TECHNICAL SPECIFICATION

ISO/TS 21219-3

First edition
2015-03-01

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

Part 3: UML to binary conversion rules

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/TS 21219-3:2015(E)

Contents

Page

Foreword ..... **iv**

Introduction ..... **vi**

1 Scope **1**

2 Normative references **1**

3 Abbreviated terms **1**

4 Rules for UML to binary format description conversion **1**

4.1 Definition of binary format description 1

4.2 Abstract data types 3

4.3 Binary format specific data types 7

4.4 TPEG tables 8

4.5 Compound data types 8

4.5.1 Rule 1: Classes 8

4.5.2 Rule 2: Datastructures 8

4.5.3 Rule 3: Selector 8

4.5.4 Rule 4: Attributes 9

4.5.5 Rule 4a: Datatypes 9

4.5.6 Rule 4b: Ordering 9

4.5.7 Rule 4c: Single multiplicity 10

4.5.8 Rule 4d: Multiplicity [0..n] and Multiplicity [1..n] 10

4.5.9 Rule 4e: Multiplicity [0..1] 11

4.6 Rule 5: Aggregations and compositions 12

4.7 Rule 6: Specialisations / Abstract classes 13

Bibliography **14**

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 and TISA 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 on the meaning of ISO specific terms and expressions related to conformity assessment, as well as information about ISO's adherence to the WTO principles in the Technical Barriers to Trade (TBT) see the following URL: Foreword - Supplementary information

The committee responsible for this document is ISO/TC 204 *Intelligent transport systems*, in cooperation with the Traveller Information Services Association (TISA), TPEG Applications Working Group through Category A Liaison status.

ISO/TS 21219 consists of the following parts, under the general title *Intelligent transport systems — Traffic and travel information (TTI) via transport protocol experts group, generation 2 (TPEG2)*:

— *Part 2: UML modelling rules* [Technical Specification]

— *Part 3: UML to binary conversion rules* [Technical Specification]

— *Part 4: UML to XML conversion rules* [Technical Specification]

— *Part 5: Service framework* [Technical Specification]

— *Part 6: Message management container* [Technical Specification]

— *Part 7: Location referencing container* [Technical Specification]

— *Part 18: Traffic flow and prediction application* [Technical Specification]

The following parts are planned:

— *Part 1: Introduction, numbering and versions* [Technical Specification]

— *Part 9: Service and network information* [Technical Specification]

— *Part 10: Conditional access information* [Technical Specification]

— *Part 14: Parking information application* [Technical Specification]

— *Part 15: Traffic event compact application* [Technical Specification]

— *Part 16: Fuel price information application* [Technical Specification]

— *Part 19: Weather information application* [Technical Specification]
— *Part 20: Extended TMC location referencing* [Technical Specification]
— *Part 21: Geographic location referencing* [Technical Specification]
— Part 22: *OpenLR·location·referencing* [Technical Specification]
— Part 23: *Roads·and·multi-modal·routes·application* [Technical Specification]

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/WG 4, in conjunction with ISO/TC 204/WG 10, established a project group comprising members of the former EBU B/TPEG and they continued the work concurrently. 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 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

With the inauguration of the Traveller Information Services Association (TISA) in December 2007 derived from former Forums and the CEN/ISO development project group, the TPEG Applications Working Group took over development work for TPEG technology.

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 Parts 2, 3, 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 (Part 6), Application (many Parts) and Location Referencing (Part 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:

Toolkit parts: TPEG2-INV (Part 1), TPEG2-UML (Part 2), TPEG2-UBCR (Part 3), TPEG2-UXCR (Part 4), TPEG2-SFW (Part 5), TPEG2-MMC (Part 6), TPEG2-LRC (Part 7)

Special applications: TPEG2-SNI (Part 9), TPEG2-CAI (Part 10)

Location referencing: TPEG2-ULR (Part 11), TPEG2-ETL (Part 20), TPEG2-GLR (Part 21), TPEG2-OLR (Part 22)

Applications: TPEG2-PKI (Part 14), TPEG2-TEC (Part 15), TPEG2-FPI (Part 16), TPEG2-TFP (Part 18), TPEG2-WEA (Part 19), TPEG2-RMR (Part 23)

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 Technical Specification is based on the TISA specification technical/editorial version number: TPEG2-UBCR/1.1/001.

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

Part 3: **UML to binary conversion rules**

1 Scope

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

2 Normative references

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

ISO/TS 18234-2:2013, *Intelligent transport systems —Traffic and travel information (TTI) via transport experts group, generation 1 (TPEG1) binary data format — Part 2: Syntax, semantics and framing structure (TPEG1-SSF)*

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

ISO/TS 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 Abbreviated terms

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

LSB	Least Significant Bit
MSB	Most Significant Bit

4 Rules for UML to binary format description conversion

4.1 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/TS 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]*.

