INTERNATIONAL STANDARD

ISO 21219-3

First edition
2019-07

Intelligent transport systems — Traffic and travel information (TTI) via transport protocol experts group, generation 2 (TPEG2) —

Part 3: **UML to binary conversion rules (TPEG2-UBCR)**

Systèmes intelligents de transport — Informations sur le trafic et le tourisme via le groupe expert du protocole de transport, génération 2 (TPEG2) —

Partie 3: Règles de conversion d'UML à système binaire

Reference number
ISO 21219-3:2019(E)

Contents

Page

Foreword

ISO (the International Organization for Standardization) is a worldwide federation of national standards bodies (ISO member bodies). The work of preparing International Standards is normally carried out through ISO technical committees. Each member body interested in a subject for which a technical committee has been established has the right to be represented on that committee. International organizations, governmental and non-governmental, in liaison with ISO, also take part in the work. ISO collaborates closely with the International Electrotechnical Commission (IEC) on all matters of electrotechnical standardization.

The procedures used to develop this document and those intended for its further maintenance are described in the ISO/IEC Directives, Part 1. In particular, the different approval criteria needed for the different types of ISO documents should be noted. This document was drafted in accordance with the editorial rules of the ISO/IEC Directives, Part 2 (see www.iso.org/directives).

Attention is drawn to the possibility that some of the elements of this document may be the subject of patent rights. ISO shall not be held responsible for identifying any or all such patent rights. Details of any patent rights identified during the development of the document will be in the Introduction and/or on the ISO list of patent declarations received (see www.iso.org/patents).

Any trade name used in this document is information given for the convenience of users and does not constitute an endorsement.

For an explanation of the voluntary nature of standards, the meaning of ISO specific terms and expressions related to conformity assessment, as well as information about ISO's adherence to the World Trade Organization (WTO) principles in the Technical Barriers to Trade (TBT) see www.iso.org/iso/foreword.html.

This document was prepared by Technical Committee ISO/TC 204, *Intelligent transport systems*.

This first edition cancels and replaces ISO/TS 21219-3:2015 which has been technically revised. The main changes compared to the previous edition are as follows:

A list of all parts in the ISO 21219 series can be found on the ISO website.

Any feedback or questions on this document should be directed to the user's national standards body. A complete listing of these bodies can be found at www.iso.org/members.html.

Introduction

History

TPEG technology was originally proposed by the European Broadcasting Union (EBU) Broadcast Management Committee, who established the B/TPEG project group in the autumn of 1997 with a brief to develop, as soon as possible, a new protocol for broadcasting traffic and travel-related information in the multimedia environment. TPEG technology, its applications and service features were designed to enable travel-related messages to be coded, decoded, filtered and understood by humans (visually and/or audibly in the user's language) and by agent systems. Originally, a byte-oriented data stream format, which may be carried on almost any digital bearer with an appropriate adaptation layer, was developed. Hierarchically structured TPEG messages from service providers to end-users were designed to transfer information from the service provider database to an end-user's equipment.

One year later, in December 1998, the B/TPEG group produced its first EBU specifications. Two documents were released. Part 2 (TPEG-SSF, which became ISO/TS 18234-2) described the syntax, semantics and framing structure, which was used for all TPEG applications. Meanwhile, Part 4 (TPEG-RTM, which became ISO/TS 18234-4) described the first application for road traffic messages.

Subsequently, in March 1999, CEN/TC 278, in conjunction with ISO/TC 204, established a group comprising members of the former EBU B/TPEG and this working group continued development work. Further parts were developed to make the initial set of four parts, enabling the implementation of a consistent service. Part 3 (TPEG-SNI, ISO/TS 18234-3) described the service and network information application used by all service implementations to ensure appropriate referencing from one service source to another.

Part 1 (TPEG-INV, ISO/TS 18234-1) completed the series by describing the other parts and their relationship; it also contained the application IDs used within the other parts. Additionally, Part 5, the public transport information application (TPEG-PTI, ISO/TS 18234-5), was developed. The so-called TPEG-LOC location referencing method, which enabled both map-based TPEG-decoders and non-map-based ones to deliver either map-based location referencing or human readable text information, was issued as ISO/TS 18234-6 to be used in association with the other applications of parts of the ISO/TS 18234 series to provide location referencing.

The ISO/TS 18234 series has become known as TPEG Generation 1.

TPEG Generation 2

When the Traveller Information Services Association (TISA), derived from former forums, was inaugurated in December 2007, TPEG development was taken over by TISA and continued in the TPEG applications working group.

It was about this time that the (then) new Unified Modelling Language (UML) was seen as having major advantages for the development of new TPEG applications in communities who would not necessarily have binary physical format skills required to extend the original TPEG TS work. It was also realized that the XML format for TPEG described within the ISO/TS 24530 series (now superseded) had a greater significance than previously foreseen, especially in the content-generation segment and that keeping two physical formats in synchronism, in different standards series, would be rather difficult.

As a result, TISA set about the development of a new TPEG structure that would be UML-based. This has subsequently become known as TPEG Generation 2.

TPEG2 is embodied in the ISO/TS 21219 series and it comprises many parts that cover introduction, rules, toolkit and application components. TPEG2 is built around UML modelling and has a core of rules that contain the modelling strategy covered in ISO 21219-2, ISO 21219-3 and ISO 21219-4 and the conversion to two current physical formats: binary and XML; others could be added in the future. TISA uses an automated tool to convert from the agreed UML model XMI file directly into an MS Word document file, to minimize drafting errors, that forms the annex for each physical format.

