TECHNICAL SPECIFICATION

ISO/TS 18234-10

First edition
2013-10-15

Intelligent transport systems — Traffic and travel information via transport protocol experts group, generation 1 (TPEG1) binary data format —

Part 10: Conditional access information (TPEG1-CAI)

Systèmes intelligents de transport — Informations sur le trafic et le tourisme via les données de format binaire du groupe d'experts du protocole de transport, génération 1 (TPEG1)

Partie 10: Information d'accès conditionnel (TPEG1-CAI)

Reference number
ISO/TS 18234-10:2013(E)

ISO copyright office
Case postale 56 • CH-1211 Geneva 20
Tel. + 41 22 749 01 11
Fax + 41 22 749 09 47
E-mail copyright@iso.org
Web www.iso.org

Published in Switzerland

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.

International Standards are drafted in accordance with the rules given in the ISO/IEC Directives, Part 2.

The main task of technical committees is to prepare International Standards. Draft International Standards adopted by the technical committees are circulated to the member bodies for voting. Publication as an International Standard requires approval by at least 75 % of the member bodies casting a vote.

In other circumstances, particularly when there is an urgent market requirement for such documents, a technical committee may decide to publish other types of normative document:

— an ISO Publicly Available Specification (ISO/PAS) represents an agreement between technical experts in an ISO working group and is accepted for publication if it is approved by more than 50 % of the members of the parent committee casting a vote;

— an ISO Technical Specification (ISO/TS) represents an agreement between the members of a technical committee and is accepted for publication if it is approved by 2/3 of the members of the committee casting a vote.

An ISO/PAS or ISO/TS is reviewed after three years in order to decide whether it will be confirmed for a further three years, revised to become an International Standard, or withdrawn. If the ISO/PAS or ISO/TS is confirmed, it is reviewed again after a further three years, at which time it must either be transformed into an International Standard or be withdrawn.

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.

ISO/TS 18234-10 was prepared by the European Committee for Standardization (CEN) Technical Committee CEN/TC 278, *Road transport and traffic telematics,* in collaboration with ISO Technical Committee ISO/TC 204, *Intelligent transport systems*, in accordance with the Agreement on technical cooperation between ISO and CEN (Vienna Agreement).

ISO/TS 18234 consists of the following parts, under the general title *Intelligent transport systems — Traffic and travel information via transport protocol experts group, generation 1 (TPEG1) binary data format*:

— *Part 1: Introduction, numbering and versions (TPEG1-INV)*

— *Part 2: Syntax, semantics and framing structure (TPEG1-SSF)*

— *Part 3: Service and network information(TPEG1-SNI)*

— *Part 4: Road Traffic Message application (TPEG1-RTM)*

— *Part 5: Public Transport Information (PTI) application*

— *Part 6: Location referencing applications*

— *Part 7: Parking information (TPEG1-PK1)*

— *Part 8: Congestion and travel-time application (TPEG1-CTT)*

— *Part 9: Traffic event compact (TPEG1-TEC)*

— *Part 10: Conditional access information (TPEG1-CAI)*

— *Part 11: Location Referencing Container (TPEG1-LRC)*

Introduction

TPEG technology uses a byte-oriented data stream format, which may be carried on almost any digital bearer with an appropriate adaptation layer. TPEG-messages are delivered from service providers to end-users and used to transfer information from the database of a service provider to an end-user's equipment.

The brief history of TPEG technology development dates back to the European Broadcasting Union (EBU) Broadcast Management Committee establishing the B/TPEG project group in autumn 1997 with the mandate 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 are 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.

One year later in December 1998, the B/TPEG group produced its first EBU specifications. Two Technical Specifications were released. ISO/TS 18234-2, described the Syntax, Semantics and Framing Structure, which is used for all TPEG applications. ISO/TS 18234-4 (TPEG-RTM) described the first application, for Road Traffic Messages.

Subsequently, CEN/TC 278/WG 4, in conjunction with ISO/TC 204, established a project group comprising the members of B/TPEG and they have continued the work concurrently since March 1999. Since then two further parts were developed to make the initial complete set of four parts, enabling the implementation of a consistent service. ISO/TS 18234-3 (TPEG-SNI) describes the Service and Network Information Application, which should be used by all service implementations to ensure appropriate referencing from one service source to another. ISO/TS 18234-1 (TPEG-INV), completes the series, by describing the other parts and their relationship; it also contains the application IDs used within the other parts. Additionally ISO/TS 18234-5 the Public Transport Information Application (TPEG-PTI) and ISO/TS 18234-6 (TPEG-LRC), were developed.

TPEG applications are developed using UML modelling and a software tool is used to automatically select content which then populates this TS. Diagrammatic extracts from the model are used to show the capability of the binary coding in place of lengthy text descriptions; the diagrams do not necessarily include all relevant content possible.

This Technical Specification describes the binary data format of the on-air interface of the Conditional Access Information application, (TPEG-CAI) with the technical version number TPEG-CAI_1.0/001.

CAI application

The basic concept behind the CAI application is to transport CAI in separate TPEG service components of a dedicated application type and to define an SNI table that contains the link between scrambled content and related CAI.

Intelligent transport systems — Traffic and travel information via transport protocol experts group, generation 1 (TPEG1) binary data format —

Part 10:
Conditional access information (TPEG-CAI)

1 Scope

This Technical Specification contains the definition of the TPEG Conditional Access Information (CAI) application. It enables dedicated conditional access data, such as management messages (e.g. Control Words and Entitlement Control Messages) to be delivered to recipient client devices. This TPEG application is designed for a service provider to: establish setup, prolongation or revocation of services to a specific client device, using a limited capacity unidirectional broadcast channel and without recourse to service-client handshaking.

This TPEG application defines:

— the logical channel, for the transmission of the additional CA information (CAI);

— how the CAI is linked and synchronized to the scrambled content.

This Technical Specification is related to conditional access applied at the service component level of a TPEG service. It is an open design for the integration of various different conditional access systems, externally specified, which are signalled by the TPEG service Encryption Indicator to allow client devices to operate correctly.

2 Normative References

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

ISO/TS 18234-1, *Intelligent transport systems — Traffic and travel information via transport protocol experts group, generation 1 (TPEG1) binary data format — Part 1: Introduction, numbering and versions (TPEG1-INV)*

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

ISO/TS 18234-3, *Intelligent transport systems — Traffic and travel information via transport protocol experts group, generation 1 (TPEG1) binary data format — Part 3: Service and network information (TPEG1-SNI)*

3 Abbreviated terms

For the purposes of this document, the following abbreviated terms apply.

AID	Application Identification
CA	Conditional Access
CAI	Conditional Access Information
CRC	Cyclic redundancy check
ECM	Entitlement Control Message
EMM	Entitlement Management Message
TPEG	Transport Protocol Expert Group
SSF	Syntax, Semantics and Framing Structures
TTI	Traffic and Traveller Information

4 Application identification and version number signalling

4.1 Application identification

The word 'application' is used in the TPEG specifications to describe specific subsets of the TPEG structure. An application defines a limited vocabulary for a certain type of messages, for example parking information or road traffic information. Each TPEG application is assigned a unique number, called the Application Identification (AID). An AID is defined whenever a new application is developed and these are all listed in ISO/TS 18234-1.

The application identification number is used within the TPEG-SNI application to indicate how to process TPEG content and facilitates the routing of information to the appropriate application decoder.

4.2 Version number signalling

Version numbering is used to track the separate versions of an application through its development and deployment. The differences between these versions may have an impact on client devices.

The version numbering principle is defined in ISO/TS 18234-1.

Table 1 shows the current version numbers for signalling CAI within the SNI application:

Table 1 — Current version numbers for signalling of CAI

major version number	1
minor version number	0

5 Service Component Data

TPEG-CAI makes use of the "Service component data with dataCRC" according to Annex A, section A.3.2.6.2.1. For explanatory purposes, this is repeated here.

< **ServCompFrameProtected** >:=	: CRC protected service component frame
<**ServCompFrameHeader**>(header),	: Component frame header as defined in A.3.2.6.1
external <**ApplicationContent**>(content),	: Content specified by the individual application
<**CRC**>(dataCRC);	: CRC starting with first byte after the header

The main frame of CAI defines ApplicationContent as follows:

<**ApplicationContent**>:=	: application content
messageCount * <**CAIMessage**>(msg);	: Any number of any CAI message components

6 Conditional Access Methodology

Conditional access (CA) is specified within TPEG-SSF and TPEG-SNI as a function being applied on service frame or service component level. The method used is indicated via the Encryption Identifier (EncID) directly in the service frame or for components via the SNI Fast Tuning Table (Guide to the Services 1). This specification is related to conditional access applied on service component level.

Generally, a broadcast based CA-system requires encryption related data to be transmitted which is independent from the content, but necessary for decryption and subscriber management.

If a conditional access system is applied on the TPEG service component level, some service components may be encrypted using the same "encryption key", while others remain unencrypted or use different "encryption keys". Therefore, several service components can share the same conditional access information , if they are supposed to be offered as one bundle and hence are encrypted with the same keys.

Each of the aforementioned bundles may require CA-management-messages, which have to be transmitted separated from the (encrypted) content in the corresponding service components. The most appropriate way for the transport is the use of separate service components of a dedicated application type.

