="IntegerT" bitLength="3"/>
</Datatype>
```

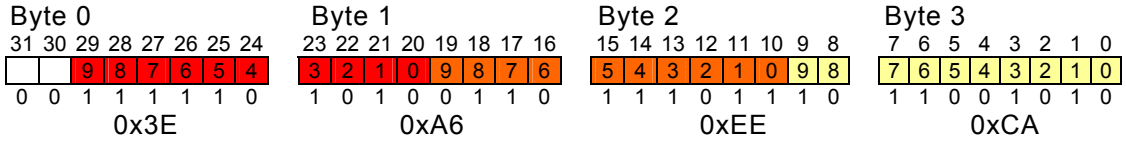
Subindex	Value
1	2
2	6
3	4
4	7
5	5



Integer array

```
<Datatype xsi:type="ArrayT" count="3">
        <SimpleDatatype xsi:type="IntegerT" bitLength="10"/>
</Datatype>
```

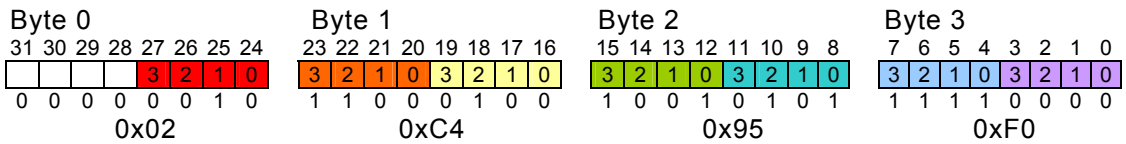
Subindex	Value
1	0x3EA
2	0x1BB
3	0x2CA



Integer array