TPEG2 has a three-container conceptual structure: message management (ISO 21219-6), application (several parts) and location referencing (ISO/TS 21219-7). This structure has flexible capability and can accommodate many differing use cases that have been proposed within the TTI sector and wider for hierarchical message content.

TPEG2 also has many location referencing options as required by the service provider community, any of which may be delivered by vectoring data included in the location referencing container.

The following classification provides a helpful grouping of the different TPEG2 parts according to their intended purpose. Note that the list below may be incomplete, e.g. new TPEG2 parts may be introduced after publication of this document.

— Toolkit parts: TPEG2-INV (ISO/TS 21219-1), TPEG2-UML (ISO 21219-2), TPEG2-UBCR (ISO 21219-3), TPEG2-UXCR (ISO 21219-4), TPEG2-SFW (ISO 21219-5), TPEG2-MMC (ISO 21219-6), TPEG2-LRC (ISO/TS 21219-7).

— Special applications: TPEG2-SNI (ISO/TS 21219-9), TPEG2-CAI (ISO/TS 21219-10), TPEG2-LTE (ISO/TS 21219-24).

— Location referencing: TPEG2-OLR (ISO/TS 21219-22), TPEG2-GLR (ISO/TS 21219-21), TPEG2-TLR (ISO17572-2), TPEG2-DLR (ISO17572-3).

— Applications: TPEG2-PKI (ISO/TS 21219-14), TPEG2-TEC (ISO/TS 21219-15), TPEG2-FPI (ISO/TS 21219-16), TPEG2-TFP (ISO 21219-18), TPEG2-WEA (ISO/TS 21219-19), TPEG2-RMR (ISO/TS 21219-23), TPEG2-EMI (ISO/TS 21219-25), TPEG2-VLI (ISO/TS 21219-26).

TPEG2 has been developed to be broadly (but not totally) backward compatible with TPEG1 to assist in transitions from earlier implementations, while not hindering the TPEG2 innovative approach and being able to support many new features, such as dealing with applications having both long-term, unchanging content and highly dynamic content, such as parking information.

This document is based on the TISA specification technical/editorial version reference: SP10032.

Intelligent transport systems — Traffic and travel information (TTI) via transport protocol experts group, generation 2 (TPEG2) —

Part 3: UML to binary conversion rules (TPEG2-UBCR)

1 Scope

TPEG applications are modelled in UML to provide an application description that is independent of a physical format representation. By separating semantics from application description, applications can easily be developed at a functional level. Different physical format representations can be generated following a well defined set of rules on how to convert UML classes to different physical formats.

This document specifies the rules for converting UML models of TPEG application to the TPEG binary format. It contains the binary format definition of the abstract data types defined in ISO 21219-2. Rules for converting compound data types are also defined.

2 Normative references

The following documents are referred to in the text in such a way that some or all of their content constitutes requirements 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.

ISO 21219-2, *Intelligent transport systems — Traffic and travel information (TTI) via transport protocol experts group, generation 2 (TPEG2) — Part 2: UML modelling rules*

ISO 21219-5, *Intelligent transport systems — Traffic and travel information (TTI) via transport protocol experts group, generation 2 (TPEG2) — Part 5: Service framework*

ISO/IEC/IEEE 60559, *Information technology — Microprocessor Systems — Floating-Point arithmetic*

3 Terms and definitions

No terms and definitions are listed in this document.

ISO and IEC maintain terminological databases for use in standardization at the following addresses:

— ISO Online browsing platform: available at https://www.iso.org/obp

— IEC Electropedia: available at http://www.electropedia.org/

4 Abbreviated terms

The abbreviated terms defined in ISO 21219-2 and the following apply.

LSB	Least Significant Bit
LSByte	Least Significant Byte
MSB	Most Significant Bit
MSByte	Most Significant Byte

5 Rules for UML to binary format description conversion

5.1 Definition of binary format description

The binary format description of TPEG applications is included in application specifications as a normative annex. This annex shall be named according to the following scheme:

[Full application name], TPEG-binary representation

The annex shall have four subclauses: *Introduction*, *Application framing and signalling*, *Application components* and *Application datastructures*. The content of these subclauses is subject to the specifications in this clause.

The introduction shall use a similar formulation as in the following:

> This chapter defines the application framing and the format of the *[Full application name]* message components, datastructures and its attributes for the TPEG binary representation of *[application abbreviation]* as described in *[reference to TPEG framework]*. For further descriptions of these objects see the related clauses *[reference to clauses]* in this specification.

The Application framing and signalling subclause shall have three parts: Application identification, Version number signalling, and Application framing. The Application identification part shall define the Application Identifier (AID) that is used for the application. The Version number signalling shall define the major and minor version number of the application that are signalled within the SNI application. The Application framing part shall state in what kind of service component the application shall be transmitted. TPEG Service Component (SC) types are defined in ISO 21219-5. Currently, the following Service Component types are defined:

— *ServCompFrame* — Standard SC

— *ServCompFrameProtected* — SC with data CRC

— *ServCompFrameCountedProtected* — SC with message count and data CRC

— *ServCompFramePrioritisedProtected* — SC with group priority and data CRC

— *ServCompFramePrioritisedCountedProtected* — SC with group priority, message count and data CRC

The wording shall be similar to the following:

> TPEG binary format messages of the *[Full application name]* type are transmitted in Service Component Frames of the *[Service Component Frame type]* type. Service Component Frames are described in *[reference to TPEG framework]*.