For each encrypted TPEG-Service component a link or reference to the service component carrying the relevant CA information is required. This is handled by TPEG-SNI GST-Table 6, Conditional Access Information Reference.

EXAMPLE

A TPEG Service may contain the following service components:

SCID	Application
0	SNI
2	TEC
5	TEC (encrypted)
7	TEC (encrypted)
8	PTI
10	PKI (encrypted)
20	CAI
21	CAI
30	CAI

The service components 5 and 7 are encrypted with key 1, while service component 10 is encrypted using key 2. Hence two components with CA-meta information for the corresponding component are required, in the example listed as SCID 20 and 21. A third CAI component, in the example number 30, contains CA-meta information that relates to all encrypted components independent which key is applied.

This specification describes the generic containers for the CAI application. The container content will be proprietary and specified individually for each CA-System indicated by the encryption indicator (EncID). The linking between encrypted service components and related CAI-Components is achieved via a reference table within the TPEG-SNI application.

7 Message Components

Unlike other TPEG applications, TPEG-CAI does not use a Message Management Container and does not use a Location Referencing Container; it only uses an Application Event Container.

Figure 1 visualises the logical structure of the Conditional Access Information (CAI) application.

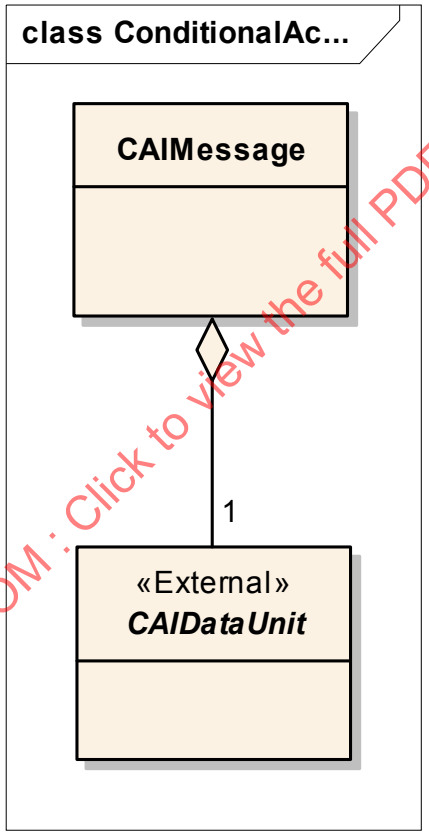

Figure 1 — Logical structure of CAI application

7.1 List of Generic Component Ids

Name	Id
CAIMessage	1

7.2 CAIMessage

A TPEG-CAI Message includes solely one single container for proprietary CA data.

The CAI Message Container is available to carry data, which is defined within the CA system specific specifications. The CAIDataUnit is directly following after the lengthAttr of the CAIMessage.

<CAIMessage(1)>:=	
<IntUnTi>(id)**,**	: Identifier = 1
<IntUnLoMB>(lengthComp),	: Length of component in bytes, excluding : the id and length indicator
<IntUnLoMB>(lengthAttr),	: Length of attributes
<CAIDataUnit>(data);	: CAI data

7.3 CAIDataUnit

The CAIDataUnit carries the data that is specified by the corresponding conditional access specification.

<CAIDataUnit>:=	CAI data
m***<byte>;**	:proprietary CA data

Annex A
(normative)

Binary SSF and Data Types

A.1 Conventions and symbols

A.1.1 Conventions

A.1.1.1 Byte ordering

All numeric values using more than one byte are coded in “Big Endian” format (most significant byte first). Where a byte is subdivided into bits, the most significant bit (“b7”) is at the left-hand end and the least significant bit (“b0”) is at the right-hand end of the structure.

A.1.1.2 Method of describing the byte-oriented protocol

TPEG uses a data-type representation for the many structures that are integrated to form the transmission protocol. This textual representation is designed to be unambiguous, easy to understand and to modify, and does not require a detailed knowledge of programming languages.

Data types are built up progressively. Primitive elements, which may be expressed as a series of bytes are built into compound elements. More and more complex structures are built up with compound elements and primitives. Some primitives, compounds and structures are specified in this Technical Specification, and apply to all TPEG Applications. Other primitives, compounds and structures are defined within applications and are local only to that application.

A resultant byte-stream coded using C-type notation is shown in ISO/TS 18234-2:2006, Annex E.

A.1.1.3 Reserved data fields

If any part of a TPEG data structure is not completely defined, then it should be assumed to be available for future use. The notation is UAV (unassigned value). This unassigned value should be encoded by the service provider as the value 00 hex. This allows newer decoders using a future TPEG Standard to ignore this data when receiving a service from a provider encoding to this older level of specification. A decoder which is not aware of the use of any former UAVs can still make use of the remaining data fields of the corresponding information entity. However, the decoder will not be able to process the newly defined additional information.

A.1.2 Symbols

A.1.2.1 Literal numbers

Whenever literal numbers are quoted in TPEG Standards, the following applies:

123 = 123 decimal

123 hex = 123 hexadecimal

A.1.2.2 Variable numbers

Symbols are used to represent numbers whose values are not predefined within the TPEG Standards. In these cases, the symbol used is always local to the data type definition. For example, within the definition of a data type, symbols such as “n” or “m” are often used to represent the number of bytes of data within the structure, and the symbol “id” is used to designate the occurrence of the identifier of the data type.

A.1.2.3 Implicit numbers

Within the definition of a data structure it is frequently necessary to describe the inclusion of a component which is repeated any number of times, zero or more. In many of these cases it is convenient to use a numerical symbol to show the component structure being repeated a number of times, but the number itself is not explicitly included within the definition of the data structure. Often, the symbol “m” is used for this purpose.

A.2 Representation of syntax

A.2.1 General

This clause introduces the terminology and the syntax that is used to define TPEG data elements and structures.

A.2.2 Data type notation

A.2.2.1 Rules for data type definition representation

The following general rules are used for defining data types:

— a data type is written in upper camel case letters in one single expression.[1] The data type may contain letters (a-z), number (0-9), underscore "_", round brackets "()" and colon ":"; the first must be a letter;

EXAMPLE 1 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] Camel case is the description given to the use of compound words wherein each individual word is signalled by a capital letter inside the compound word. Upper camel case means that the compound word begins with an upper-case (capital) letter, and lower camel case means the compound word begins with a small letter.

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. (See Clause A2.3.3 - Components);

4) “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

EXAMPLE 2

<MMCLink(1)>:=	: externally defined component
external <MessageManagementContainer(1)>;	: id = 1, See Annex B (Message Management Container)

— 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

EXAMPLE 3

m * **<IntUnTi>**(Attribute) [1..*] ,	: The “Attribute” must occur once at least and up to infinite.

— a function “ f_n ( ) ” that is calculated over a data type is indicated by italic lower case letters. The comment behind the definition of the function shall explain which function is used;

— any text after a colon “ : ” is regarded as a comment;

— a data type definition can be a *template* (i.e. not fully defined declarative structure) having a *parameter* inside of round brackets "(x)" at the end of the data type name. Templates define structures, whose structural definition is included as a basis for other data type definitions. To declare the given template (making it identifiable) the name of the parameter is repeated as a descriptor in a nested data type of the subsequent definition list. Templates allow for reading the generalised part of different instances i.e. to specify data type interfaces. (See Clause A2.3.2 - Using templates as interfaces for further description)

EXAMPLE 4

<Template(x)> :=	: x defines the template parameter
<IntUnTi>(x);	: descriptor x defines position of setting the parameter in the list

— a data type can *inherit* a template by concatenating the data type name of the template including the square brackets to its own name. The data type itself can again be a template having the "(x)" at its end of name, or it instantiates the inherited template by defining the value of the parameter in the brackets. In the latter case the brackets shall contain the **decimal** number of the identifier and the value shall be set in the subsequent definition list. The structural definition of the inherited template is repeated as the first part of the definition list before new data types are specified. (See Clause A2.3.2 - Using templates as interfaces for further description)

EXAMPLE 5

<AnotherTemplate(x)<Template(x)>>:=	: second template inherits first
<IntUnTi>(x),	: repeated definition from 1st template
<IntUnLi>(n);	: additional structural definition
<Instance<AnotherTemplate(1)>>:=	**: instantiation of the second template**
<IntUnTi>(1),	: definition of parameter in the stream
<IntUnLi>(n),	: structural definition from template
<IntUnTi>(value);	: some more definition

— in the definition list a specific instance of a template (i.e. declarative structure) is described without the brackets. Any inherited data type of this template may occur at that position in the data stream

EXAMPLE 6

<SomeData>:=	
<AnotherTemplate>(anyAnotherTemplate);	: Data stream contains e.g. **<Instance>**

The following additional guidelines help to improve the readability of data type definitions:

— data type names are written in bold;

— nested data type definitions are defined from top to bottom (i.e. higher levels first, then lower levels);

— a box is drawn around a data type definition;

— 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

EXAMPLE 7

<LongLinesExample>:=	
<DateTimeVeryLongType	: First line
\NameMayBeInSeveralLines>,	: Second line
<DateTime>,	
<ShortString>;	

A.2.2.2 Description of data type definition syntax

A data type is an interpretation of one or more bytes. Each data type has a structure, which may describe the data type as a composition of other defined data types. The data type structure shows the composition and the position of each data element. TPEG defines data structures in the following manner:

<NewDataType>:=	: Description of data type
<DataTypeA>(descriptorA),	: Description of data A
<DataTypeB>(descriptorB);	: Description of data B

This shows an example data structure, which has just two parts, one of type **<DataTypeA>** and the other of **<DataTypeB>**. A descriptor may be assigned to the data type, to relate the element to another part of the definition. Comments about the data structure are included at the right-hand side delimited by the colon ":" separator. Each of the constituent data types may be itself composed of other data types, which are defined separately. Eventually each data type is expressible as one or more bytes.