```
<Datatype xsi:type="ArrayT" count="7">
        <SimpleDatatype xsi:type="IntegerT" bitLength="4"/>
</Datatype>
```

Subindex	Value
1	2
2	12
3	4
4	9
5	5
6	15
7	0



8.3.3 Records

A record addressed by an Index, is a data structure with data items of different data types. The Subindex allows addressing individual data items within the record on certain bit positions to be defined via the IODD of the particular device. The structuring rules for records are:

1. The Subindices within the IODD shall be listed in ascending order from 1 to n describing an octet sequence. Gaps within the list of Subindices are allowed.

2. Bit offsets shall always be indicated within this octet sequence (may show no strict order in the IODD).
3. The bit offset starts with the last octet within the sequence; this octet starts with offset 0 for the least significant bit and offset 7 for the most significant bit.
4. The following data types shall always be aligned on octet boundaries: Float32T, StringT, OctetStringT, TimeT, and TimeSpanT.
5. UIntegerT and IntegerT with a length of ≥ 58 bit shall always be aligned on one side of an octet boundary.
6. It is highly recommended for UIntegerT and IntegerT with a length of ≥ 8 bit to align always on one side of an octet boundary.
7. It is highly recommended for UIntegerT and IntegerT with a length of < 8 bit not to cross octet boundaries.

If recommendations 6 and 7 are not followed, it will be painful for the PLC programmer to extract the desired element(s) from the record. The PLC programmer will usually circumvent this by accessing individual elements via their subindex, but this does not work if

- the record must be accessed completely because consistency is needed
- the Subindex access is not supported (attribute subindexAccessSupported is false)
- the record is used with ProcessData

It is recommended that the Subindices occur in increasing order within the octet sequence. If Subindices are placed in previously unused areas of the octet sequence, one might deviate from this recommendation. If compatible extensions are foreseen, it is better to reserve enough Subindices for the unused areas.

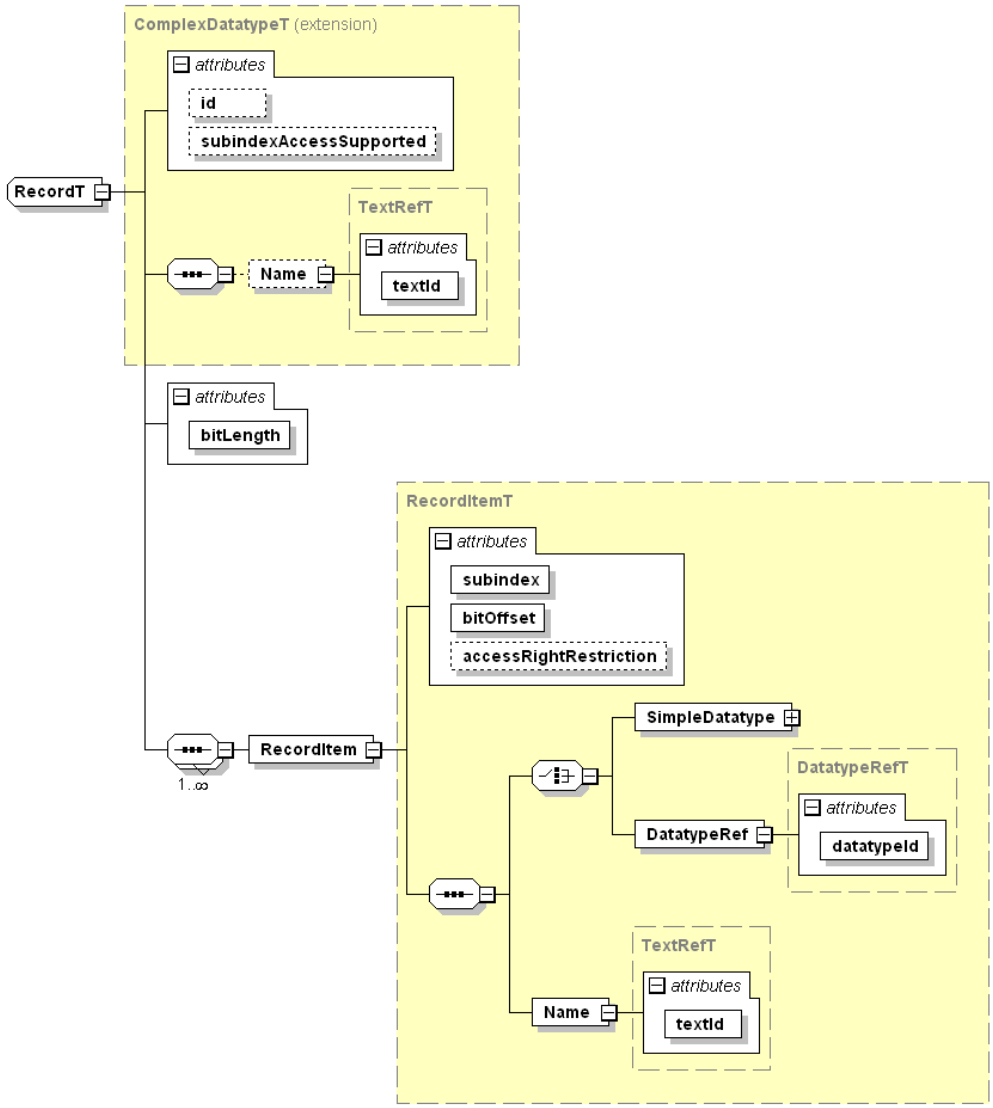


Figure 49 – RecordT

If the attribute ‘subindexAccessSupported’ is present and set to false, individual record items cannot be accessed via their Subindex. It is only possible to access the complete record via Subindex 0.

Within the record item, the SimpleDatatype element allows any of the types derived from SimpleDatatypeT. Instead of defining simple data types inside the record definition, it is also possible to reference the definition of simple data types from the DatatypeCollection with DatatypeRef/@datatypeId.

The attribute ‘subindex’ specifies the Subindex assigned to this record item. The record items shall be ordered by Subindex within the record.

The ‘bitOffset’ specifies the bit position of the record item within the octet sequence.

Individual record items may have less access rights than the record in general. This is indicated by the attribute ‘accessRightRestriction’. For the access to the complete record, this means:

- If the Record is ‘rw’ and the record item is restricted to ‘ro’ the device must tolerate (ignore) the data written to this Subindex.
- If the Record is ‘rw’ and the record item is restricted to ‘wo’, the engineering system must ignore data read from this Subindex.

The attribute ‘accessRightRestriction’ is only applicable for service parameter, not for process data.

The element ‘Name’ assigns a human readable name to the record item. This name shall be displayed additionally by the engineering system.

Lexical representation:

There is no lexical representation for a value of type RecordT.

Examples:

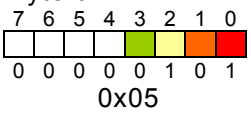
Regarding the notation see 8.3.2.

Several Booleans in a Byte