The *Application components* description shall have a first subclause with title *List of generic component IDs*. This clause contains unique component IDs for each application UML class that is not stereotyped as <<DataStructure>>. The component IDs should be ordered in the order of appearance in the model.

The list of generic component IDs subclause is followed by subclauses providing the binary format description of each application UML class that is not stereotyped as <<DataStructure>>. This binary format description shall follow the rules as specified in 5.5. The generic component ID of each component defined in the list of generic component IDs shall be inserted in the binary format description where the rules of 5.5 read 'gcid'.

The *Application datastructures* description shall provide the binary format description of each application UML class that is stereotyped as <<DataStructure>>. This binary format description shall follow the rules as specified in 5.5.

5.2 Abstract data types

This clause presents the binary format definition of the abstract data types that are defined in the TPEG UML modelling rules document ISO 21219-5.

The following general rules were used for defining data types in the column "binary format definition":

A data type is written in upper camel case letters in one single expression. The data type may contain letters (a-z), number (0-9), underscore "_", round brackets "()" and colon ":"; the first must be a letter.

EXAMPLE IntUnLo stands for Integer Unsigned Long

A data type is framed by angle brackets “ < > ” .

The content of a data type is defined by a colon followed by an equal sign “ := ”.

The end of a data type is indicated by a semicolon “ ; ”.

A descriptor written in lower camel case may be added to a data type as one single expression without spaces.

A descriptor is framed by round brackets “ ( ) ”.

The descriptor contains either a value or a name of the associated type.

Data types in a definition list of another one are separated by commas “ , ”. The order of definition is defined as the order of occurrence in a data stream.

Curly brackets (braces) “ { } ” group together a block of data types.

Control statements (“if”, “infinite”, “unordered” or “external”) are noted in lower case letters. A control statement is followed by a block statement or only one data type:

1) **if** defines a condition statement. The block’s (or data type’s) occurrence is conditional to the condition statement being valid. The condition statement is framed with round brackets. This statement applies to any data type.

2) **infinite** defines endless repetition of the block (or data type). This is only used to mark the main TPEG stream as not ending stream of data.

3) **unordered** defines that the following block contains data types which may occur in any order, not only the one used to specify subsequent data types. This statement applies to components only.

4) **ordered** defines that the following block contains data types of which the order of definition is defined as the order of occurrence in a data stream. This statement applies to components only.

5) **external** defines that the content of the data type is being defined external to the scope of given specification. The control statement “external” must be followed by only one data. A reference to the corresponding specification should follow in the comment. All types specified in TYP specification are treated as being in scope of any application.

The expression “ n * ” indicates multiplicity of occurrence of a data type. The lower and upper bound are implicitly from 0 to infinite; other bounds are described in square brackets between two points " .. " and behind the data type descriptor. The " * " stands for no limitation at upper bound.

Any text after a colon “ : ” is regarded as a comment.

For clear graphical presentation, lines in a coding box if they are too long to fit, are broken with a backslash “\” followed by a carriage return. The broken line restarts with an additional backslash.

Data type	Binary format definition
BitArray	**<BitArray>**:= m * **<byte>**[1..*]; : Byte containing bits. MSB signals following bytes Bit set (= 1) signals logical true Bit not set (= 0) or not present signals logical false The bits in a BitArray are encoded in a sequence of bytes, where the first bit of each byte (MSB) is a continuation flag (marked as CF in the table below). If this bit is set (=1) there follows at least one more byte in this BitArray. The last byte always has the continuation flag not set (=0). A BitArray represents a list of Boolean values which is implemented in the same way as for all lists. The first byte holds bits numbered from zero to six in that order. The second byte holds bits numbered seven to 13, again in that order, and so on. The ordering is sequential from first bit (MSB) to last bit (LSB).

Table 1 — Binary format coding of BitArray

Byte 0								Byte 1								...	
	Bit number								Bit number								
CF	0	1	2	3	4	5	6	CF	7	8	9	10	11	12	13		...

Data type	Binary format definition
	If all bits after a certain bit in a BitArray are not set, the remaining bytes containing only unset bits may be removed. The continuation flag of the new last byte is set to false. Decoders shall interpret undefined bits as logical value false. EXAMPLE BitArray =05 hex: Bit 4 and bit 6 are set, the BitArray consists of only one byte (continuation flag not set).
Boolean	The TPEG binary format knows three representations for Booleans. — Mandatory Booleans are stored in the selector of a class — Multiple mandatory Booleans are stored in **<MultipleBooleans>** — Single, optional Booleans are stored in a table of type typ008:OptionalBoolean as defined in ISO 21219-2 The default value of a Boolean is *false*.
DataStructure	**<DataStructure>**:= : Name of data structure **<...>**, : Content of data structure ...;
DateTime	**<DateTime>**:= : Date and time **<IntUnLo>**; : Number of seconds since 1970-01-01T00:00:00 Universal Coordinated Time (UTC) NOTE The formula for date and time calculation is given in ISO/TS 18234-2:2013, Annex D.