Where a data structure is repeated a number of times, this may be shown as follows:

<NewDataType>:=	: Description of data type
<DataTypeA>,	: Description of data A
m * **<DataTypeB>**[0..*];	: Description of data B

Often, in such cases it is necessary to explicitly deliver to the decoder the number of times a data type is repeated; sometimes it is not, because other means like framing or internal length coding allows knowledge of the end of the list of the repeated data type. In other cases the overall length of a data structure in bytes needs to be specified. Additionally the constraint on occurrences can be added, which tells how many instances of the data type must be expected by the decoder. The "*" as upper bound means in this case that at this place no restriction is given to the upper bound; in other words, infinite elements may follow.

Where the number of repetitions must be signalled, it may be accomplished using another data element as follows:

<NewDataType>:=	: Description of data type
<IntUnTi>(n),	: An integer representing the value of "n"
n * **<DataTypeA>**[0..255],	: Description of data A
<DataTypeB>;	: Description of data B

In the above example a decoder has to have the value of "n" in order to correctly determine the n'th position of the **<DataTypeB>** in the list. Here as consequence of data type IntUnTi not more as 255 instances of the data type can be coded.

In the following example the decoder uses the value of "n" to determine the overall length of the data structure, and the value of "m" determines that **<DataTypeB>** is repeated m times:

<NewDataType>:=	: Description of data type
<IntUnTi>(n),	: Length, n, of data structure in bytes
m * **<DataTypeA>;**	: Description of data A

This data type definition is used to describe a variable structure switched by the value of x:

<NewDataType>:=	: Description of data type
<IntUnTi>(x),	: Select parameter, x
if (x=1) then **<DataTypeA>,**	: Included if x equals 1
if (x=2) then **<DataTypeB>,**	: Included if x equals 2

A.2.3 Application dependent data types

This clause describes the methodology and syntax by which application data types may be constructed within TPEG Applications. Two basic forms are described: data structures (being non-declarative) and components (being declarative). Components contain an identifier which labels the structure, and which can be used by a decoder to determine the definition of content of the structure. As such, components are used where options are required, or where an application needs to build in 'future proofing'. Data structures do not contain such information, and are used in all other positions.

This Annex does not specify the structures, which are actually used in TPEG Applications. Such specifications are made in the respective parts of the Standard. However examples are given to show how such structures may be built from the primitive elements already described above.

A.2.3.1 Data structures

Data structures are built up from several (i.e. more than one) elements: primitive, compound or other structures (both non-declarative and declarative). As such, any application specific data type definition having no component identifier is per definition a data structure. The term data structure is specifically used for data type definitions having more than one sub element defined.

Examples of data structure might be:

EXAMPLE 1

<Activity>:=	: Activity
<DateTime>,	: Beginning
<DateTime>,	: End
<ShortString>;	: Text

EXAMPLE 2

<Wave>:=	: Sound sample
<IntUnLi>(n),	: Length of samples, n
n * **<IntSiTi>**(sample)[0..8000];	: Between 0 and 8000 occurrences of a sample

Another example making use of a condition within a data type definition is shown below.

EXAMPLE 3 An application could use the example data types above in the following way

<Appointment>:=	: Appointment
<IntUnTi>(at),	: Alarm type
if (at = 1)	
<WaveAlarm>,	: Remind with a sound
if (at = 2)	
<TextAlarm>,	: Remind with a text
<Activity>;	: Let some action follow

<WaveAlarm>:=	: Sound alarm
<DateTime>,	: When to wake up
<Wave>;	: Sound to wake up to!

<TextAlarm>:=	: Text alarm
<DateTime>,	: When to display
<ShortString>;	: Text to display

For optional values a general mechanism is provided, using a bitarray for signalling optional values. In the case that a corresponding bit of the bitarray is set (=1), the optional attribute is stored in the stream. In case the bit is unset the attribute is not available and the next following attribute shall be processed in the stream.

EXAMPLE 4 Data structure with optional elements, signalled by a preceding bitarray as selector

<TimeInterval>:=	
<BitArray>(selector),	: DaySelector
if (bit 0 of selector is set)	
<IntUnTi>(years),	: Number of years between 0 and 100
if (bit 1 of selector is set)	
<IntUnTi>(months),	: Number of months between 0 and twelve
if (bit 2 of selector is set)	
<IntUnTi>(days),	: Number of days between 0 and 31
if (bit 3 of selector is set)	
<IntUnTi>(hours),	: Number of hours between 0 and 24
if (bit 4 of selector is set)	
<IntUnTi>(minutes),	: Number of minutes between 0 and 60
if (bit 5 of selector is set)	
<IntUnTi>(seconds);	: Number of seconds between 0 and 60

A.2.3.2 Using templates as interfaces

In addition to the possibility of coding the complete and static structural definition of a data structure, the syntax does foresee that parts of the structure are conditionally different; signalled by a well defined first part some other data types are different.

EXAMPLE

A tagged value (also known as TagLengthValue-Coding) starts with a type and length; afterwards the value follows. Let's assume the type is an enumeration of some possible values, one would first specify the interface having only the type defined. The different tagged value types would now inherit this interface, i.e. would have the type defined as first element amended with the definition of the tagged value data type. The decoder now reads the interface information (the type attribute) and knows how to proceed for reading the rest of the tagged value from the stream.

<DifferentDataList>:=	: A list of data
n * **<TaggedValue>**(value)**;**	: Different instances can have different types

<TaggedValue(x)>:=	: Template for tagged value
<tav001:ValueType>(type)**,**	: Type of this tagged value
<IntUnTi>(length)**;**	: Length in bytes in case that value type is unknown

Example table tav001:ValueType:

Code	Reference-English 'word'	Comment
001	Service name	
002	Price per month	

Then the resulting list of inherited tagged value data types would be:

<ServiceName<TaggedValue(1)>>:=	: Template for tagged value
<tav001:ValueType>(1)**,**	: Type of this tagged value
<IntUnTi>(length)**,**	: Length in bytes in case that value type is unknown
<**ShortString**>(serviceName)**;**	: Service name

<ServiceName<TaggedValue(2)>>:=	: Template for tagged value
<tav001:ValueType>(2),	: Type of this tagged value
<IntUnTi>(length),	: Length in bytes in case that value type is unknown
<**Float**>(pricePerMonth);	: Price per month

This interface allows a subsequent list of data types which can easily be extended, by using the same interface.

A.2.3.3 Components

A component is understood as a declarative structure having an interface as described in the previous clause. A decoder of the data stream can identify the content of the structure with the help of the identifier which is unique in the scope of any one TPEG Application Standard. In addition to the identifier a length indicator allows the decoder to step over those components whose ids are unknown to it. This enables the possibility of introducing new components in the data stream although decoders in the market do not know their content. The old decoder does expect the content of the first version of a protocol and ignores simply unrecognized data with small performance loss. The new decoder expects the second version of the protocol and can fully decode that version of the protocol. Components should be used wherever *future extensions* are envisioned, and where 'future proofing' is a strong requirement.

NOTE With this method even non-backwards compatible changes can be introduced into the existing market by having a migration period being backward compatible and then later cutting of not longer supported devices, even though it is expected that the migration will take its time.

In Addition to the concept of declarative structuring a second step of improvement of size efficiency combined with the backward compatibility is specified. The first part following the header of a component in the data stream is defined as *attribute block*. The attribute block starts with the length of the block in bytes which again allows the decoder to step over attributes that are not specified in a first version of the protocol.

The decoder reads the attribute block length and decreases the count of bytes while reading the attributes in case that the last known attribute is read, and the attribute block count is not zero, the remaining bytes in the data stream are omitted to step over to the next well-known part of the data stream.

A.2.3.3.1 Definition of standard component interface

A component, including attributes, which is the general standard component, containing a unique "generic component id", a length indicating count of bytes following as data after the component length and an attribute length indicating the count of bytes in the attribute block (as first part of the component data). The structure is defined by:

<Component(x)>:=	: Component template used for standard components
<IntUnTi>(x),	: id is unique within the scope of the application.
<IntUnLoMB>(compLengthInByte),	: length of the component counted in bytes.
<IntUnLoMB>(attributeBlockLengthInByte);	: length of the attribute block in bytes.

A.2.3.3.2 Example for jumping over unknown content types

— let C1 be a component with an attribute a1 as ShortInt and a sub component C2;

— let C2 be a component with an attribute a2 as one IntUnTi and a second a3 as ShortString;

— let C3 be a component being the successor of C1.