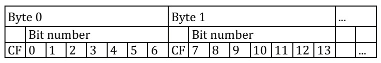
The *Application components* description shall have a first subclause with title *List of generic component IDs*. This clause contains unique component IDs for each application UML class that is not stereotyped as <<DataStructure>>. The component IDs should be ordered in the order of appearance in the 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 4.5. 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 4.5 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 4.5.

4.2 Abstract data types

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

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 figure 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).  Byte 0 Byte 1 ... Bit number Bit number CF 0 1 2 3 4 5 6 CF 7 8 9 10 11 12 13 ... **Figure 1 — Binary format coding of BitArray** NOTE: 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/TS 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 Annex D of ISO/TS 18234-2:2013.

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
FixedPointNumber	**<FixedPointNumber>**:= **<IntSiLoMB>**(integerPart), : integer part of the number **<IntUnTi>**(decimalPart); : fraction of 2 decimal digits [0...99]
Float	**<Float>**:= 4 * **<byte>;** : ISO/IEC/IEEE 60559 single precision floating point number [4] 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: 31 (bit #) Exponent 23-30 (bit #) Mantissa 0-22 (bit #)
IntSiTi	**<IntSiTi>**:= **<byte>**; : Two's complement

IntSiLi	**<IntSiLi>**:=
	  <byte>, : MSB, two's complement
	  <byte>; : LSB, two's complement

Data type	**Binary format definition**
IntSiLo	**<IntSiLo>**:= **<byte>**, : MSB, two's complement **<byte>,** **<byte>,** **<byte>**; : LSB, two's complement
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 byte #5 shall be set to 111 in case of negative numbers with 5 byte length and 000 otherwise, to be up-ward compatible in case of introduction of a 64-bit integer value in future. Signed values from 0 to −26 are stored in one byte, to −213 in two bytes, to −220 in three bytes, to −227in four bytes and to −232 in five bytes. EXAMPLE A value 62 hex (0110 0010) would be encoded with the one byte 0x62. The integer value A7 hex (1010 0111) would be encoded with a two-byte sequence 0x8127. The signed representation of −1 is 0x7F. And −2345 is represented in two bytes so that the complement on two is 0x36D7 = (110 1101.101 0111). A serialisation in multi-byte then results in 1110 1101.0101 0111 = 0xED57.
IntUnTi	**<IntUnTi>**:= **<byte>**; : Primitive
IntUnLi	**<IntUnLi>**:= **<byte>**, : MSB **<byte>**; : LSB
IntUnLo	**<IntUnLo>**:= **<byte>**, : MSB **<byte>,** **<byte>,** **<byte>**; : LSB
IntUnLoMB	**<IntUnLoMB>**:= m * **<byte>**[1..5]; : MSB of each byte is continuation flag. Three LSBs from last 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 as "reserved for future use". This leads into the maximum number $2^{(32)}$ - 1 which is the maximum value of a four byte unsigned integer.

Data type	**Binary format definition**
ShortString	**<ShortString>**:= **<IntUnTi>**(n), : Number of bytes, n n * **<byte>**; : String of characters. The number of characters depends on the chosen character set
LongString	**<LongString>**:= **<IntUnLi>**(n), : Number of bytes, n n * **<byte>**; : String of characters. The number of characters depends on the chosen character set.
LocalizedShortString	**<LocalizedShortString>**:= **<typ001:LanguageCode>**(languageCode), : Specifies the language used for this string. **<ShortString>**(string); : Short string
LocalizedLongString	**<LocalizedLongString>**:= **<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

Data type	Binary format definition
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].
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

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.

4.3 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.

4.4 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>**:= as defined in ISO/TS 21219-2 **<IntUnTi>**(code); : Language code

4.5 Compound data types

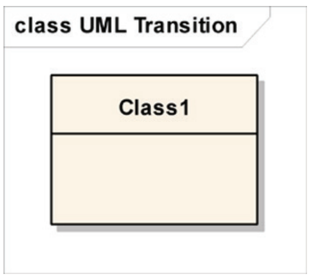
4.5.1 Rule 1: Classes

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

— generic component ID;

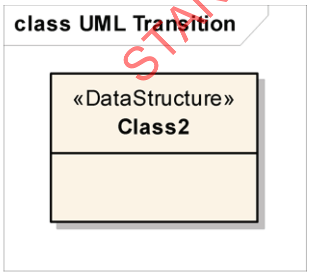
— total length of the component in bytes;

— 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 **<IntUnLoMB>**(lengthAttr), : number of bytes in attributes **<...>**; : component data

4.5.2 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.