Data type	Binary format definition
DaySelector	**<DaySelector>**:= **<BitArray>**(selector), : DaySelector if (bit 0 of selector is set) **<Boolean>**(Saturday), : every Saturday if (bit 1 of selector is set) **<Boolean>**(Friday), : every Friday if (bit 2 of selector is set) **<Boolean>**(Thursday), : every Thursday if (bit 3 of selector is set) **<Boolean>**(Wednesday), : every Wednesday if (bit 4 of selector is set) **<Boolean>**(Tuesday), : every Tuesday if (bit 5 of selector is set) **<Boolean>**(Monday), : every Monday if (bit 6 of selector is set) **<Boolean>**(Sunday); : every Sunday
DistanceMetres	**<DistanceMetres>**:= **<IntUnLoMB>**; : Distance in integer units of metres
DistanceCentiMetres	**<DistanceCentiMetres>**:= **<IntUnLoMB>**; : Distance in integer units of centimetres
Duration	**<Duration>**:= **<IntUnLoMB>**; : Time duration in number of seconds
FixedPercentage	**<FixedPercentage>**:= **<IntUnTi>**; : integer value of percentage
Deprecated: FixedPointNumber	**<FixedPointNumber>**:= **<IntSiLoMB>**(integerPart), : integer part of the number **<IntUnTi>**(decimalPart); : fraction of 2 decimal digits [0…99] The datatype is deprecated and may be removed in one of the future versions of this document. Please do not make use of this datatype anymore.
Float	**<Float>**:= 4 * **<byte>;** : ISO/IEC/IEEE 60559 single precision floating point number NOTE Floating-point numbers are in the form of $s\ (m\ /\ 2^{N-1})\ 2^{e}$, with s the sign, m the mantissa, e the exponent and N the number of bits in the mantissa. For single precision floating point numbers, $N = 24$ the first bit in the mantissa is always one and can therefore be omitted. Sign: bit# 31 (bit #) Exponent: bit# 23-30 Mantissa: bit# 0-22
IntSiTi	**<IntSiTi>**:= **<byte>**; : Two's complement
IntSiLi	**<IntSiLi>**:= **<byte>,** : MSByte, two's complement **<byte>**; : LSByte, two's complement
IntSi24	**<IntSi24>:=** **<byte>,** : MSByte, two's complement **<byte>,** **<byte>;** : LSByte, two's complement
IntSiLo	**<IntSiLo>:=** **<byte>,** : MSByte, two's complement **<byte>,** **<byte>,** **<byte>**; : LSByte, two's complement

Data type	Binary format definition
IntSiLoMB	**<IntSiLoMB>**:= m * **<byte>**[1..5]; : MSB of each byte is continuation flag. Two's complement after elimination of continuation flags The signed multi-byte is defined in the same way as IntUnLoMB except in case of signed value interpretation; the complement on two is used on the 7-bit wide byte series. The count of bytes is then defined by the magnitude of the positive value to be stored in multi-byte. The three reserved bits in most significant byte #5 shall be set to 111 in case of negative numbers with 5-byte length and 000 otherwise, to be upward compatible in case of introduction of a 64-bit integer value in future. Signed values from 0 to -2^{6} are stored in one byte, to -2^{13} in two bytes, to -2^{20} in three bytes, to -2^{27} in four bytes and to -2^{31} in five bytes.
IntUnTi	**<IntUnTi>**:= **<byte>**; : Primitive
IntUnLi	**<IntUnLi>**:= **<byte>**, : MSByte **<byte>**; : LSByte
IntUn24	**<IntUn24>**:= **<byte>**, : MSByte **<byte>**, **<byte>**; : LSByte
IntUnLo	**<IntUnLo>**:= **<byte>**, : MSByte **<byte>**, **<byte>**, **<byte>**; : LSByte
IntUnLoMB	**<IntUnLoMB>**:= m * **<byte>**[1..5]; : MSB of each byte is continuation flag. Three MSBs from the fifth, i.e. most significant, byte are reserved for future use. A multi-byte integer consists of a series of bytes, where the most significant bit is the continuation flag and the remaining seven bits are a scalar value. The continuation flag indicates that a byte is not the end of the multi-byte sequence. A single integer value is encoded into a sequence of N bytes. The first N–1 bytes have the continuation flag set to a value of one (1). The final byte in the series has a continuation flag value of zero (0). This allows to know exactly the end of a series of bytes belonging to one multi-byte, being the one with MSB=0. The bytes are encoded in "big-endian" order i.e. most significant byte first. The maximum number of concatenated bytes is 5, so that the maximum unsigned integer, which can be encoded is $2^{(40-5)}$ –1. However, the actual specification defines the three most significant bits of the fifth byte (i.e. the most significant byte) as "reserved for future use" i.e. to be set to 000. This leads into the maximum number $2^{(32)}$ –1 which is the maximum value of a four-byte unsigned integer.
ShortString	**<ShortString>**:= **<IntUnTi>**(n), : Number of bytes, n n * **<byte>**; : String of characters. The number of characters depends on the chosen character set.