<C1<Component(1)>>:=	
<IntUnTi>(1),	: id = 1.
<IntUnLoMB>(compLengthInByte),	: length of the component counted in bytes
<IntUnLoMB>(attributeBlockLengthInByte);	: length of the attribute block in bytes
<ShortInt>(a1),	: first attribute in C1
<C2>(c2);	: sub component from C1

<C2<Component(2)>>:=	
<IntUnTi>(2),	: id = 2.
<IntUnLoMB>(compLengthInByte),	: length of the component counted in bytes
<IntUnLoMB>(attributeBlockLengthInByte);	: length of the attribute block in bytes
<IntUnTi>(a2),	: first attribute in C2
<ShortString>(a3);	: second attribute in C2

<C3<Component(3)>>:=	
<IntUnTi>(3),	: id = 3.
<IntUnLoMB>(compLengthInByte),	: length of the component counted in bytes
<IntUnLoMB>(attributeBlockLengthInByte);	: length of the attribute block in bytes

For example to demonstrate the method some padding bytes with value CD hex could be added to the stream whereby a decoder could still read C1 – C3. In Figure A.1 one can see a first line with a position number, a second line with the abbreviated function of that byte and a third line with sample content. The arrows under the table show the possible jumps allowing the seeking over the different padding bytes.

Line function abbreviations mean:
CL : component (data) length in bytes
AL : attribute block length in bytes
P : padding bytes
A1, A2, A3 : attributes
C1, C2, C3 : component identifier, begin of the component

Pos	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
Func	C1	CL	AL	A1'	A1"	P	P	C2	CL	AL	A2	A3	A3	A3	A3	A3	P	C3	CL	AL
Val	1	15	4	42	12	CDh	CDh	2	8	7	3	4	'T'	'E'	'S'	'T'	CDh	3	1	0

Figure A.1 — Example for jumping over unknown content with component header information

A.2.4 Toolkits and external definition

Some functionality is shared between different TPEG Applications. This is for example the case for location referencing container and message management container. A TPEG Application therefore can refer to a data type definition not specified in the same Technical Specification.

Toolkits are designed, so that the root components usable as external reference are defined as templates. A TPEG Application using a toolkit template therefore needs to specify a unique generic component id for this instantiation of the interface.

All subsequent components in a toolkit are defined as out of scope of the TPEG Application; i.e. the toolkit on its own defines subcomponents beginning with 0. With that on one hand application decoder must be aware

that component ids of the application may be repeated in sub components of a toolkit. On the other hand further development of application and toolkit can be done independently.

A.2.5 Application design principles

This clause describes design principles that will be helpful in building TPEG applications. A fundamental assumption is that applications will develop and new features will be added. If design principles are adopted properly then older decoders will still operate properly after extending features. Correct design should permit applications to be upgraded and extended over time, providing new features to new decoders, and yet permit existing decoders to continue to operate.

A.2.5.1 Variable data structures

Switches may be included within an application, which permit variations in the subsequent data structure. However, the switch fixes the values of variations. A new type cannot be introduced without breaking backward compatibility. This may be achieved by using components. When new features are likely to be incorporated, attention should be given to the fact that old decoders just 'skip over' new data fields and still expect the old components if they were mandatory.

A.2.5.2 Re-usable and extendable structures

Within an application there will be data structures, which are used repeatedly in a variety of places. There will also certainly be an ever-growing set of structures, as the application protocol develops and incorporates new features. Component templates may be used to minimize the number of occasions within the decoder's software in which the structure needs to be defined, and to permit an increasing variety of structures to be used in a given location.

A.2.5.3 Validity of declarative structures

The Identifier of a component is uniquely defined within each application. The same number may be used in different applications for completely different purposes. Within an application one identifier designates one definition of a component. The design of an application may use components to implement placeholders or to change the composition of elements in a fixed structure.

A.3 TPEG data stream description

A.3.1 Diagrammatic hierarchy representation of frame structure

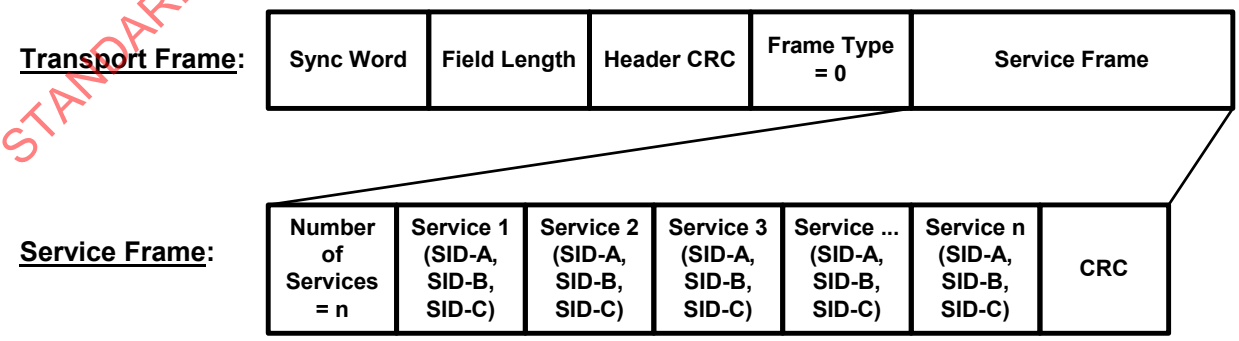

Figure A.2 — TPEG Frame Structure, Frame Type = 0 (i.e. stream directory)

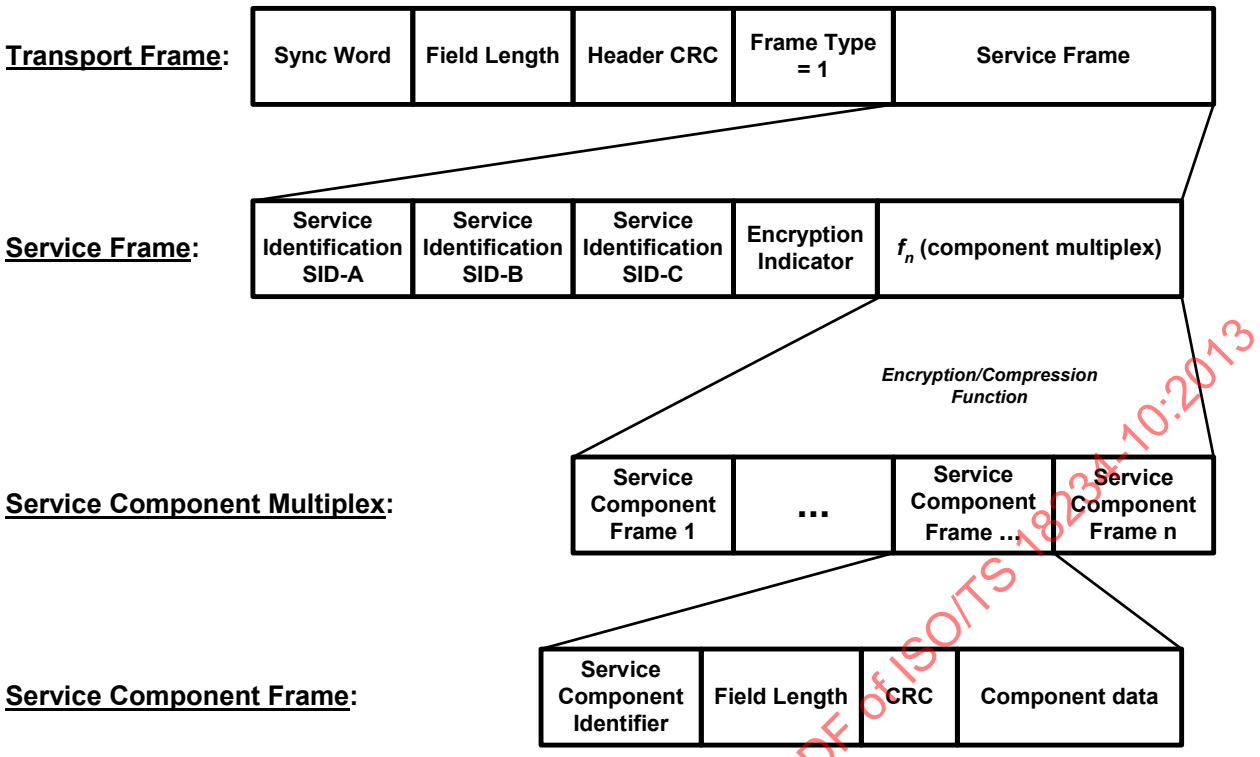

Figure A.3 — TPEG Frame Structure, Frame Type = 1 (i.e. conventional data)

A.3.2 Syntactical Representation of the TPEG Stream

A.3.2.1 TPEG transport frame structure

The following boxes are the syntactical representation of the TPEG frame structure shown in Clause A.3.1 The byte stream contains consecutive transport frames. Each frame includes:

The synchronization word (syncword)	2 bytes	(See Clause A.3.3.1)
The length of the service frame in bytes (field length)	2 bytes	(See Clause A.3.3.2)
The header CRC	2 bytes	(See Clause A.3.3.3)
The frame type indicator	1 byte	(See Clause A.3.3.4)
The service frame	n bytes (n = Field Length)	

The byte stream is built according to the above-mentioned repetitive structure of transport frames. Normally one transport frame should follow another directly, however if any spacing bytes are required these should be set to 0 hex (padding bytes).

```
<TpegStream>:=                          : The data stream.
    infinite {                          : Control element, (loop continues infinitely)
        n * <IntUnTi>(0),               : Any number of padding bytes (0 hex)
        <TransportFrame>                : Transport frames
    };
```

<TransportFrame>:=	
<IntUnLi>(FF0F hex),	: Sync word (FF0F hex)
<IntUnLi>(m),	: Number of bytes in Service Frame
<CRC>(headCRC),	: Header CRC, (See Clause A.3.3.4)
<IntUnTi>(x),	: Frame type of service frame
<ServiceFrame(x)>;	: Any service frame follows

A.3.2.2 TPEG service frame template structure

This service frame comprises:

<ServiceFrame(x)>:=	: Template for service frame
n * **<byte>;**	: Content of service frame

A.3.2.3 Service frame of frame type = 0

The service frame is solely used to transport the stream directory information.