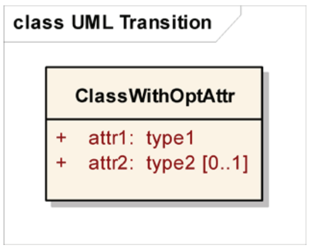
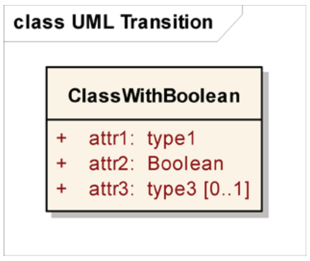
UML	Binary format definition
	**<Class2>**:= **<...>**; : content definition

4.5.3 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
 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

4.5.4 Rule 4: Attributes

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

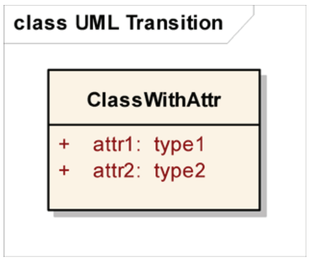
4.5.5 Rule 4a: Datatypes

Attributes of primitive data type shall be converted into binary format as defined in 4.2, 4.3 and 4.4.

Attributes of compound data type shall be converted into binary format according to the rules in this clause (4.5).

4.5.6 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

4.5.7 Rule 4c: Single multiplicity

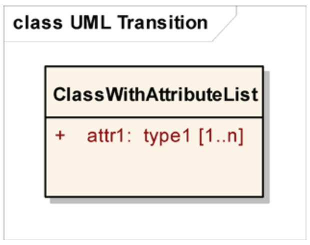
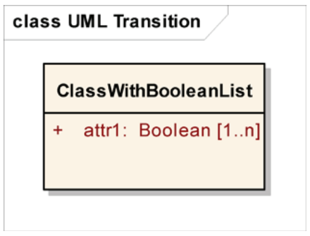
No additional rules.

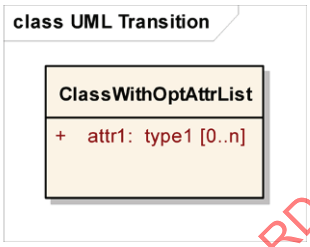
NOTE The explicit notation for single multiplicity ([1]) may be omitted in a model. If not stated otherwise, an attribute has single multiplicity.

4.5.8 Rule 4d: Multiplicity [0..n] and Multiplicity [1..n]

Attributes with multiplicity [0..n] or [1..n] 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 multiplicity [0..n] 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 4.3).

UML	**Binary format definition**
 class UML Transition ClassWithAttributeList + attr1: type1 [1..n]  class UML Transition ClassWithBooleanList + attr1: Boolean [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 **<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

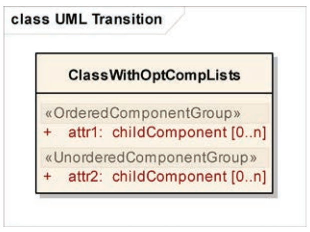
 class UML Transition
ClassWithOptAttrList
+ attr1: type1 [0..n]
	<ClassWithOptAttrList(gcid)>:=
	<IntUnTi>(gcid), : id of this component
	<IntUnLoMB>(lengthComp), : number of bytes in component
	<IntUnLoMB>(lengthAttr), : number of bytes in attributes
	<BitArray>(selector), : selector
	if (bit 0 of selector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

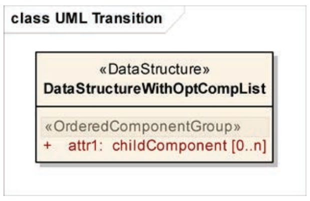
NOTE 1 If n equals zero, the multibyte n is still present having value zero, but no attributes are instantiated.

For Components with multiplicity [0..n] or [1..n], included as attribute (stereotyped as <<OrderedComponentGroup>> or <<UnorderedComponentGroup>>) or as aggregation, 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
	n * **<childComponent>**(attr1), : the component list unordered {
	n * **<childComponent>**(attr2),
	};

 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
	};

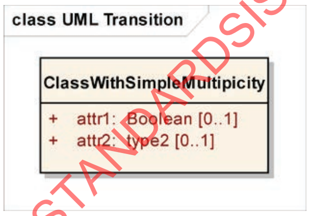
NOTE 2 Components included in other Components require no multiplicity indication and selector entry as a component is identifiable by its ID and carries its own length information. The number of Components within the Component can be derived from the containing Component length attribute.

NOTE 3 DataStructures cannot include classes by aggregation or as <<UnorderedComponentGroup>>.

4.5.9 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 ClassWithSimpleMultipicity + 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, as defined in ISO/TS 21219-2 **<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>>) or as aggregation, the following applies: