TECHNICAL SPECIFICATION

ISO/TS 18234-2

First edition
2006-06-01

Traffic and Travel Information (TTI) — TTI via Transport Protocol Expert Group (TPEG) data-streams —

Part 2: Syntax, Semantics and Framing Structure (SSF)

Informations sur le trafic et le tourisme (TTI) — Messages TTI via les flux de données du groupe d'experts du protocole de transport (TPEG) —

Partie 2: Structure de syntaxe, de sémantique et de cadrage (SSF)

Reference number
ISO/TS 18234-2:2006(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 ... iv

Introduction ... v

1 **Scope** ... 1

2 **Normative references** ... 1

3 **Symbols and abbreviations** ... 2

4 **Design principles** ... 3
4.1 **TPEG transmission** ... 4
4.2 **TPEG layer model** ... 5

5 **Conventions and symbols** ... 7
5.1 **Conventions** ... 7
5.2 **Symbols** ... 7

6 **Representation of syntax** ... 8
6.1 **General** ... 8
6.2 **Data type notation** ... 8
6.3 **Application independent elements** ... 9
6.4 **Application dependent elements** ... 13
6.5 **Application design principles** ... 14

7 **TPEG description** ... 15
7.1 **Hierarchy (frame structure)** ... 15
7.2 **Transport level** ... 18
7.3 **Service level** ... 19
7.4 **Service component multiplex level** ... 21
7.5 **Service component level** ... 21

Annex A (normative) **Character tables** ... 22

Annex B (normative) **Method for coding quantities of objects** ... 23

Annex C (normative) **CRC calculation** ... 25

Annex D (normative) **Time calculation** ... 27

Annex E (informative) **A description of the TPEG byte-stream using C-type notation** ... 30

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 18234-2 was prepared by Technical Committee ISO/TC 204, *Intelligent transport systems*.

ISO/TS 18234 consists of the following parts, under the general title *Traffic and Travel Information (TTI) — TTI via Transport Protocol Expert Group (TPEG) data-streams:*

— *Part 1: Introduction, numbering and versions*

— *Part 2: Syntax, Semantics and Framing Structure (SSF)*

— *Part 3: Service and Network Information (SNI) application*

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

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

— *Part 6: Location referencing applications*

Introduction

TPEG technology uses a byte-oriented 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 are used to transfer application data from the database of a service provider to a user's equipment.

This document describes the Service and Network Information Application, which provides a means of informing end-users about all possible services and their content which are considered relevant by a service provider to either provide continuity of his services or inform the end-user about other related services. As stated in the design criteria, TPEG is a bearer independent system. Therefore some rules are established for the relation of information contents of the same service on different bearers. Also the mechanisms for following a certain service on one single bearer have to be defined. For the receiver it is essential to find an adjacent or similar service if it leaves the current reception area. Nonetheless, basic information describing the service itself is necessary. For the ease of the user, e.g. the service name, the service provider name, the operating time and many other hints are delivered by the TPEG-SNI application.

Also, general models for the hand-over and the referencing of services are developed and shown in detail. It is important to note that this Part 2 of CEN ISO/TS 18234 (TPEG-SSF) is closely related to Part 3 (TPEG-SNI) and so they must be used together, being dependent upon each other.

The Broadcast Management Committee of the European Broadcast Union (EBU) established 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. The 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 public specifications. Two documents were released. Part 2 (TPEG-SSF, CEN ISO/TS 18234-2, this document) described the Syntax, Semantics and Framing structure, which will be used for all TPEG applications. Part 4 (TPEG-RTM, CEN ISO/TS 18234-4) described the *first* application, for Road Traffic Messages.

CEN/TC 278/WG 4, in conjunction with ISO/TC 204/WG 10, 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 have been developed to make the initial complete set of four parts, enabling the implementation of a consistent service. Part 3 (TPEG-SNI, CEN ISO/TS 18234-3) describes the Service and Network Information Application, which is likely to be used by all service implementations to ensure appropriate referencing from one service source to another. Part 1 (TPEG-INV, CEN ISO/TS 18234-1) completes the work, by describing the other parts and their relationships; it also contains the application IDs used within the other parts.

In April 2000, the B/TPEG group released revised Parts 1 to 4, all four parts having been reviewed and updated in the light of initial implementation results. Thus a consistent suite of specifications, ready for wide scale implementation, was submitted to the CEN/ISO commenting process.

In November 2001, after extensive response to the comments received and from many internally suggested improvements, all four parts were completed for the next stage: the Parallel Formal Vote in CEN and ISO. But a major step forward has been to develop the so-called TPEG-Loc location referencing method, which enables both map-based TPEG-decoders and non map-based ones to deliver either map-based location referencing or human readable information. Part 6 (TPEG-Loc, CEN ISO/TS 18234-6) is now a separate specification and is used in association with the other parts of CEN ISO/TS 18234 to provide comprehensive location referencing. Additionally Part 5, the Public Transport Information Application (TPEG-PTI, CEN ISO/TS 18234-5), has been developed and been through the commenting process.

This Technical Specification, CEN ISO/TS 18234-2, provides a full specification to the primitives used, framing, time calculation, numbers and to specific rules such as CRC calculation.

During the development of the TPEG technology a number of versions have been documented and various trials implemented using various versions of the specifications. At the time of the publication of this Technical Specification, all parts are fully inter-workable and no specific dependencies exist. This Technical Specification has the technical version number TPEG-SSF_3.0/002.

Traffic and Travel Information (TTI) — TTI via Transport Protocol Expert Group (TPEG) data-streams —

Part 2: Syntax, Semantics and Framing Structure (SSF)

1 Scope

This Technical Specification establishes the method of referencing used within a TPEG data-stream to allow a service provider to signal availability of the same service on another bearer channel or similar service data from another service.