Number of services (n)	1 byte
n *(SID-A, SID-B, SID-C)	n * (3 bytes)
CRC	2 bytes

<StreamDirectory<ServiceFrame(0)>>:=	: Stream directory
<IntUnTi>(n),	: Number of services
n * **<ServiceIdentifier>,**	: Any number of Service IDs
<CRC>;	: CRC of Service IDs

A.3.2.4 Service frame of frame type = 1

Each service frame comprises:

SID-A, SID-B, SID-C	3 bytes	(See Clause A.3.4.2)
The encryption indicator	1 byte	(See Clause A.3.4.1)
The component data	m bytes	

The service level is defined by the service frame. Each transport frame carries one and only one service frame. The service frame includes a component multiplex comprising one or more component frames.

Each service frame may contain a different range and number of component frames as required by the service provider.

Each transport frame may be used by only one service provider and one dedicated service, which supports a mixture of applications. A multiplex of service providers or services is realized by concatenation of multiple transport frames. Each service frame includes service information that comprises the service identification elements and the encryption indicator.

<ConventionalData<ServiceFrame(1)>>:=	: Conventional data
<ServiceIdentifier>,	: Service identification
<IntUnTi>(encIdentifier),	: Encryption indicator n. 0 = no encryption
f_n**(<ServCompMultiplex>);**	: Function f_n (…) is utilized according to the chosen encryption algorithm

A.3.2.5 TPEG service component frame multiplex

The component multiplex is a collection of one or more component frames, the type and order of which are freely determined by the service provider. The resultant multiplex is transformed according to the encryption method required (if the encryption indicator is not 0) or is left unchanged (if the encryption indicator = 0). The length of the resultant data must be less than or equal to 65531 bytes.

<ServCompMultiplex>:=	
n * **<ServCompFrame>**(data)**;**	: Any number of any component frames

A.3.2.6 Interface to application specific frames

The service component frame introduces the application specific code. This means further details of the data stream are specified by the application specification. In the history for different needs slightly different frames have been defined in the existing application specifications. To harmonize this kind of frames, especially for new developments of specifications, this clause specifies not only a basic frame, which is required for any application but also a selection of possible other frames, whereof an application can just choose one without the need to specify its own frame.

An application specification, however, can specify its own frame, which shall at minimum include the following base service component frame as first sub type.

A.3.2.6.1 TPEG base service component frame structure

In a TPEG data stream it shall be possible to have not only one content stream but more; even different from the same application. This is possible with the help of the Service and Network Information (SNI) Application, which is served like variable directory information in the datastream. Therein a table defines a unique number for any content stream being transmitted. This includes also the definition which application is expected in one specific frame. In other words the frame starts not with a typical interface template, but with a header, defining three first values being in common with all service component frames. Therefore, any service component frame is built as shown below:

<ServCompFrame>:=	: Service component frame
<ServCompFrameHeader>(header),	: Common service component header
<ApplicationData>(data);	: Component data

Where the service component header is specified as:

<ServCompFrameHeader>:=	: Common service component frame header
<IntUnTi>(scId),	: Service component identifier (scid is defined by SNI service component designating the application in this service component frame)
<IntUnLi>(lengthInByte),	: Length, n, of component data in bytes
<CRC>(headerCRC);	: Header CRC (See Clause A.3.5.3)

At the component level data is carried in component frames which have a limited length. If applications require greater capacity then the application must be designed to distribute data between component frames and to recombine this information in the decoder.

The inclusion of the field length enables the decoder to skip a component.

The maximum field length of the component data (assuming that there is no transformation, and only one component is included in the service frame) = 65526.

A.3.2.6.2 TPEG specialized service component data schemata

It is in interest of consistency to make sure that service component frames still become defined in as similar as possible in different applications. Specifically with three further attributes being of general nature. The following proposed specialized service component data schemata can be used to inform on this general level about following information:

a) The application data of a component frame with **dataCRC** is error-free.

Data CRC on this level makes it possible, that in case of errors only the service component frame (e.g. one relatively small package of data) would be lost. Other parts of the service multiplex may still be valid and could still be used. (See Clause A.3.5.4)

b) Count of messages the service component frame contains named **messageCount**.

Sometimes it is useful not only to know the opaque count of bytes, but also how many different message have to be expected by the decoder (e.g. for displaying purpose).

c) Prioritization can be made by assigning a **groupPriority**.

In some cases the different service components received shall not just be handled by a FIFO buffer but also with some qualification of priority of messages. In this case high priority message may take precedence over other messages in the decoder. These may be presented to the user even before low priority messages are decoded.

A.3.2.6.2.1 Service component data with dataCRC

Any application should at least specify a data CRC as defined in Clause A.3.5.4 at the end of application data ensuring that bit errors can be detected on service component frame level.

< **ServCompFrameProtected** >:=	: CRC protected service component frame
<**ServCompFrameHeader**>(header),	: Component frame header as defined in A.3.2.6.1
external <**ApplicationContent**>(content),	: Content specified by the individual application
<**CRC**>(dataCRC);	: CRC starting with first byte after the header

A.3.2.6.2.2 Service component data with dataCRC and messageCount

This service frame is used for applications containing messages more or less directly presented to the user which indicate already on frame level how many messages are to be expected. Data CRC is contained as well.

< **ServCompFrameCountedProtected**>:=	: CRC protected service component frame with message count
<**ServCompFrameHeader**>(header),	: Component frame header as defined in A.3.2.6.1
<**IntUnTi**>(messageCount),	: count of messages in this ApplicationContent
external <**ApplicationContent**>(content),	: actual payload of the application
<**CRC**>(dataCRC);	: CRC starting with first byte after the header

A.3.2.6.2.3 Service component data with dataCRC and groupPriority

When messages need to be grouped by priority, this service component frame is used. If not all messages within the frame have the same priority, 'typ007_000: undefined' shall be used. Data CRC is contained as well.

< **ServCompFramePrioritisedProtected**>:=	: CRC protected service component frame with message count
<**ServCompFrameHeader**>(header),	: Component frame header as defined in A.3.2.6.1
<**typ007:Priority**>(groupPriority),	: group priority applicable to all messages in this ApplicationContent
external <**ApplicationContent**>(content),	: actual payload of the application
<**CRC**>(dataCRC);	: CRC starting with first byte of after the header

A.3.2.6.2.4 Service component frame with dataCRC, groupPriority, and messageCount

Additionally, an application can also make use of all features described in previous clauses.

< **ServCompFramePrioritisedCountedProtected**>:=	: CRC protected service component frame with group priority and message count
<**ServCompFrameHeader**>(header),	: Component frame header as defined in A.3.2.6.
<**typ007:Priority**>(groupPriority),	: group priority applicable to all messages in the ApplicationContent
<**IntUnTi**>(messageCount),	: count of messages in this ApplicationContent
external <**ApplicationContent**>(content),	: actual payload of the application
<**CRC**>(dataCRC);	: CRC starting with first byte after the header

A.3.2.6.3 Example of an application implementing a service component frame

An application specification is required to specify first the component frame just as a written sentence. It may for information repeat the definition of the frame, but in this case it shall add a note, that this definition can be superseded by a future release of this specification.

As second definition tree of application starts with:

<**ApplicationContent**>:=	: link provided by SSF
n * <**MyComponent**>(comp);	: n root components of the application

<MyComponent<Component(0)>>:=	
<IntUnTi>(0),	: id = 1
<IntUnLoMB>(compLengthInByte),	: length of the component in bytes
<IntUnLoMB>(attributeBlockLengthInByte),	: length of the attribute block in bytes
<ShortString>(myText),	: some first attribute of the application
<SubComp>(sub);	: some sub components of Component(0)

A.3.3 Description of data on Transport level

A.3.3.1 Syncword

The syncword is 2 bytes long, and has the value of FF0F hex.

The nibbles F hex and 0 hex have been chosen for simplicity of processing in decoders. The patterns 0000 hex and FFFF hex were deprecated to avoid the probability of false triggering in the cases of some commonly used transmission channels.

A.3.3.2 Field length

The field length consists of 2 bytes and represents the number of bytes in the service frame.

This derives from the need of variable length frames.

A.3.3.3 Header CRC

The Header CRC is two bytes long, and is based on the ITU-T polynomial $x^{16} + x^{12} + x^5 + 1$. The Header CRC is calculated on 16 bytes including the syncword, the field length, the frame type and the first 11 bytes of the service frame. In the case that a service frame is shorter than 11 bytes, the sync word, the field length, the frame type and the *whole* service frame shall be taken into account.