Data type	Binary format definition	
LongString	**<LongString>**:=	
	<IntUnLi>(n),	: Number of bytes, n
	n * **<byte>**;	: String of characters. The number of characters depends on the chosen character set.
LocalisedShortString	**<LocalisedShortString>**:=	
	<typ001:LanguageCode>(languageCode),	: Specifies the language used for this string.
	<ShortString>(string);	: Short string
LocalisedLongString	**<LocalisedLongString>**:=	
	<typ001:LanguageCode>(languageCode),	: Specifies the language used for this string.
	<LongString>(string);	: Long string
Probability	**<Probability>**:=	
	<FixedPercentage>;	: Probabilities mapped to whole percents.
ServiceIdentifier	**<ServiceIdentifier>**:=	
	<IntUnTi>(SID_A),	: Service identification part A
	<IntUnTi>(SID_B),	: Service identification part B
	<IntUnTi>(SID_C);	: Service identification part C
Table	**<Table>**:=	
	<IntUnTi>(entry);	: The corresponding table defines valid entries of a table.
TimeInterval	**<TimeInterval>**:=	
	<BitArray>(selector)	
	if (bit 0 of selector is set)	
	<IntUnTi>(years),	: Number of years
	if (bit 1 of selector is set)	
	<IntUnTi>(months),	: Number of months
	if (bit 2 of selector is set)	
	<IntUnTi>(days),	: Number of days
	if (bit 3 of selector is set)	
	<IntUnTi>(hours),	: Number of hours
	if (bit 4 of selector is set)	
	<IntUnTi>(minutes),	: Number of minutes
	if (bit 5 of selector is set)	
	<IntUnTi>(seconds);	: Number of seconds
TimePoint	**<TimePoint>**:=	
	<BitArray>(selector)	
	if (bit 0 of selector is set)	
	<IntUnTi>(years),	: Number of years
	if (bit 1 of selector is set)	
	<IntUnTi>(months),	: Number of months
	if (bit 2 of selector is set)	
	<IntUnTi>(days),	: Number of days
	if (bit 3 of selector is set)	
	<IntUnTi>(hours),	: Number of hours
	if (bit 4 of selector is set)	
	<IntUnTi>(minutes),	: Number of minutes
	if (bit 5 of selector is set)	
	<IntUnTi>(seconds);	: Number of seconds
	The year value has a range of [0..130] and is calculated by subtracting 1970 from the actual year. The year range [1970..2100] thus maps to values [0..130].	

Data type	Binary format definition
TimeToolkit	**<TimeToolkit>**:= **<BitArray>**(selector) if (bit 0 of selector is set) **<TimePoint>**(startTime), : Starting time if (bit 1 of selector is set) **<TimePoint>**(stopTime), : Stopping time if (bit 2 of selector is set) **<TimeInterval>**(duration), : Time interval if (bit 3 of selector is set) **<typ002:SpecialDay>**(specialDay), : Special day type selection if (bit 4 of selector is set) **<DaySelector>**(daySelector); : Weekday selector
Velocity	**<Velocity>**:= **<IntUnTi>**; : Speed in whole metres per second
Weight	**<Weight>**:= **<IntUnLoMB>**; : Weight in whole kilogrammes
MajorMinorVersion	**<MajorMinorVersion>**:= **<byte>**; : Byte containing bits. The four MSB bits indicate the MajorVersion. The four LSB bits indicate the MinorVersion. MajorMinorVersion encoded as MajorVersion.MinorVersion where the MajorVersion is stored in the four most significant bits of the MajorMinorVersion and the MinorVersion is stored in the four least significant bits of the MajorMinorVersion. This allows for a range of [0..15] for each of the MajorVersion and MinorVersion indicators. The ordering is sequential from first bit (MSB) to last bit (LSB).

Table 2 — Binary format coding of MajorMinorVersion

MajorMinorVersion							
Bit number							
7	6	5	4	3	2	1	0
MajorVersion				MinorVersion			

EXAMPLE:
Majorversion is 3 and MinorVersion is 15.

MajorMinorVersion is then encoded as 0011 1111 binary, which is 63 decimal, and 3F hex.

Data type	Binary format definition
ByteFieldAttribute	**<ByteFieldAttribute>**:= **<IntUnLi>**(n), : Number of bytes, n * **<byte>**; : The above indicated number of bytes.

The character set used in (Localized) short and long strings is signalled in the TPEG SNI application for each TPEG Service. The character set may either be of the ISO 8859 series or the UTF character sets.

<u>Examples for IntSiLoMB</u>

Positive number to be encoded

— in Decimal: 1093567633

— Binary (32bit): 0100 0001001 0111010 0001001 0010001

Multibyte encoded:

Byte [5] (MSByte)	Byte [4]	Byte [3]	Byte[2]	Byte[1] (LSByte)
(1)"000"0100	(1)0001001	(1)0111010	(1)0001001	(0)0010001

Bits in brackets are continuity flags, the ones in quotes are reserved and set to 0.

Multibyte encoded hex: 8489ba8911

Negative number to be encoded

— in Decimal: −1093567633

— Binary (two's complement): 1011 1110110 1000101 1110110 1101111

Multibyte encoded:

Byte [5] (MSByte)	Byte [4]	Byte [3]	Byte[2]	Byte[1] (LSByte)
(1)"111"1011	(1)1110110	(1)1000101	(1)1110110	(0)1101111

Bits in brackets are continuity flags, the ones in quotes are reserved and set to 1.

Multibyte encoded hex: FBF6C5F66F

Example for IntUnLoMB

Positive number to be encoded

— in Decimal: 1093567633

— Binary (32bit): 0100 0001001 0111010 0001001 0010001

Multibyte encoded:

Byte [5] (MSByte)	Byte [4]	Byte [3]	Byte[2]	Byte[1] (LSByte)
(1)"000"0100	(1)0001001	(1)0111010	(1)0001001	(0)0010001

Bits in brackets are continuity flags, the ones in quotes are reserved and set to 0.