```
<Datatype xsi:type="RecordT" bitLength="4">
        <Name textId="TI_Switches"/>
        <RecordItem subindex="1" bitOffset="0">
                <SimpleDatatype xsi:type="BooleanT"/>
                <Name textId="TI_Switch1"/>
        </RecordItem>
        <RecordItem subindex="2" bitOffset="1">
                <SimpleDatatype xsi:type="BooleanT"/>
                <Name textId="TI_Switch2"/>
        </RecordItem>
        <RecordItem subindex="3" bitOffset="2">
                <SimpleDatatype xsi:type="BooleanT"/>
                <Name textId="TI_Switch3"/>
        </RecordItem>
        <RecordItem subindex="4" bitOffset="3">
                <SimpleDatatype xsi:type="BooleanT"/>
                <Name textId="TI_Switch4"/>
        </RecordItem>
</Datatype>
```

RecordItem	Subindex	Datentyp	bitLength	bitOffset	Value
1	1	BooleanT	–	0	true
2	2	BooleanT	–	1	false
3	3	BooleanT	–	2	true
4	4	BooleanT	–	3	false

Byte 0



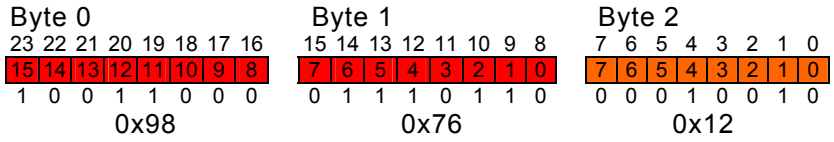
0x05

A word and a byte

```
<Datatype xsi:type="RecordT" bitLength="24">
        <Name textId="TI_Values"/>
        <RecordItem subindex="1" bitOffset="8">
                <SimpleDatatype xsi:type="UIntegerT" bitLength="16"/>
                <Name textId="TI_Value1"/>
        </RecordItem>
        <RecordItem subindex="2" bitOffset="0">
                <SimpleDatatype xsi:type="UIntegerT" bitLength="8"/>
                <Name textId="TI_Value2"/>
        </RecordItem>
```

```
</Datatype>
```

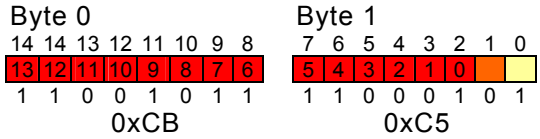
RecordItem	Subindex	Datentyp	bitLength	bitOffset	Value
1	1	UIntegerT	16	8	0x9876
2	2	UIntegerT	8	0	0x12



Analog value and two signal bits

```
<Datatype xsi:type="RecordT" bitLength="16">
        <Name textId="TI_ProcessData"/>
        <RecordItem subindex="1" bitOffset="2">
                <SimpleDatatype xsi:type="UIntegerT" bitLength="14"/>
                <Name textId="TI_AnalogValue"/>
        </RecordItem>
        <RecordItem subindex="2" bitOffset="1">
                <SimpleDatatype xsi:type="BooleanT"/>
                <Name textId="TI_Signal2"/>
        </RecordItem>
        <RecordItem subindex="3" bitOffset="0">
                <SimpleDatatype xsi:type="BooleanT"/>
                <Name textId="TI_Signal1"/>
        </RecordItem>
</Datatype>
```

RecordItem	Subindex	Datentyp	bitLength	bitOffset	Value
1	1	UIntegerT	14	2	0x32F1
2	2	BooleanT	–	1	false
3	3	BooleanT	–	0	true



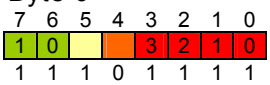
Boolean and enumerations in a byte

```
<Datatype xsi:type="RecordT" bitLength="8">
        <Name textId="TI_ComplexSettings"/>
        <RecordItem subindex="1" bitOffset="0">
                <SimpleDatatype xsi:type="UIntegerT" bitLength="4"/>
                <Name textId="TI_Enum1"/>
        </RecordItem>
        <RecordItem subindex="2" bitOffset="4">
                <SimpleDatatype xsi:type="BooleanT"/>
                <Name textId="TI_Switch1"/>
        </RecordItem>
        <RecordItem subindex="3" bitOffset="5">
                <SimpleDatatype xsi:type="BooleanT"/>
                <Name textId="TI_Switch2"/>
        </RecordItem>
```