TPEG is a byte-oriented 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 are used to transfer application data from the database of a service provider to a user's equipment.

The protocol is structured in a layered manner and employs a general purpose framing system which is adaptable and extensible, and which carries frames of variable length. This has been designed with the capability of explicit frame length identification at nearly all levels, giving greater flexibility and integrity, and permitting the modification of the protocol and the addition of new features without disturbing the operation of earlier client decoder models.

2 Normative references

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

ISO/IEC 7498-1, *Information technology — Open Systems Interconnection — Basic Reference Model: The Basic Model*

ISO/IEC 8859-1, *Information technology — 8-bit single byte coded graphic character sets — Part 1: Latin alphabet No. 1*

ISO/IEC 8859-2, *Information technology — 8-bit single byte coded graphic character sets — Part 2: Latin alphabet No. 2*

ISO/IEC 8859-3, *Information technology — 8-bit single byte coded graphic character sets — Part 3: Latin aAlphabet No. 3*

ISO/IEC 8859-4, *Information technology — 8-bit single byte coded graphic character sets — Part 4: Latin alphabet No. 4*

ISO/IEC 8859-5, *Information technology — 8-bit single byte coded graphic character sets — Part 5: Latin/Cyrillic alphabet*

ISO/IEC 8859-6, *Information technology — 8-bit single byte coded graphic character sets — Part 6: Latin/Arabic alphabet*

ISO/IEC 8859-7, *Information technology — 8-bit single byte coded graphic character sets — Part 7: Latin/Greek alphabet*

ISO/IEC 8859-8, *Information technology —8-bit single byte coded graphic character sets — Part 8: Latin/Hebrew alphabet*

ISO/IEC 8859-9, *Information technology — 8-bit single byte coded graphic character sets — Part 9: Latin alphabet No. 5*

ISO/IEC 8859-10, *Information technology — 8-bit single byte coded graphic character sets — Part 10: Latin alphabet No. 6*

ISO/IEC 8859-13, *Information technology — 8-bit single byte coded graphic character sets — Part 13: Latin alphabet No. 7*

ISO/IEC 8859-14, *Information technology — 8-bit single byte coded graphic character sets — Part 14: Latin alphabet No. 8 (Celtic)*

ISO/IEC 8859-15, *Information technology — 8-bit single byte coded graphic character sets — Part 15: Latin alphabet No. 9*

ISO/IEC 10646, *Information technology — Universal Multiple-Octet Coded Character Set (UCS)*

3 Symbols and abbreviations

For the purposes of this Technical Specification, the following abbreviations apply:

3.1
AID
Application Identification

3.2
BPN
Broadcast, Production and Networks (an EBU document publishing number system)

3.3
B/TPEG
Broadcast/TPEG (the EBU project group name for the specification drafting group)

3.4
CEN
Comité Européen de Normalisation

3.5
DAB
Digital Audio Broadcasting

3.6
DARC
Data Radio Channel - an FM sub-carrier system for data transmission

3.7
DVB
Digital Video Broadcasting

3.8
EBU
European Broadcasting Union

3.9
INV
Introduction, Numbering and Versions (see CEN ISO/TS 18234-1)

3.10
IPR
Intellectual Property Right(s)

3.11
ISO
International Organization for Standardization

3.12
ITU-T
International Telecommunication Union - Telecom

3.13
OSI
Open Systems Interconnection

3.14
RTM
Road Traffic Message application (see CEN ISO/TS 18234-4)

3.15
SNI
Service and Network Information application (see CEN ISO/TS 18234-3)

3.16
SSF
Syntax, Symantics and Framing Structure (this specification)

3.17
TPEG
Transport Protocol Experts Group

3.18
TTI
Traffic and Travel Information

3.19
UAV
unassigned value

3.20
UTC
Coordinated Universal Time

4 Design principles

The following principles have been assumed in the development of the TPEG protocol, structure and semantics:

— TPEG is unidirectional

— TPEG is byte-oriented, where a byte is represented by eight bits

— TPEG provides a protocol structure, which employs asynchronous framing

— TPEG includes a CRC error detection capability applicable on a variety of different levels

— TPEG assumes the use of a transparent data channel

— TPEG assumes that underlying systems will have an appropriate level of reliability

— TPEG assumes that underlying systems may employ error correction

— TPEG has a hierarchical data frame structure

— TPEG is used to transport information from database to database

— TPEG provides service provider name, service name and network information

— TPEG permits the use of encryption mechanisms, if required by an application

4.1 TPEG transmission

TPEG is intended to operate via almost any simple digital data channel, and it assumes nothing of the channel other than the ability to convey a stream of bytes. To this end, the concept of transmission via a “piece of wire” is envisaged, in which the bearer has no additional service features.

In Figure 1, a variety of possible transmission channels are shown. The only requirement of the channel is that a sequence of bytes may be carried between the TPEG generator and the TPEG decoder. This requirement is described as “transparency”. However it is recognized that data channels may introduce errors. Bytes may be omitted from a sequence, bytes may become corrupted or additional and erroneous data could be received. Therefore TPEG incorporates error detection features at appropriate points and levels. It is assumed that bearer systems will introduce an appropriate level of error correction.

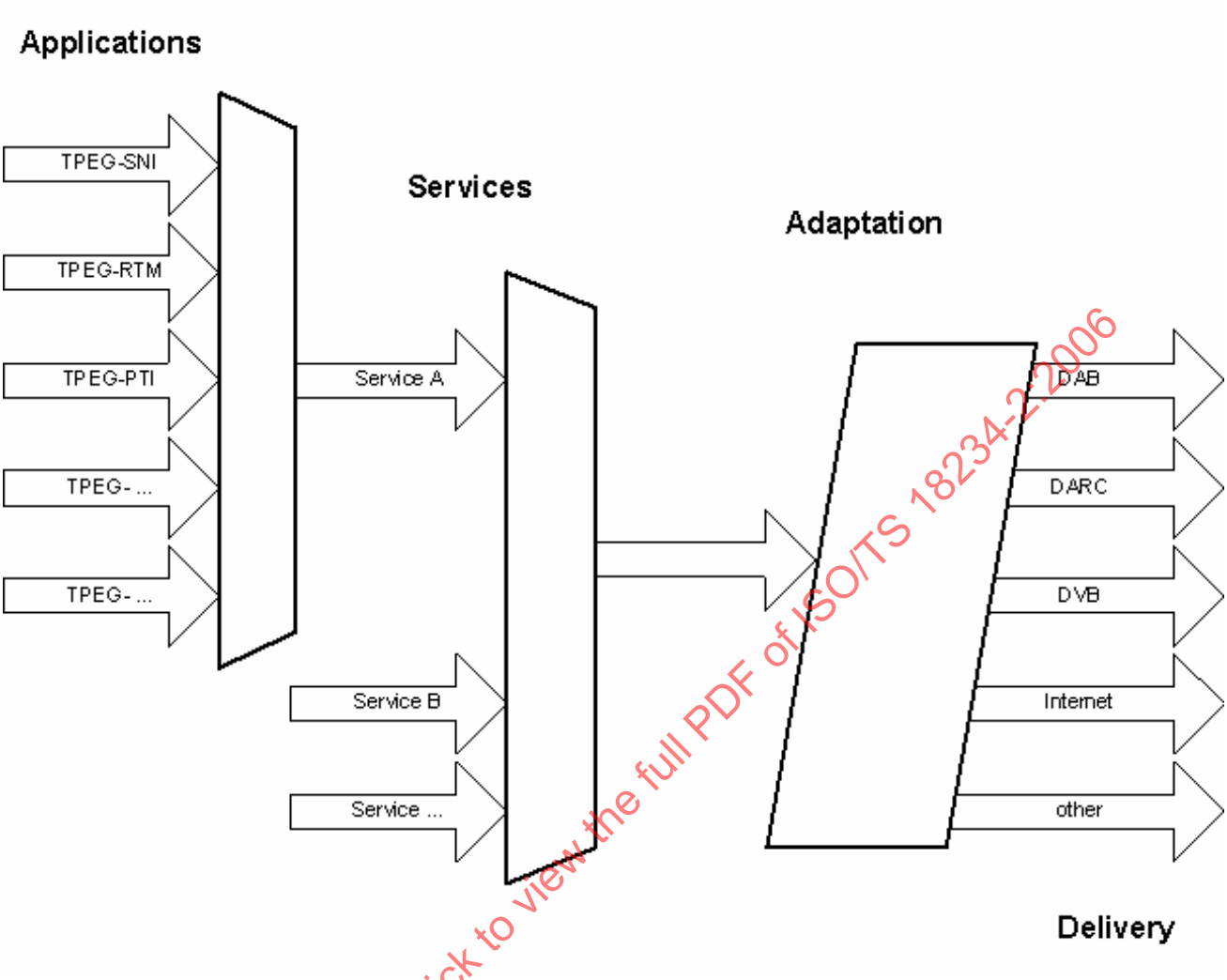

Figure 1 — TPEG data may be delivered simultaneously via different bearers

4.2 TPEG layer model

In Figure 2, the different layers of the TPEG protocol are identified in accordance with the ISO/OSI model (ISO/IEC 7498-1).