Multibyte encoded hex: 8489ba8911

5.3 Binary format specific data types

Data type	**Binary format definition**
MultipleBooleans	**<MultipleBooleans(n)>**:= **<IntUnLoMB>**(n), : Number of Booleans in bitArray. **<BitArray>**(bitArray); : BitArray containing a series of bit switches. Each bit switch represents one Boolean.

5.4 TPEG tables

TPEG tables have up to 256 entries. Each entry is addressed by its code which is defined in the specification of the respective table. As the highest code value is 255, table codes are represented by an IntUnTi in the TPEG binary format.

Data type	**Binary format definition**
TPEG table <<enumeration>>	e.g. **<typ001:LanguageCode>**:= **<IntUnTi>**(code); : Language code as defined in ISO/TS 21219-2

5.5 Compound data types

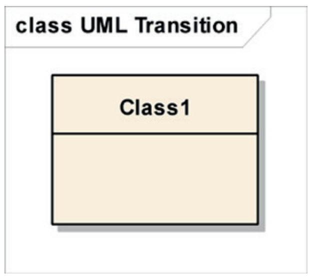
5.5.1 Rule 1: Classes

A class shall be converted into a generic component having a header consisting of:

— generic component ID,

— total length of the component in bytes,

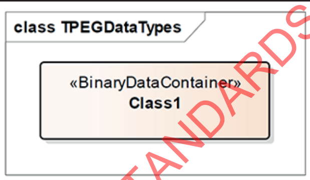
— length of the attributes (not being sub-components) within the component in bytes:

UML	**Binary format definition**
	**<Class1(gcid)>**:= **<IntUnTi>**(gcid), : id of this component **<IntUnLoMB>**(lengthComp), : number of bytes in component (excluding the id and this length indicator) **<IntUnLoMB>**(lengthAttr), : number of bytes in attributes **<…>**; : component data

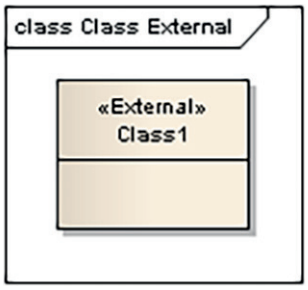
Each binary representation of a TPEG class includes attributes for the related Component Length and Attribute Length (parameters 'lengthComp' and 'lengthAttr' in the binary format definitions above). The Component Length is overall the number of bytes of this component following (not including) the 'lengthComp' parameter. In particular this includes the attribute block with the Attribute Length parameter 'lengthAttr', the mandatory attributes (if present), the selector bitarray (if present), the optional attributes (if present) and the sub-components (if present). The Attribute Length is the overall number of bytes of the attribute Block of this component following (not including) the 'lengthAttr' parameter and not including the sub-components (if present). In particular this includes the mandatory attributes, the selector bitarray (if present) and optional attributes (if present).

See Annex A for usage of the fields lengthComp and LengthAttr to skip unknown content.

Classes stereotyped as <<BinaryDataContainer>> provide the generic component ID and the length indicators, but contain raw binary data that may be specified elsewhere.

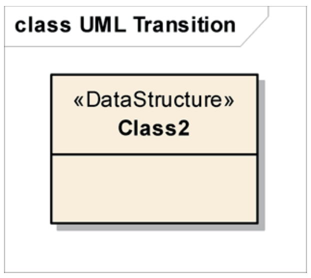
UML	**Binary format definition**
	**<Class1(gcid)>**:= **<IntUnTi>**(gcid), : id of this component **<IntUnLoMB>**(lengthComp), : number of bytes in component, (excluding the id and this length indicator) **<IntUnLoMB>**(lengthAttr), : number of bytes in attributes m * **<IntUnTi>**; : binary component data

Classes stereotyped as <<External>> provide the generic component ID and the length indicators, but inherit the component data from an external specifications class.

UML	Binary format definition
	**<Class1(gcid)>**:= **<IntUnTi>**(gcid), : id of this component **<IntUnLoMB>**(lengthComp), : number of bytes in component, (excluding the id and this length indicator) **<IntUnLoMB>**(lengthAttr), : number of bytes in attributes External **<ImportedClassName(gcid)>**; : external component data

5.5.2 Rule 2: Datastructures

A class with stereotype <<DataStructure>> defines a data record without a component header. A class with stereotype <<DataStructure>> does not have a generic component ID.

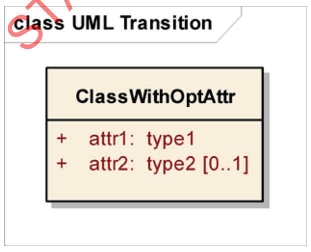
UML	Binary format definition
	**<Class2>**:= **<…>**; : content definition

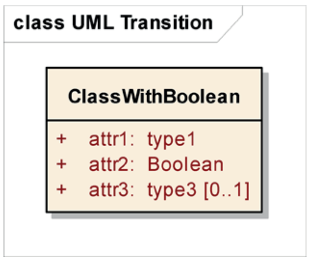
5.5.3 Rule 3: Selector

Presence of optional attributes of a class is signalled in a preceding selector bitarray. The selector bitarray also encodes mandatory Boolean values. The index in the selector corresponds with the order of the optional elements and mandatory Booleans in the model.