In this case the Header CRC calculation does not run into the next transport frame.

The calculation of the CRC is described in ISO/TS 18234-2, Annex C.

A.3.3.4 Frame type

The frame type (FTY) indicates the content of the service frame. Its length is 1 byte. The following table gives the meaning of the frame type:

FTY value (dec):	Content of service frame:	Kind of information in service frame:
0	Number of services, n * (SID-A, SID-B, SID-C)	Stream directory information
1	SID-A, SID-B, SID-C, Encryption ID, Component Multiplex	Conventional service frame data

If FTY = 0, an extra CRC calculation is done over the whole service frame, i.e. starting with n (number of services) and ending with the last SID-C of the last service.

The calculation of the CRC is described in ISO/TS 18234-2, Annex C.

A.3.3.5 Synchronization method

A three-step synchronization algorithm can be implemented to synchronize the receiver:

a) search for an FF0F hex value;

b) calculate and check the header CRC, which follows;

c) check the two bytes, which follow the end of the service frame as defined by the field length.

The two bytes following the end of the service frame should either be a sync word or 00 hex, when spaces are inserted.

A.3.3.6 Error detection

The CRC header provides error detection and protection for the synchronization elements and not for the data within the service frame (except the first 11 bytes, when applicable).

A.3.4 Description of data on Service level

A.3.4.1 Encryption indicator

Length: 1 byte

The encryption indicator is defined as one byte according to TPEG primitive syntax. If the indicator has value 00 hex all data in the component multiplex are non-encrypted. Every other value of the encryption indicator indicates that one of several mechanisms for data encryption or compression has been utilized for all data in the following data multiplex. The encryption/compression technique and algorithms may be freely chosen by the service provider.

0	=	no encryption/compression
1 to 127	=	reserved for standardized methods
128 to 255	=	may be freely used by each service provider, may indicate the use of proprietary methods

A.3.4.2 Service identification

The service IDs are structured in a similar way to Internet IP addresses as follows:

SID-A . SID-B . SID-C

The combination of these three SID elements must be uniquely allocated on a worldwide basis.

The following address allocation system applies:

— SID range for TPEG technical tests SIDs = 000.000.000 - 000.127.255

— SID range for TPEG public tests SIDs = 000.128.000 - 000.255.255

— SID range for TPEG regular public services SIDs = 001.000.000 - 100.255.255

— SID range: reserved for future use SIDs = 101.000.000 - 255.255.255

NOTE The above allocations and structure is significantly changed from that originally specified in ISO/TS 18234-2.

A.3.5 Description of data on Service component level

A.3.5.1 Service component identifier

The service component identifier with the value 0 is reserved for the SNI Application. (See ISO/TS 18234-3).

A.3.5.2 Field length

The field length consists of 2 bytes and represents the number of bytes of the component data.

A.3.5.3 Service component frame header CRC

The component header CRC is two bytes long, and based on the ITU-T polynomial $x^{16}+x^{12}+x^5+1$.

The component header CRC is calculated from the service component identifier, the field length and the first 13 bytes of the component data. In the case of component data shorter than 13 bytes, the component identifier, the field length and all component data shall be taken into account.

The calculation of the CRC is described in ISO/TS 18234-2 Annex C.

A.3.5.4 Service component frame data CRC

The DataCRC is two bytes long, and is based on the ITU polynomial $x^{16}+x^{12}+x^5+1$. This CRC is calculated from all the bytes of the service component frame data after the service component frame header.

The calculation of the CRC is described in ISO/TS 18234-2 Annex C.

A.4 General binary data types

This clause describes the primitive elements and compound elements that are used by TPEG applications.

A.4.1 Primitive data types

The fundamental data element in TPEG technology is the byte, which is represented by 8 bits. All other primitive data types are expressed in terms of bytes as follows:

A.4.1.1 Basic numbers

The following data type represent the general notation of integral numbers either coded as signed or unsigned.

<IntUnTi>:=	: Integer Unsigned Tiny, range 0..255
<byte>;	: Primitive

<IntSiTi>:=	: Integer Signed Tiny, range -128..(+)127
<byte>;	: Two's complement

<IntUnLi>:=	: Integer Unsigned Little, range 0..65 535
<byte>,	: MSB, Most Significant Byte
<byte>;	: LSB, Least Significant Byte

<IntSiLi>:=	: Integer Signed Little, range -32 768..(+)32 767
<byte>,	: MSB, Two's complement
<byte>;	: LSB, Two's complement

<IntUnLo>:=	: Integer Unsigned Long, range 0..4 294 967 295
<byte>,	: MSB
<byte>	
<byte>,	
<byte>;	: LSB

<IntSiLo>:=	: Integer Signed Long, range -2 147 483 648..(+)2 147 483 647
<byte>,	: MSB, Two's complement
<byte>,	
<byte>,	
<byte>;	: LSB, Two's complement

A.4.1.2 MultiByte

A.4.1.2.1 Unsigned Long MultiByte

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, this specification defines the three most significant bits of the fifth, most significant byte as “reserved for future use”, 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.

<IntUnLoMB>:=	: Integer Unsigned Long, range 0..4 294 967 295
m***<byte>**[1..5];	: MS-Bit = 1 signals one more byte follows.

EXAMPLE

Positive number to be encoded

— in Decimal: 1093567633

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

Multibyte encoded:

Byte [5] (MSB)	Byte [4]	Byte [3]	Byte [2]	Byte [1] (LSB)
(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.

A.4.1.2.2 Signed Long MultiByte

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 -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^{32} in five bytes.

For example a value 0x62 (0110 0010) would be encoded with the one byte 0x62. The integer value 0xA7 (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.

<IntSiLoMB>:=	: Integer Signed Long, range -2 147 483 648..(+)2 147 483 647
m***<byte>**[1..5];	: Two’s complement after elimination of continuation flags.
	MS-Bit = 1 signals one more byte follows.

EXAMPLE

Positive number to be encoded

in Decimal: 1093567633

Binary (32bit): 0100 0001001 0111010 0001001 0010001

Multibyte encoded:

Byte [5] (MSB)	Byte [4]	Byte [3]	Byte [2]	Byte [1] (LSB)
(1)"000"0100	(1)0001001	(1)01110100	(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] (MSB)	Byte [4]	Byte [3]	Byte [2]	Byte [1] (LSB)
(1)"111"'1'011	(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.

A.4.1.3 BitArray

This is an encoding specific data type used for encoding an array of Booleans. The bits are encoded in a sequence of bytes, where the first bit of each byte is a continuation flag (shown as c in A.4). If this bit is set (=1) there follows at least one more byte in this bit array. The last byte always has this bit cleared (=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 to last bit. This use, to ensure consistency with other lists, differs from the encoding of numeric values which use a Big-endian bit and byte order.

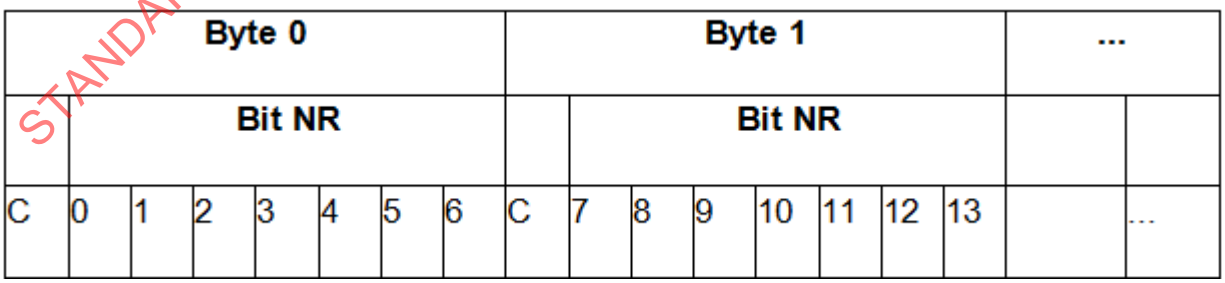

Figure A.4 — BitArray coding format

<BitArray>:=	
m * **<byte>**[1..*];	: byte of flags; MS-Bit = 1 signals one more byte follows

A.4.1.4 Boolean

A single true or false value. The Boolean is differently defined for the following cases:

A.4.1.4.1 Mandatory Boolean

Mandatory Booleans are defined directly in the selector bitarray used for signalling optional attributes. This saved additional bytes for Boolean values. If bit x of selector is set, the Boolean value is interpreted as being true, otherwise false.

A.4.1.4.2 Mandatory Multiple Booleans

For multiple Boolean values (Boolean value with multiplicity higher then 1) the coding requires as first a multibyte "n" as counter how many bit of the then following extra bitarray are in use. The bitarray then contains n valid bits, coding the same as single Booleans. If n = 0 the bitarray attribute is not existing.

A.4.1.5 Date and time information:

The number of seconds elapsed since the start 1970-01-01T00:00:00 Universal Coordinated Time (UTC). This gives values for 136 years until 2106, i. e. 2^{32} seconds from the year 1970.

The exact formula for the date and time calculation can be found in ISO/TS 18234-2 Annex D.

<DateTime>:=	: Date and time
<IntUnLo>;	: Number of seconds since 1970-01-01T00:00:00 Universal Coordinated Time (UTC)

A.4.1.5.1 Day selection

This type gives the possibility to select one or more days of the week to indicate the repetition of an event.