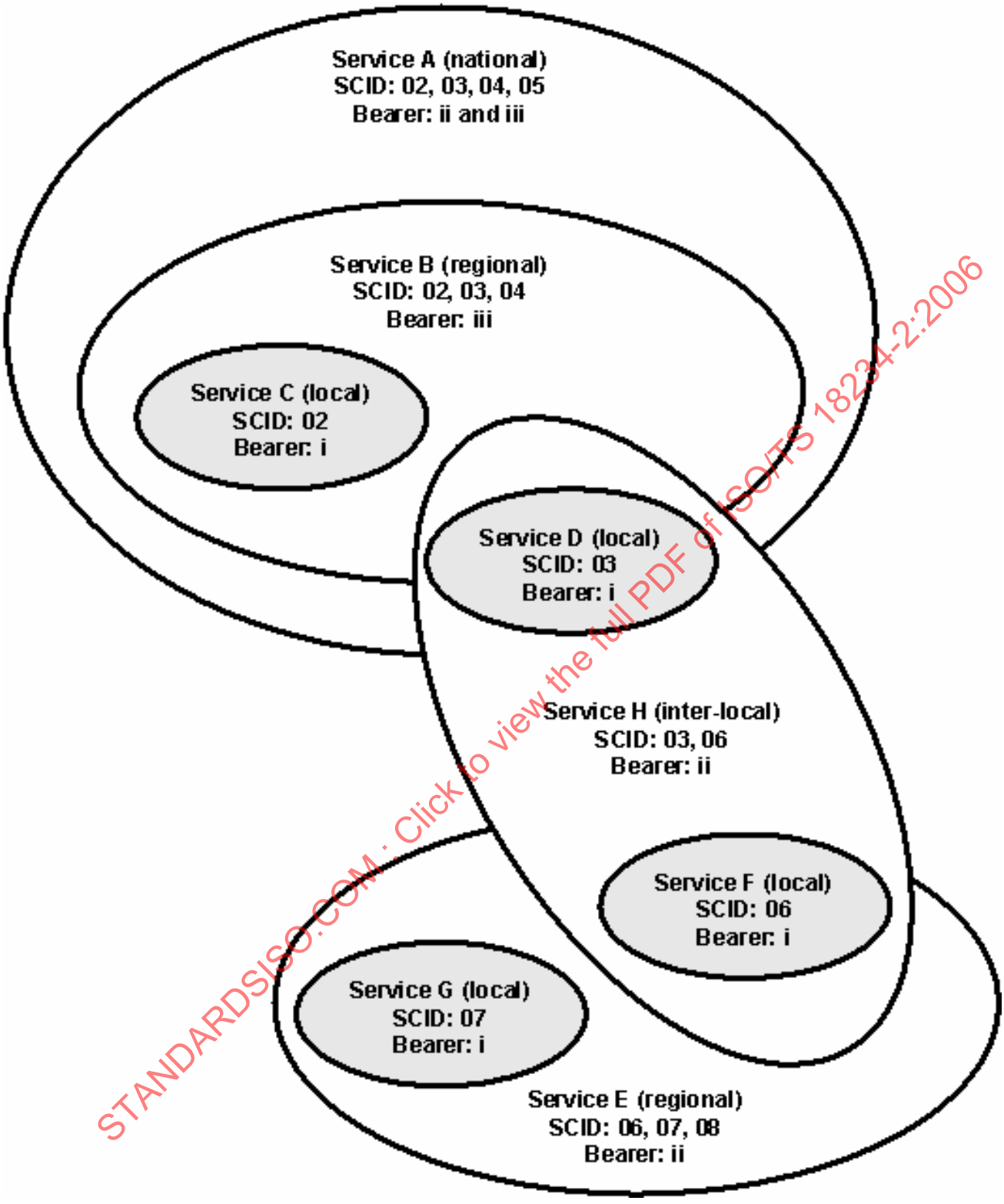

Figure 2 — TPEG in relation to the ISO/OSI Layer Model

Layer 7 is the top level and referred to in TPEG as the application layer. Initially the following applications were defined:

— TPEG specifications - Part 3: Service and Network Information Application (Service provider name, logo, hand-over information, etc.) (CEN ISO/TS 18234-3)

— TPEG specifications - Part 4: Road Traffic Message application (Event description, location description, etc.) (CEN ISO/TS 18234-4)

Layer 4 is the packetization layer. Components are merged into a single stream and encrypted and/or compressed.

Layer 3 is the network layer. This layer defines the means for synchronization and routing. This is the lowest layer of the TPEG protocol.

Layer 2 is the datalink layer. This layer consists of a wide range of different bearers, which are suitable carriers for the TPEG protocol. An adaptation layer may be required in order to map the TPEG stream onto that bearer.

Layer 1 is the physical layer. This defines the transmission medium (radio waves, wire, optical, etc.). One particular bearer can make use of different physical layers.

5 Conventions and symbols

5.1 Conventions

5.1.1 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.

5.1.2 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 document, 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 Annex E.

5.1.3 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 specification 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.

5.2 Symbols

5.2.1 Literal numbers

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

123 = 123 decimal

123 hex = 123 hexadecimal

5.2.2 Variable numbers

Symbols are used to represent numbers whose values are not predefined within the TPEG specifications. 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.

5.2.3 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.

6 Representation of syntax

6.1 General

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

6.2 Data type notation

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:

<new_data_type>:=	: Description of data type
<data_type_a>(optional symbol);	: Component description
<data_type_b>(optional symbol);	: Component description

This shows an example data structure, which has just two components, one of type **<data_type_a>** and the other of **<data_type_b>**. A symbol 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 component 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 component structure is repeated a number of times, this may be shown as follows:

<new_data_type>:=	: Description of data type
<data_type_a>,	: Component description
m * **<data_type_b>**;	: Component description

Often, in such cases it is necessary to explicitly deliver to the decoder the number of times a component is repeated or the overall length of a data structure in bytes. Sometimes it is not.

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

<new_data_type>:=	: Description of data type
<intunti>(n),	: An integer representing the value of "n"
n * **<data_type_a>,**	: Component description
<data_type_b>;	: Component description

In the above example a decoder has to have the value of “n” in order to correctly determine the position of **<data_type_b>**.

In this example the decoder uses the value of “n” to determine the overall length of the component data, and the value of m is determined implicitly:

<new_data_type>:=	: Description of data type
<intunti>(n),	: Length, n, of component data in bytes
m * **<data_type_a>;**	: Component data

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

<new_data_type>:=	: Description of data type
<intunti>(x),	: Select parameter, x
if (x=1) then **<data_type_a>,**	: Included if x equals 1
if (x=2) then **<data_type_b>,**	: Included if x equals 2

A hypothetical example serves to illustrate the basic TPEG notation concepts:

<school_data>:=	: School data
<intunti>(n),	: Number of children in school (n)
n * **<child_data>;**	: Data about children

<child_data>:=	: Information about a child
<intunti>,	: Height in cm
<intunti>,	: Weight in kg
<intunti>;	: Age in years

<intunti>:=	: An integer in the range 0..255
<byte>;	: Primitive element

6.3 Application independent elements

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

6.3.1 Primitive elements

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:

6.3.1.1 Basic numbers:

<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

6.3.1.2 Numerical magnitude:

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. (See Annex B)

<numag>:=	: Counting numbers with magnitude, $0 <= r <= 3 \times 10^6$
<intunti>(n)**;**	: Where $r := (5 + \text{sign}(n-5) \times (\text{abs}(n-5) \bmod 45)) \times 10^{(n-5) \text{ div } 45}$

6.3.1.3 Default character type:

<ch_def>:=	: Default character type
<byte>;	: According to Character Table ISO/IEC 8859-1

Note: Use of **<ch_def>** always specifies usage of character table ISO/IEC 8859-1, irrespective of the particular table selected within an application for strings.

6.3.1.4 Character table identifier:

<chartab>:=	: Character table identifier
<intunti>(t)**;**	: Table used, t, according to Annex A

Explanation: **<chartab>** identifies the character table number used (t) according to the **reference character table index** (See Annex A). The table number indicates the number of bytes used per character (k), the name of the character table, and the size of the character tables (in bytes).

<chartab> is used to switch (within the scope defined within a particular application) the applicability of bytes contained in TPEG strings (both short and long). Characters defined as type **<ch_def>** are not affected by switching **<chartab>.**

6.3.1.5 Bit switch type:

<bit_switch>:=	: Bit switch
<byte>;	: Octet of flags

Explanation: Each bit in this type is a separate flag, which may be used for a variety of purposes, for example indicate the existence of a specific feature in an application.

6.3.1.6 CRC-word data type:

<crc>:=	: Cyclic redundancy check
<intunli>;	: According to CCITT polynomial, over an indicated : Range of elements. (See Annex C)

6.3.2 Compound elements

A Compound element is a basic data structure that comprises more than one primitive data type:

<short_string>:=	: Short string
<intunti>(n),	: Number of bytes, n
n * **<byte>;**	: String of n/k characters

<long_string>:=	: Long string
<intunli>(n),	: Number of bytes, n
n * **<byte>;**	: String of n/k characters

Explanation: 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 (k). Each application will have appropriate mechanisms to attach a code table to the strings which it uses. Where multiple code tables are used, an application needs mechanisms to set the scope of applicability of each table.

In the context of a string a general character may be represented as follows:

Character: = k * **<byte>** (according to the specified or switched character table)

Examples of use may be:

— An application has only one code table, and this is specified at the time of writing.

— The application has a means of describing the code table to be used for strings.

— The application has a means of describing a code table, which applies to a string or group of strings.

— The application may use the position or context of a string to determine the code table, which may be applicable.

Maximum number of characters in a string:

The number of bytes (n) used to represent a string of characters (n/k) is dependent on the code table which is used. The maximum length of a string which may be coded is therefore Int(255/k) for a short string and Int(65535/k) for a long string, where Int indicates the integer part of the result.

Order of characters in a string:

Characters are coded in a string in the order in which they are normally read in their language represented. For example: Arabic letters are read from right to left, and the right-most letter would be coded first.

6.3.2.1 Date and time information:

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

Definition:

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 Annex D of this document.

6.3.2.2 Day mask

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

If the value of **<day_mask>** is set to zero then no day is selected. This can be used to signal there is no repetition on a certain weekday even if the user normally would expect it. Values between 128 and 255 (decimal) are not allowed. The MSB is always set to zero.

<day_mask>:=	: Day mask
<bit_switch>(selector);	
if (selector = 00000000)	: no day selected
if (selector = 0xxxxxx1)	: every Sunday
if (selector = 0xxxxx1x)	: every Monday
if (selector = 0xxxx1xx)	: every Tuesday
if (selector = 0xxx1xxx)	: every Wednesday
if (selector = 0xx1xxxx)	: every Thursday
if (selector = 0x1xxxxx)	: every Friday
if (selector = 01xxxxxx)	: every Saturday

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

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

6.3.2.3 Duration

The definition of the duration time is similar to that of the **<time_t>** type and is represented as an unsigned four byte integer.

<duration>:=	: Time duration
<intunlo>;	: Number of seconds

6.4 Application dependent elements

This section describes the methodology and syntax by which application data structures may be constructed within TPEG. Two basic forms are described: non-declarative and declarative. Declarative structures contain an element which labels the structure, and which may be used by a decoder to determine the functionality of the structure. As such, declarative structures are used where options are required, or where an application needs to build in 'future proofing'. Non-declarative structures do not contain such an element, and are used in positions in which the structure is expected by a decoder.

This document does not specify the structures, which are actually used in TPEG applications. Such specifications are made in the respective parts of CEN ISO/TS 18234. However, examples are given below of how such structures may be built from the primitive elements described above.

6.4.1 Non-declarative structures

Non-declarative structures are built up from several (more than one) elements: primitive, compound or other structures (both non-declarative and declarative).

Examples of the use of a non-declarative structure might be:

EXAMPLE 1:

<activity>:=	: Activity
<time_t>,	: Beginning
<time_t>,	: End
<short_string>;	: Text

EXAMPLE 2:

<wave>:=	: Sound sample
<intunli>,	: Length of samples, n
n * **<intsiti>,**	: Samples

6.4.2 Declarative structures

Declarative structures should be used wherever **future extensions** are envisioned, and where 'future proofing' is a strong requirement.

6.4.2.1 Description

Declarative structures are composed of one or more elements: primitive, compound or other structures (both non-declarative and declarative). The elements are preceded by a structure identifier and a length. The structure identifier permits a decoder to identify the purpose of the data structure in a position where options are possible, and is valid globally within a given application. The same number may be used in different applications for completely different purposes. Within an application a common identifier designates a common structure. The length permits a decoder to skip over the data structure where required, for example in the case in which a new data type is presented to an old decoder. The identifier and length precede the data element(s).

6.4.2.2 Scope

Each declarative structure may be nested within another declarative structure, to any practicable level. This relationship is described as parent-child; where each parent structure may have a number of child structures. The identifier of a child declarative structure is always unique to its parent. This means that two separate parent structures may have each a child with exactly the same identifier, but these children have completely different functions and definitions. Each application frame is intrinsically a declarative structure and consequently all top level declarative structures within an application are unique to that application.