The selector bitarray is inserted immediately before the first occurring instance of either an optional attribute or a mandatory Boolean.

Additional rules for selector use and placement for components that are modelled as attributes (using the <<OrderedComponentGroup>> stereotype) apply. These rules are defined in Rule 4d and Rule 4e.

UML	Binary format definition
 class UML Transition ClassWithOptAttr + attr1: type1 + attr2: type2 [0..1]	**<ClassWithOptAttr(gcid)>**:= **<IntUnTi>**(gcid), : id of this component **<IntUnLoMB>**(lengthComp), : number of bytes in component **<IntUnLoMB>**(lengthAttr), : number of bytes in attributes **<type1>**(attr1), : mandatory attribute **<BitArray>**(selector), : selector if (bit 0 of selector is set) **<type2>**(attr2); : optional attribute

UML	Binary format definition
 class UML Transition ClassWithBoolean + attr1: type1 + attr2: Boolean + attr3: type3 [0..1]	**<ClassWithBoolean(gcid)>**:= **<IntUnTi>**(gcid), : id of this component **<IntUnLoMB>**(lengthComp), : number of bytes in component **<IntUnLoMB>**(lengthAttr), : number of bytes in attributes **<type1>**(attr1), : mandatory attribute **<BitArray>**(selector), : selector if (bit 0 of selector is set) **<Boolean>**(attr2), : Boolean attr2 has value *true* if (bit 1 of selector is set) **<type3>**(attr3); : optional attribute

5.5.4 Rule 4: Attributes

Rule 4 consists of five sub-rules, each specifying how to handle class attributes.

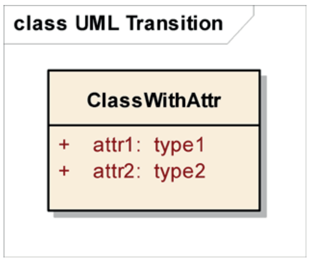
5.5.5 Rule 4a: Datatypes

Attributes of primitive data type shall be converted into binary format as defined in 5.2, 5.3 and 5.4.

Attributes of compound data type shall be converted into binary format according to the rules in this clause (5.5).

5.5.6 Rule 4b: Ordering

The order of the attributes as listed in the UML model shall be preserved.

UML	Binary format definition
 class UML Transition ClassWithAttr + attr1: type1 + attr2: type2	**<ClassWithAttr(gcid)>**:= **<IntUnTi>**(gcid), : id of this component **<IntUnLoMB>**(lengthComp), : number of bytes in component **<IntUnLoMB>**(lengthAttr), : number of bytes in attributes **<type1>**(attr1), : the first attribute **<type2>**(attr2); : the second attribute

5.5.7 Rule 4c: Single multiplicity

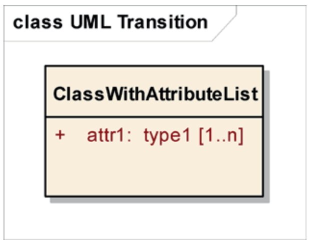
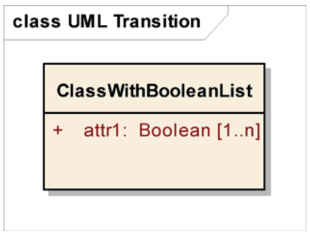
No additional rules.

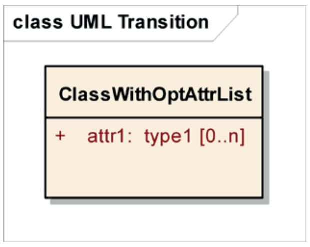
NOTE The explicit notation for single multiplicity ([1]) can be omitted in a model. If not stated otherwise, an attribute has single multiplicity.

5.5.8 Rule 4d: Multiplicity [0..maxOccurs>1], Multiplicity [1..maxOccurs>1] and Multiplicity [minOccurs>1..maxOccurs>1]

Attributes with multiplicity maxOccurs>1 ormaxOccurs>1 not stereotyped as <<OrderedComponentGroup>> or <<UnorderedComponentGroup>> shall indicate the number of occurrences before the list of attributes using a multibyte “n”. The presence of an attribute list with minOccurs =0 is indicated in a selector. In this case, Rule 3 applies.

For Booleans, the multibyte “n” indicates the number of bits that are used. The bits are stored in a multiplebooleans list (see 5.3).

UML	Binary format definition
 class UML Transition ClassWithAttributeList + attr1: type1 [1..n]	**<ClassWithAttributeList(gcid)>**:= **<IntUnTi>**(gcid), : id of this component **<IntUnLoMB>**(lengthComp), : number of bytes in component **<IntUnLoMB>**(lengthAttr), : number of bytes in attributes **<IntUnLoMB>**(n), : {n > 0} n * **<type1>**(attr1); : the attribute list
 class UML Transition ClassWithBooleanList + attr1: Boolean [1..n]	**<ClassWithBooleanList(gcid)>**:= **<IntUnTi>**(gcid), : id of this component **<IntUnLoMB>**(lengthComp), : number of bytes in component **<IntUnLoMB>**(lengthAttr), : number of bytes in attributes **<MultipleBooleans(n)>**; : the Booleans list

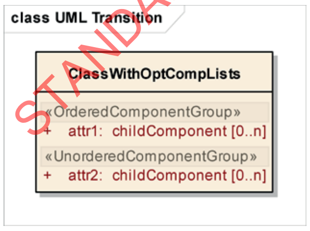
 class UML Transition
ClassWithOptAttrList
+ attr1: type1 [0..n]
	<ClassWithOptAttrList(gcid)>:=
	<IntUnTi>(gcid), : id of this component
	<IntUnLoMB>(lengthComp), : number of bytes in component
	<IntUnLoMB>(lengthAttr), : number of bytes in attributes
	<BitArray>(selector), : selector
	if (bit 0 of selector is set)
	{
	<IntUnLoMB>(n),
	n * **<type1>**(attr1), : optional attribute
	};

NOTE 1 If n equals zero, the multibyte n is still present having value zero, but no attributes are instantiated.