A DaySelector attribute can be used to select one or more week days. The Boolean attributes indicate whether a particular day is included in the selection, if the attribute value is "true", the day is selected. These seven attributes are mandatory Booleans, encoded using a BitArray.

<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

EXAMPLE 1 **<DaySelector>** = 05 hex - Meaning: The event (e. g. service) is repeated every Sunday and Tuesday.

EXAMPLE 2 **<DaySelector>** = 7E hex - Meaning: The event (e. g. service) is repeated every day except Sunday.

A.4.1.5.2 Duration

Values of this type define temporal duration, expressed in a number of whole seconds. Values must be between 0 and 4294967295. Because it is expected that in many cases the amount of the value is low, variable length coding is used.

<Duration>:=	: Time duration
<IntUnLoMB>;	: Number of seconds

A.4.1.6 DistanceMetres

Distance in integer units of metres.

<DistanceMetres>:=	
<IntUnLoMB>;	: Distance in integer units of metres.

A.4.1.7 DistanceCentiMetres

Distance in integer units of centimetres.

<DistanceCentiMetres>:=	
<IntUnLoMB>;	: Distance in integer units of centimetres.

A.4.1.8 CRC-word data type

The Cyclic redundancy check (CRC) is a calculated hash value over a defined array of bytes. The definition of a CRC must include the definition of the array.

<CRC>:=	: Cyclic redundancy check
<IntUnLi>;	: According to ITU-T polynomial, over an indicated Range of elements. (See ISO/TS 18234-2 Annex C)

A.4.1.9 FixedPercentage

FixedPercentage defines a fixed percentage value in integer units in the range 0 and 100.

A fixed percentage can *not* be used as an indication of a change, where both negative values and values larger than a 100% might be required.

<FixedPercentage>:=	
<IntUnTi>;	: valid values of percentage from 0 to 100

A.4.1.10 Probability

Probability defines a percentage value between zero and one with a precision of two decimals. Where zero denotes no probability and one hundred certainty.

<Probability>:=	
<FixedPercentage>;	: valid values from 0 to 100

A.4.1.11 Float

A float defines a number with decimal precision. It is encoded as an IEC 60559 single precision floating point number (32 bit).

<Float>:=	
<IntUnLo>;	: IEC 60559 single precision floating point number

A.4.1.12 Severity

Severity is application specific and defined in the range from 1 to 255 where higher values are expected to be more severe. Value 0 is predefined as undefined.

<Severity>:=	
<IntUnTi>;	: Application specific value of severity

A.4.1.13 Velocity

Velocity in integer units of metres per second in the range from 0 to 255.

<Velocity>:=	
<IntUnTi>;	: Speed in m/s

A.4.1.14 Weight

Weight in integer units of kilogram's. The value is in the range from 0 to 4294967295, encoded as IntUnLoMB.

<Weight>:=	
<IntUnLoMB>;	: Weight in kg

A.4.2 Compound data types

A.4.2.1 ServiceIdentifier

A service identifier is an data type that defines a single service identifier.

<ServiceIdentifier>:=	
<IntUnTi>(sidA),	: Service identification part A
<IntUnTi>(sidB),	: Service identification part B
<IntUnTi>(sidC);	: Service identification part C

A.4.2.2 FixedPointNumber

Defines a value from -2147483648.99 to 2147483647.99 with a fixed precision of 2 decimals.

<FixedPointNumber>:=	
<IntSiLoMB>(integralPart),	: integral part of the number
<IntUnTi>(decimalPart);	: fraction of 2 decimal digits values from 0 to 99

A.4.2.3 Strings

The string of characters is represented by a series of n bytes. These bytes need to be interpreted according to a character table, which will designate the byte width of each character. The encoding of the characters is defined in ISO/TS 18234-3. Where multiple code tables are used, an application needs mechanisms to set the scope of applicability of each table.

<ShortString>:=	: Short string
<IntUnTi>(n),	: Number of bytes, n
n * **<byte>;**	: String of characters; count of characters depends on chosen coding

<LongString>:=	: Long string
<IntUnLi>(n),	: Number of bytes, n
n * **<byte>;**	: String of characters; count of characters depends on chosen coding

A.4.2.4 Localised Strings

A string accompanied with a language code that identifies the language that the string is given in. The typ001:LanguageCode is derived from ISO 639: 2002 - Codes for the representation of names of languages.

<LocalisedShortString>:=	
<typ001:LanguageCode>(languageCode),	: Specifies the language used for this string.
<ShortString>(string);	: Short string

<LocalisedLongString>:=	
<typ001:LanguageCode>(languageCode),	: Specifies the language used for this string.
<LongString>(string);	: Long string

A.4.2.5 Compound time information

A.4.2.5.1 TimeInterval

The TimeInterval data structure can be used when an interval in time must be specified with more flexibility than the simple Duration type allows.

Each TimeInterval attribute has a number of optional attributes. It is maximally 101 years long. Each attribute can be used as stand-alone attribute or in combination with other attributes. When an attribute is not given, the value zero is implied. Every TimeInterval must specify at least one attribute.

<TimeInterval>:=	
<BitArray>(selector),	: DaySelector
if (bit 0 of selector is set)	
<IntUnTi>(years),	: Number of years in the range from 0 to 100.
if (bit 1 of selector is set)	
<IntUnTi>(months),	: Number of months in the range from 0 to 12.
if (bit 2 of selector is set)	
<IntUnTi>(days),	: Number of days in the range from 0 to 31.
if (bit 3 of selector is set)	
<IntUnTi>(hours),	: Number of hours in the range from 0 to 24.
if (bit 4 of selector is set)	
<IntUnTi>(minutes),	: Number of minutes in the range from 0 to 60.
if (bit 5 of selector is set)	
<IntUnTi>(seconds);	: Number of seconds in the range from 0 to 60.

A.4.2.5.2 TimePoint

The TimePoint data structure can be used when a point in time must be specified with fewer granularities than the simple DateTime allows. Each TimePoint attribute has a number of optional attributes. Each attribute can be used as stand-alone attribute or in combination with other attributes. Every TimePoint must specify at least one attribute. In binary encoding, 1970 is subtracted from the year, mapping the range 1970-2100 to the values 0-130.

<TimePoint>:=	
<BitArray>(selector),	: DaySelector
if (bit 0 of selector is set)	
<IntUnTi>(year),	: The year in the range from 1970 to 2100.
if (bit 1 of selector is set)	
<IntUnTi>(month),	: Number of months in the range from 1 to 12.
if (bit 2 of selector is set)	
<IntUnTi>(day),	: Number of days in the range from 1 to 31.
if (bit 3 of selector is set)	
<IntUnTi>(hour),	: Number of hours in the range from 0 to 23.
if (bit 4 of selector is set)	
<IntUnTi>(minute),	: Number of minutes in the range from 0 to 59.
if (bit 5 of selector is set)	
<IntUnTi>(second);	: Number of seconds in the range from 0 to 59.

A.4.2.5.3 TimeToolkit

The time toolkit allows different date and time information to be described. For example where an event has a start but no known end-time. In such a case we should used only a start point but omit an end-time. Each TimeToolkit attribute has a number of optional attributes. Each attribute can be used as a stand-alone attribute or in combination with other attributes. Every TimeToolkit must specify at least one attribute. For a timestamp the DateTime type should be used.

<TimeToolkit>:=	
<BitArray>(selector),	: 1 byte containing 5 switches
if (bit 0 of *selector* is set)	
<TimePoint>(startTime),	: An event time point (e.g. flight departure) or an event starting time (e.g. open from)
if (bit 1 of *selector* is set)	
<TimePoint>(stopTime),	: An event stopping time (e.g. open to). The stop time can be used only together with a start time
if (bit 2 of *selector* is set)	
<TimeInterval>(duration),	: A time interval (e.g. free parking limit)
if (bit 3 of *selector* is set)	
<typ002:SpecialDay>(specialDay),	: Relevant days of a certain type (e.g. weekdays or holiday)
if (bit 4 of *selector* is set)	
<DaySelector>(daySelector);	: Gives the option to specify days of the week

A.4.3 Table definitions

A.4.3.1 Table entry

In TPEG much information is based on tables. These tables represent clearly defined groupings of pre-defined concepts. The idea is to inform the device about the concept and let the device choose the best possible presentation of this concept in the context of the other parts of the TPEG message. This approach means that devices can present concepts e.g. in any language or even as graphical icons. This Table data type only serves as a basis for all tables used in the toolkits and applications.

A table can have up to 256 entries.

<Table>:=	
<IntUnTi>(entry)**;**	: The corresponding table defines valid entries of a table

A.4.3.2 Tables of general use

Some tables are of general use in different TPEG Applications, therefore this clause describes the content of those tables.

A.4.3.3 Typ001:LanguageCode

ISO 639: 2002 - Codes for the representation of names of languages. See Clause A.4.4.1 for values.

<Typ001:LanguageCode>:=	
<Table>;	: Specifies the language

A.4.3.4 Typ002:SpecialDay

The SpecialDay table lists special types of days, such as “public holiday” or “weekdays” and similar. See Clause A.4.4.2 for values.

<Typ002:SpecialDay>:=	
<Table>;	: Identifies the special day

A.4.3.5 Typ003:CurrencyType

CurrencyType, based on the three-alpha codes of ISO 4217. See Clause A.4.4.3 for values.