6.4.2.3 Format

<declarative_structure(x)>:=	: Structure template
<intunti>(id),	: Structure identifier (id = x)
<intunli>(n),	: Length of following data in bytes
<....>;	: Data

6.4.2.4 Examples

EXAMPLE 1

<alarm(01)>:=	: Sound alarm
<intunti>(id),	: Alarm identifier, id = 01 hex
<intunli>(n),	: Length of following data in bytes
<time_t>,	: When to wake up
<wave>;	: Sound to wake up to!

EXAMPLE 2

<alarm(02)>:=	: Text alarm
<intunti>(id),	: Alarm identifier, id = 02 hex
<intunli>(n),	: Length of following data in bytes
<time_t>,	: When to display
<short_string>;	: Text to display

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

<appointment>:=	: Appointment
<intunti>(at),	: Alarm type
if (at = 1) then **<alarm(01)>,**	
if (at = 2) then **<alarm(02)>,**	
<activity>;	: What is to be done

6.5 Application design principles

This section 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 proper design principles are not adopted then older decoders may not operate properly, even to the extent that existing features no longer operate. Correct design should permit applications to be upgraded and extended, providing new features to new decoders, and yet permit existing decoders to continue to operate.

6.5.1 Variable data structures

Switches may be included within an application, which permit variations in the subsequent data structure. This may be achieved by using declarative data structures or by explicit switching fields. In the latter case, if new features are likely to be incorporated, attention should be given to the need to include a length element to permit old decoders to 'skip over' new data fields.

6.5.2 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. Declarative structures 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. The highest level example of such a re-usable and extensible structure is the

application data frame itself, whereby the application frame can take any position in the multiplex, yet it is defined only once, and whereby new applications can be added to the protocol suite without affecting an existing decoder's ability to decode the applications for which it was built.

6.5.3 Validity of declarative structures

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

7 TPEG description

7.1 Hierarchy (frame structure)

7.1.1 Diagramatic representation

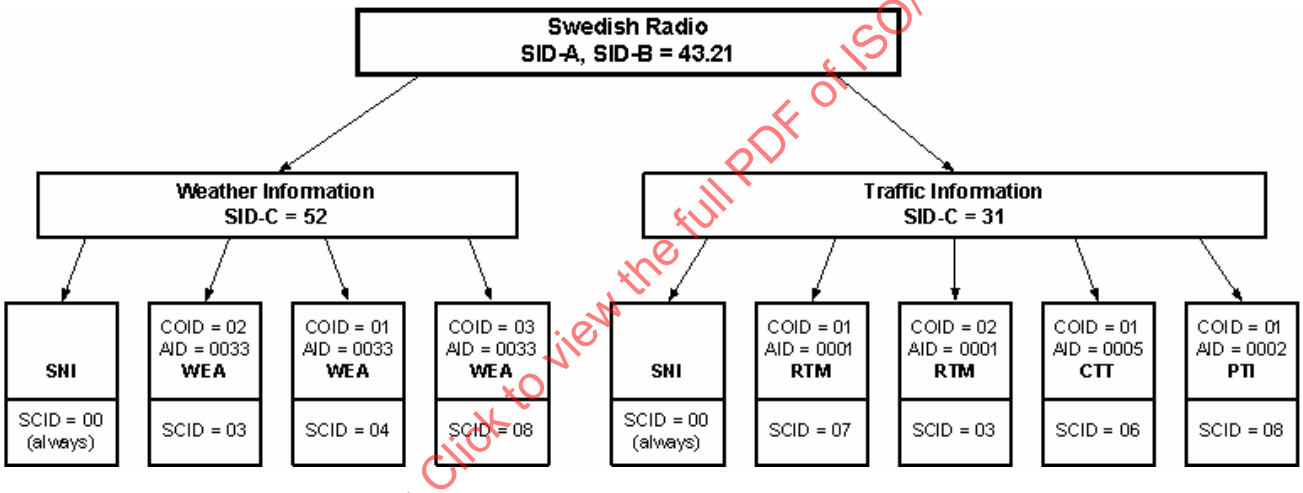

Figure 3 — TPEG Frame Structure, FTY =1 (i.e. conventional data)