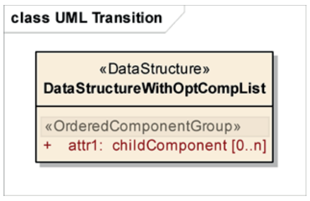
For Components with multiplicity [0..n] or [1..n], included as attribute (stereotyped as <<OrderedComponentGroup>> or <<UnorderedComponentGroup>>), the following applies.

If the Component is included in a class that is not stereotyped as <<DataStructure>> (a Component), the multibyte “n” and the selector shall not be used for indicating the presence and multiplicity of the Component. If the Component is included in a class that is stereotyped as <<DataStructure>>, the Component shall be treated as a regular attribute.

UML	Binary format definition

 class UML Transition
ClassWithOptCompLists
«OrderedComponentGroup»
+ attr1: childComponent [0..n]
«UnorderedComponentGroup»
+ attr2: childComponent [0..n]
	<ClassWithOptCompList(gcid)>:=
	<IntUnTi>(gcid), : id of this component
	<IntUnLoMB>(lengthComp), : number of bytes in component
	<IntUnLoMB>(lengthAttr), : number of bytes in attributes
	ordered {
	n * **<childComponent>**(attr1), : the component list
	}
	unordered {
	n * **<childComponent>**(attr2),
	};

UML	Binary format definition

 class UML Transition
«DataStructure»
DataStructureWithOptCompList
«OrderedComponentGroup»
+ attr1: childComponent [0..n]
	<DataStructureWithOptCompList>:=
	<BitArray>(selector), : selector
	if (bit 0 of selector is set)
	{
	<IntUnLoMB>(n),
	n * **<childComponent>**(attr1), : optional component
	};

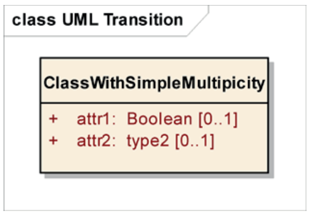
NOTE 2 Components included in other Components require no multiplicity indication and selector entry as a component is identifiable by its ID and carries its own length information. The number of Components within the Component can be derived from the containing Component length attribute.

NOTE 3 DataStructures cannot include classes as <<UnorderedComponentGroup>>.

5.5.9 Rule 4e: Multiplicity [0..1]

Attributes with multiplicity [0..1] not stereotyped as <<OrderedComponentGroup>> or <<UnorderedComponentGroup>> require a selector that indicates the presence of the attribute. See Rule 3 for placement rules of the selector.

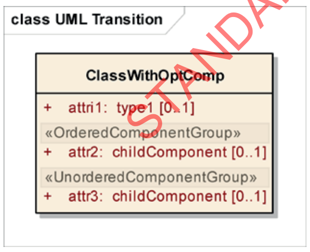
Presence of Booleans with multiplicity [0..1] is not indicated in a selector. Booleans with multiplicity [0..1] are represented using the table typ008:OptionalBoolean.

UML	Binary format definition
 class UML Transition ClassWithSimpleMultiplicity + attr1: Boolean [0..1] + attr2: type2 [0..1]	**<ClassWithSimpleMultiplicity(gcid)>**:= **<IntUnTi>**(gcid), : id of this component **<IntUnLoMB>**(lengthComp), : number of bytes in component **<IntUnLoMB>**(lengthAttr), : number of bytes in attributes **<typ008:OptionalBoolean>**(attr1), : Optional Boolean **<BitArray>**(selector), if (bit 0 of selector is set) **<type2>**(attr2); : optional attribute

For Components with multiplicity [0..1], included as attribute (stereotyped as <<OrderedComponentGroup>> or <<UnorderedComponentGroup>>), the following applies.

If the Component is included in a class that is not stereotyped as <<DataStructure>> (a Component), the selector shall not be used for indicating the presence of the Component. If the Component is included in a class that is stereotyped as <<DataStructure>>, the Component shall be treated as a regular attribute.

UML	Binary format definition

 class UML Transition
ClassWithOptComp
+ attr1: type1 [0..1]
«OrderedComponentGroup»
+ attr2: childComponent [0..1]
«UnorderedComponentGroup»
+ attr3: childComponent [0..1]
	<ClassWithOptComp(gcid)>:=
	<IntUnTi>(gcid), : id of this component
	<IntUnLoMB>(lengthComp), : number of bytes in component
	<IntUnLoMB>(lengthAttr), : number of bytes in attributes
	<BitArray>(selector), : selector
	if (bit 0 of selector is set)
	<type1>(attr1), : optional attribute
	ordered {
	n * **<childComponent>**(attr2)[0..1], : optional component
	}
	unordered {
	n * **<childComponent>**(attr3)[0..1], : optional component
	};