<Typ003:CurrencyType>:=	
<Table>;	: Three-alpha codes of ISO 4217

A.4.3.6 Typ004:NumericalMagnitude

At a number of places within TPEG’s applications there is a need to use a number to describe a quantity of people, animals, objects, etc. The range of the number needs to be at least from 0 to a few million. At the bottom end of this range, numbers need to be in unit intervals, up to 50. Above 50, tens may be used up to 500, then hundreds up to 5000. This same principle is required for each decade. The table contains unsigned integer values in the range 0 to 3000000 with decreasing precision. See Clause A.4.4.4 for the translated values. For a formal mathematical definition of numerical magnitude values, refer to Annex B of ISO/TS 18234-2.

<Typ004:NumericalMagnitude>:=	
<Table>(n)**;**	: Numerical magnitude

A.4.3.7 Typ005:CountryCode

This table lists countries as defined by ISO 3166-1. See Clause A.4.4.5 for values.

NOTE "undecodable country" is to be used by a client device unable to read the typ005 code used by a service provider - no code value for this word is ever transmitted.

<Typ005:CountryCode>:=
<Table>; : Countries as defined by ISO 3166-1

A.4.3.8 Typ006:OrientationType

This is the table of values of the compass orientation like “north-west”. See Clause A.4.4.6 for values.

<Typ006:OrientationType>:=
<Table>; : Denotes a compass orientation

A.4.3.9 Typ007:Priority

This is the table of values of the priority of messages. See Clause a.4.4.7 for values.

<Typ007:Priority>:=
<Table>; : Denotes priority of a message

A.4.4 Tables

A.4.4.1 typ001:LanguageCode

ISO 639:2002 - Codes for the representation of names of languages. The Comment column lists the 2-alpha codes of ISO 639-1.

Code	Reference-English Language Name	Comment 2-alpha code
000	Unknown	
001	Afar	aa
002	Abkhazian	ab
003	Avestan	ae
004	Afrikaans	af
005	Akan	ak
006	Amharic	am
007	Aragonese	an
008	Arabic	ar
009	Assamese	as
010	Avaric	av
011	Aymara	ay
012	Azerbaijani	az
013	Bashkir	ba
014	Belarusian	be
015	Bulgarian	bg
016	Bihari	bh
017	Bislama	bi
018	Bambara	bm
019	Bengali	bn
020	Tibetan	bo

021	Breton	br
022	Bosnian	bs
023	Catalan	ca
024	Chechen	ce
025	Chamorro	ch
026	Corsican	co
027	Cree	cr
028	Czech	cs
029	Church Slavic	cu
030	Chuvash	cv
031	Welsh	cy
032	Danish	da
033	German	de
034	Divehi	dv
035	Dzongkha	dz
036	Ewe	ee
037	Greek	el
038	English	en
039	Esperanto	eo
040	Spanish	es
041	Estonian	et
042	Basque	eu
043	Persian	fa
044	Fulah	ff
045	Finnish	fi
046	Fijian	fj
047	Faroese	fo
048	French	fr
049	Western Frisian	fy
050	Irish	ga
051	Scottish Gaelic	gd
052	Galician	gl
053	Guaraní	gn
054	Gujarati	gu
055	Manx	gv
056	Hausa	ha
057	Hebrew	he
058	Hindi	hi
059	Hiri Motu	ho
060	Croatian	hr
061	Haitian	ht
062	Hungarian	hu
063	Armenian	hy
064	Herero	hz
065	Interlingua (International Auxiliary Language Association)	ia
066	Indonesian	id
067	Interlingue	ie
068	Igbo	ig
069	Sichuan Yi	ii
070	Inupiaq	ik

071	Ido	io
072	Icelandic	is
073	Italian	it
074	Inuktitut	iu
075	Japanese	ja
076	Javanese	jv
077	Georgian	ka
078	Kongo	kg
079	Kikuyu	ki
080	Kuanyama	kj
081	Kazakh	kk
082	Kalaallisut	kl
083	Khmer	km
084	Kannada	kn
085	Korean	ko
086	Kanuri	kr
087	Kashmiri	ks
088	Kurdish	ku
089	Komi	kv
090	Cornish	kw
091	Kirghiz	ky
092	Latin	la
093	Luxembourgish	lb
094	Ganda	lg
095	Limburgish	li
096	Lingala	ln
097	Lao	lo
098	Lithuanian	lt
099	Luba-Katanga	lu
100	Latvian	lv
101	Malagasy	mg
102	Marshallese	mh
103	Ma-ori	mi
104	Macedonian	mk /sl
105	Malayalam	ml
106	Mongolian	mn
107	Moldavian	mo
108	Marathi	mr
109	Malay	ms
110	Maltese	mt
111	Burmese	my
112	Nauru	na
113	Norwegian Bokmål	nb
114	North Ndebele	nd
115	Nepali	ne
116	Ndonga	ng
117	Dutch	nl
118	Norwegian Nynorsk	nn
119	Norwegian	no
120	South Ndebele	nr

121	Navajo	nv
122	Chichewa	ny
123	Occitan	oc
124	Ojibwa	oj
125	Oromo	om
126	Oriya	or
127	Ossetian	os
128	Panjabi	pa
129	Pa-li	pi
130	Polish	pl
131	Pashto	ps
132	Portuguese	pt
133	Quechua	qu
134	Raeto-Romance	rm
135	Kirundi	rn
136	Romanian	ro
137	Russian	ru
138	Kinyarwanda	rw
139	Sanskrit	sa
140	Sardinian	sc
141	Sindhi	sd
142	Northern Sami	se
143	Sango	sg
144	Serbo-Croatian	sh
145	Sinhalese	si
146	Slovak	sk
147	Slovenian	sl
148	Samoan	sm
149	Shona	sn
150	Somali	so
151	Albanian	sq
152	Serbian	sr
153	Swati	ss
154	Southern Sotho	st
155	Sundanese	su
156	Swedish	sv
157	Swahili	sw
158	Tamil	ta
159	Telugu	te
160	Tajik	tg
161	Thai	th
162	Tigrinya	ti
163	Turkmen	tk
164	Tagalog	tl
165	Tswana	tn
166	Tonga	to
167	Turkish	tr
168	Tsonga	ts
169	Tatar	tt
170	Twi	tw

171	Tahitian	ty
172	Uighur	ug
173	Ukrainian	uk
174	Urdu	ur
175	Uzbek	uz
176	Venda	ve
177	Vietnamese	vi
178	Volapük	vo
179	Walloon	wa
180	Wolof	wo
181	Xhosa	xh
182	Yiddish	yi
183	Yoruba	yo
184	Zhuang	za
185	Chinese	zh
186	Zulu	zu

A.4.4.2 typ002:SpecialDay

The SpecialDay table lists special types of days, such as public holiday and similar.

Code	Reference-English 'word'	Comment	Example
000	unknown		
001	weekdays	Monday to Friday	
002	weekends	Saturday and Sunday	
003	holiday		
004	public holiday		
005	religious holiday		e.g. Christmas Day
006	federal holiday		
007	regional holiday		
008	national holiday		e.g. In UK: Mayday
009	school days		
010	every day		

A.4.4.3 typ003:CurrencyType

CurrencyType, based on the three-alpha codes of ISO 4217.

Code	Reference-English 'word'	Comment
000	unknown	
001	AED	
002	AFA	
003	ALL	
004	AMD	
005	ANG	
006	AOA	
007	ARS	
008	AUD	
009	AWG	
010	AZM	

011	BAM	
012	BBD	
013	BDT	
014	BGN	
015	BHD	
016	BIF	
017	BMD	
018	BND	
019	BOB	
020	BRL	
021	BSD	
022	BTN	
023	BWP	
024	BYR	
025	BZD	
026	CAD	
027	CDF	
028	CHF	
029	CLP	
030	CNY	
031	COP	
032	CRC	
033	CSD	
034	CUP	
035	CVE	
036	CYP	
037	CZK	
038	DJF	
039	DKK	
040	DOP	
041	DZD	
042	EEK	
043	EGP	
044	ERN	
045	ETB	
046	EUR	
047	FJD	
048	FKP	
049	GBP	
050	GEL	
051	GGP	
052	GHC	
053	GIP	
054	GMD	
055	GNF	
056	GTQ	
057	GYD	
058	HKD	
059	HNL	
060	HRK	

061	HTG	
062	HUF	
063	IDR	
064	ILS	
065	IMP	
066	INR	
067	IQD	
068	IRR	
069	ISK	
070	JEP	
071	JMD	
072	JOD	
073	JPY	
074	KES	
075	KGS	
076	KHR	
077	KMF	
078	KPW	
079	KRW	
080	KWD	
081	KYD	
082	KZT	
083	LAK	
084	LBP	
085	LKR	
086	LRD	
087	LSL	
088	LTL	
089	LVL	
090	LYD	
091	MAD	
092	MDL	
093	MGA	
094	MKD	
095	MMK	
096	MNT	
097	MOP	
098	MRO	
099	MTL	
100	MUR	
101	MVR	
102	MWK	
103	MXN	
104	MYR	
105	MZM	
106	NAD	
107	NGN	
108	NIO	
109	NOK	
110	NPR	