```
        <RecordItem subindex="4" bitOffset="6">
                <SimpleDatatype xsi:type="UIntegerT" bitLength="2"/>
                <Name textId="TI_Enum2"/>
        </RecordItem>
</Datatype>
```

RecordItem	Subindex	Datentyp	bitLength	bitOffset	Value
1	1	UIntegerT	4	0	0xF
2	2	BooleanT	–	4	false
3	3	BooleanT	–	5	true
4	4	UIntegerT	2	6	0x3

Byte 0



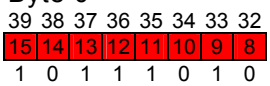
0xEF

With a gap (reserved area for future extension)

```
<Datatype xsi:type="RecordT" bitLength="40">
        <Name textId="TI_Gap"/>
        <RecordItem subindex="1" bitOffset="24">
                <SimpleDatatype xsi:type="UIntegerT" bitLength="16"/>
                <Name textId="TI_Value1"/>
        </RecordItem>
        <RecordItem subindex="3" bitOffset="0">
                <SimpleDatatype xsi:type="UIntegerT" bitLength="16"/>
                <Name textId="TI_Value2"/>
        </RecordItem>
</Datatype>
```

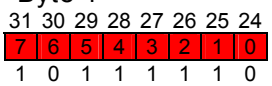
RecordItem	Subindex	Datentyp	bitLength	bitOffset	Value
1	1	UIntegerT	16	24	0xBABE
2	3	UIntegerT	16	0	0xCAFE

Byte 0



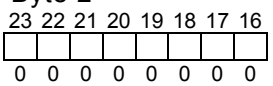
0xBA

Byte 1



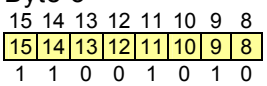
0xBE

Byte 2



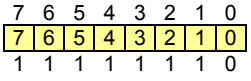
0x00

Byte 3



0xCA

Byte 4



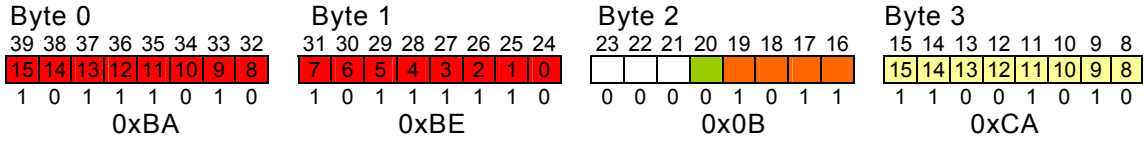
0xFE

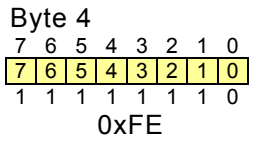
Previous example, extended with two record items

```
<Datatype xsi:type="RecordT" bitLength="40">
        <Name textId="TI_GapFilled"/>
        <RecordItem subindex="1" bitOffset="24">
                <SimpleDatatype xsi:type="UIntegerT" bitLength="16"/>
                <Name textId="TI_Value1"/>
        </RecordItem>
        <RecordItem subindex="2" bitOffset="16">
                <SimpleDatatype xsi:type="UIntegerT" bitLength="4"/>
```