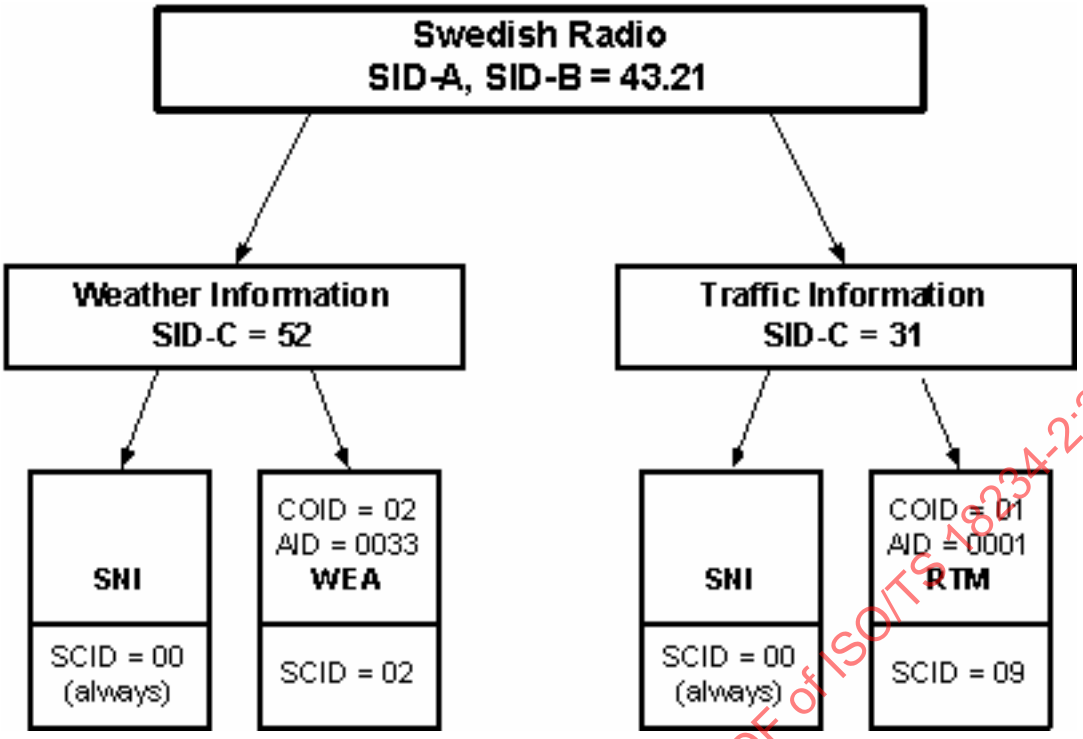

Figure 4 — TPEG Frame Structure, FTY =0 (i.e. stream directory)

7.1.2 Syntactical Representation

The following boxes are the syntactical representation of the TPEG frame structure shown in 7.1.1.

```
<tpeg_stream>:=
infinite *                                  : Control element, (loop continues infinitely)
      [
      n * <intunti>(pad),                   : Any number of padding bytes (pad = 00 hex)
      <transport_frame>                     : Transport frames
      ];
```

```
<transport_frame>:=
      <intunli>(sync),                      : Sync word (sync = FF0F hex)
      <intunli>,                            : Number of bytes in Service Frame
      <crc>(headcrc),                       : Header CRC, calculated according to 7.2.5
      <intunti>(fty),                       : Frame type of transport frame
      if (fty = 0) then
            <service_frame(00)>,
      if (fty = 1) then                     : Service frame of frame type 0, stream directory
            <service_frame(01)>;

                                            : Service frame of frame type 1, conventional data
```

```
<service_frame(00)>:=                       : Stream directory
      <intunti>(n),                         : Number of services
      n * <sid_abc>,                        : Any number of Service IDs
      <crc>;                                : CRC of Service IDs
```

<sid_abc>:=	: Any service identification
<intunti>,	: Service identification-A
<intunti>,	: Service identification-B
<intunti>;	: Service identification-C

<service_frame(01)>:=	: Conventional data
<sid_abc>,	: Service identification
<intunti>,	: Encryption indicator
f_n**(<serv_comp_multiplex>);**	: Function f_n (…) is utilized according to the chosen encryption algorithm

<serv_comp_multiplex>:=	
n * **<serv_comp_frame()>;**	: Any number of component frames

<serv_comp_frame>:=	: Service component frame template
<intunti>(scid)**,**	: Service component identifier (scid = x)
<intunli>(n)**,**	: Length, n, of component data in bytes
<crc>,	: Header CRC according to 7.5.4
n * **<byte>;**	: n bytes of component data

7.1.3 Rules for the data type representation

Some **general rules** used for data structures can be illustrated easily:

NOTE For ease of reading the expression “data element” is used both for a single data type and a composite data structure.

— A data element is written in bold lower case letters in one single expression.

— A data element is framed by angle brackets “ **< >** ” .

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

— The end of a data element in indicated by a semicolon “ **;** ”.

— Any descriptor may be added to a data element.

— A descriptor is written within round brackets “ ( ) ”.

— The descriptor contains the value, the name or the quality of associated type.

— Types inside a structure are separated by commas “ **,** ”.

— Control statements (e.g. “if” , “infinite”, etc .) are noted in standard lower case letters.

— The expression “ n * ” indicates the occurrence of a data element for n times.

— Square brackets “ [ ] ” group together a block of data elements.

— A function “ *fn* ( ) ” that is calculated over a data element is indicated by italic lower case letters.

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

— Nested structures are defined from top to bottom (i.e. higher levels first, then lower levels).

— For logical separation it is possible to draw a box around a data type block.

— Primitive elements or basic data types are defined in section 6 of this document.

— A general declarative structure is noted with round bold brackets

EXAMPLE 1: “<component_frame()>”

— A declarative structure that is used as a template (i.e. more than once) has an x inside the round brackets

EXAMPLE 2: “<component_frame(x)>”

— A specific instance of a declarative structure is described by a distinctive number

EXAMPLE 3: “<component_frame(01)>”

7.2 Transport level

7.2.1 Overall frame structure

The byte stream contains consecutive transport frames. Each frame includes:

— The synchronization word (syncword)	2 bytes
— The length of the service frame (field length)	2 bytes
— The header CRC	2 bytes
— The frame type indicator	1 byte
— 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 00 hex (padding bytes).

7.2.2 Syncword

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

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

7.2.3 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.

7.2.4 Frame type

The frame type 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 Annex C.

7.2.5 Header CRC

The Header CRC is two bytes long, and is based on the ITU-T (formerly CCITT) 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 Annex C.

7.2.6 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.

7.2.7 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).

7.3 Service level

7.3.1 Service frame of frame type = 0

This service frame comprises:

— Number of services (n) 1 byte

— n *(SID-A, SID-B, SID-C) n * (3 bytes)

— CRC 2 bytes

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

7.3.2 Service frame of frame type = 1

Each service frame comprises:

— SID-A, SID-B, SID-C 3 bytes

— The encryption indicator 1 byte

— 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 trames. Each service frame includes service information that comprises the service identification elements and the encryption indicator.

7.3.2.1 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

7.3.3 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:

Range of TPEG Class-A = 0.0.x – 127.255.x

Range of TPEG Class-B = 128.0.0 – 255.255.255

The value of x can be allocated by service providers in Class-A and can take any value between 0 and 255. A Class-A address is intended for large service providers that are able to manage the service allocation by themselves. A Class-B address is used for e.g. local radio stations, which provide only one service.