```
            <Name textId="TI_Enum"/>
        </RecordItem>
        <RecordItem subindex="3" bitOffset="0">
            <SimpleDatatype xsi:type="UIntegerT" bitLength="16"/>
            <Name textId="TI_Value2"/>
        </RecordItem>
        <RecordItem subindex="4" bitOffset="20">
            <SimpleDatatype xsi:type="BooleanT"/>
            <Name textId="TI_Switch"/>
        </RecordItem>
</Datatype>
```

RecordItem	Subindex	Datentyp	bitLength	bitOffset	Value
1	1	UIntegerT	16	24	0xBABE
2	2	UIntegerT	4	16	0xB
3	3	UIntegerT	16	0	0xCAFE
4	4	BooleanT	–	20	false

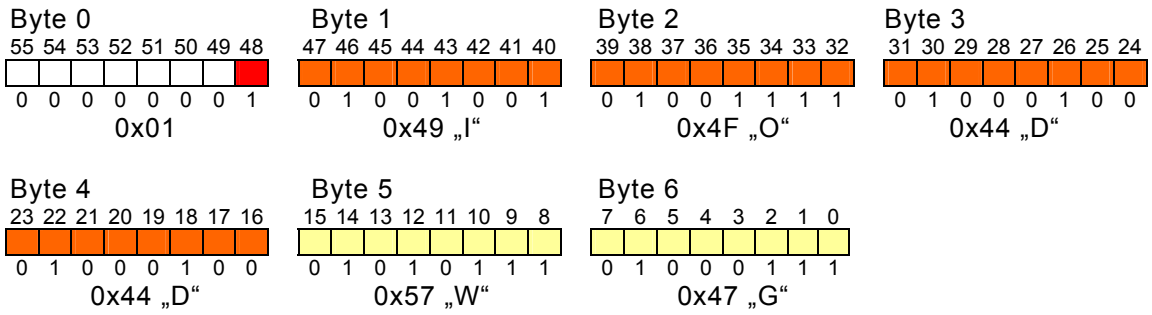




Strings

```
<Datatype xsi:type="RecordT" bitLength="49">
        <Name textId="TI_String"/>
        <RecordItem subindex="1" bitOffset="48">
            <SimpleDatatype xsi:type="BooleanT"/>
            <Name textId="TI_Valid"/>
        </RecordItem>
        <RecordItem subindex="2" bitOffset="16">
            <SimpleDatatype xsi:type="StringT" fixedLength="4"/>
            <Name textId="TI_Text1"/>
        </RecordItem>
        <RecordItem subindex="3" bitOffset="0">
            <SimpleDatatype xsi:type="StringT" fixedLength="2"/>
            <Name textId="TI_Text2"/>
        </RecordItem>
</Datatype>
```

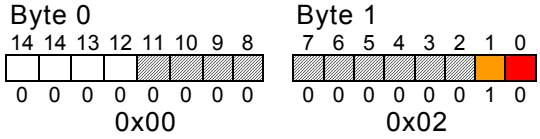
RecordItem	Subindex	Datentyp	fixedLength	bitLength	bitOffset	Value
1	1	BooleanT	–	–	48	true
2	2	StringT	4	–	16	„IODD“
3	3	StringT	2	–	0	„WG“



Two signal bits with reserved space

```
<Datatype xsi:type="RecordT" bitLength="12">
        <Name textId="TI_ProcessData"/>
        <RecordItem subindex="1" bitOffset="0">
                <SimpleDatatype xsi:type="BooleanT"/>
                <Name textId="TI_Signal2"/>
        </RecordItem>
        <RecordItem subindex="2" bitOffset="1">
                <SimpleDatatype xsi:type="BooleanT"/>
                <Name textId="TI_Signal1"/>
        </RecordItem>
</Datatype>
```

RecordItem	Subindex	Datentyp	bitLength	bitOffset	Value
1	1	BooleanT	–	0	false
2	2	BooleanT	–	1	true



Annex A IODD schemas

The following schemas and standard definition files are part of this specification:

Schema files

- IODD1.0.1.xsd main IODD schema
- IODD-Primitives1.0.1.xsd basic definitions
- IODD-Datatypes1.0.1.xsd data types
- IODD-Events1.0.1.xsd events
- IODD-Variables1.0.1.xsd variables
- IODD-UserInterface1.0.1.xsd user interface
- IODD-Communication1.0.1.xsd communication network profile
- IODD-StandardDefinitions1.0.1.xsd main schema for the standard definition files

Standard definition files

- IODD-StandardDefinitions1.0.1.xml list of standard variables + english texts
- IODD-StandardDefinitions1.0.1-de.xml german texts
- IODD-StandardUnitDefinitions1.0.1.xml list of available unit codes + english texts
- IODD-StandardUnitDefinitions1.0.1-de.xml german texts

IO-Link Consortium
Haid-und-Neu-Str. 7
76131 Karlsruhe
Germany
Phone: +49 (0) 721 / 96 58 590
Fax: +49 (0) 721 / 96 58 589
e-mail: info@io-link.com
http://www.io-link.com/