7.4 Service component multiplex level

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 00hex) or is left unchanged (if the encryption indicator = 00hex). The length of the resultant data must be less than or equal to 65531 bytes.

7.5 Service component level

7.5.1 Overall

Each component frame comprises:

— The service component identifier 1 byte

— The length of the component data (field length) 2 bytes

— The component header CRC 2 bytes

— The component data m bytes

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.

7.5.2 Service component identifier

The service component identifier with the value 0 is reserved for the SNI application.

7.5.3 Field length

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

7.5.4 Component header CRC

The component header CRC is two bytes long, and based on the ITU-T (formerly CCITT) 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 Annex C.

Annex A
(normative)

Character tables

A.1 Character tables

The default character coding table used in TPEG is ISO/IEC 8859-1.

A.2 Reference character table index

Table A.1 — Reference character table index

t = Char-Tab	k = bytes/char	Name of Character Table
0	-	Reserved
1	1	ISO/IEC 8859-1 (Default)
2	1	ISO/IEC 8859-2
3	1	ISO/IEC 8859-3
4	1	ISO/IEC 8859-4
5	1	ISO/IEC 8859-5
6	1	ISO/IEC 8859-6
7	1	ISO/IEC 8859-7
8	1	ISO/IEC 8859-8
9	1	ISO/IEC 8859-9
10	1	ISO/IEC 8859-10
11	1	Reserved
12	1	Reserved
13	1	ISO/IEC 8859-13
14	1	ISO/IEC 8859-14
15	1	ISO/IEC 8859-15
...	....	....
125	1	Unicode ISO/IEC 10646 UTF-8
126	2	Unicode ISO/IEC 10646 UTF-16
127	4	Unicode ISO/IEC 10646 UTF-32
128		Reserved
...	....	....
255		Reserved

The selection of TPEG coding tables is implemented according to the character table switch in 6.3.1.4 with the following value ranges:

a) The range t = 1 to 127 is reserved for standardized character tables.

b) The range t = 128 to 255 may be freely used by a service provider and may indicate the use of proprietary character tables. In combination with the service provider identification this guarantees uniqueness.

Annex B
(normative)

Method for coding quantities of objects

B.1 Numag derivation

Within applications of TPEG there is a frequent need to describe with a single byte a quantity of people, objects, etc., using a non-linear coding system which provides a high resolution for low numbers and progressively lower resolution for higher numbers.

The primitive **<numag>** describes, in a single byte, quantities which lie in the range 0-3 000 000.

<numag>:=	: Counting numbers with magnitude, $0 <= r <= 3 \times 10^6$
<intunti>(n);	: Where $r := (5 + \text{sign}(n-5) \times (\text{abs}(n-5) \bmod 45)) \times 10^{(n-5) \text{ div } 45}$

The following formula translates the value, n, to the result, r.

$r := (5 + \text{sign}(n-5) \times (\text{abs}(n-5) \bmod 45)) \times 10^{(n-5) \text{ div } 45}$

This formula, which produces the sequence of numbers shown in table B.2, is calculated as follows:

a) n is an integer in the range 0.255 and is used to code the number, r

b) Intermediate values are generated:

$a := \text{sign}(n-5)$

$b := \text{abs}(n-5) \bmod 45$

$c := (n-5) \text{ div } 45$

c) The result, r, is generated from these intermediate values as follows:

$r := (a \times b + 5) \times 10^c$

B.2 Numag table

Table B.1 — Numag table

n:	R:	n:	r:	n:	r:	n:	r:
0	0	64	190	128	3 800	192	120 000
1	1	65	200	129	3 900	193	130 000
2	2	66	210	130	4 000	194	140 000
3	3	67	220	131	4 100	195	150 000
4	4	68	230	132	4 200	196	160 000
5	5	69	240	133	4 300	197	170 000
6	6	70	250	134	4 400	198	180 000
7	7	71	260	135	4 500	199	190 000
8	8	72	270	136	4 600	200	200 000
9	9	73	280	137	4 700	201	210 000
10	10	74	290	138	4 800	202	220 000
11	11	75	300	139	4 900	203	230 000
12	12	76	310	140	5 000	204	240 000
13	13	77	320	141	6 000	205	250 000
14	14	78	330	142	7 000	206	260 000
15	15	79	340	143	8 000	207	270 000
16	16	80	350	144	9 000	208	280 000
17	17	81	360	145	10 000	209	290 000
18	18	82	370	146	11 000	210	300 000
19	19	83	380	147	12 000	211	310 000
20	20	84	390	148	13 000	212	320 000
21	21	85	400	149	14 000	213	330 000
22	22	86	410	150	15 000	214	340 000
23	23	87	420	151	16 000	215	350 000
24	24	88	430	152	17 000	216	360 000
25	25	89	440	153	18 000	217	370 000
26	26	90	450	154	19 000	218	380 000
27	27	91	460	155	20 000	219	390 000
28	28	92	470	156	21 000	220	400 000
29	29	93	480	157	22 000	221	410 000
30	30	94	490	158	23 000	222	420 000
31	31	95	500	159	24 000	223	430 000
32	32	96	600	160	25 000	224	440 000
33	33	97	700	161	26 000	225	450 000
34	34	98	800	162	27 000	226	460 000
35	35	99	900	163	28 000	227	470 000
36	36	100	1 000	164	29 000	228	480 000
37	37	101	1 100	165	30 000	229	490 000
38	38	102	1 200	166	31 000	230	500 000
39	39	103	1 300	167	32 000	231	600 000
40	40	104	1 400	168	33 000	232	700 000
41	41	105	1 500	169	34 000	233	800 000
42	42	106	1 600	170	35 000	234	900 000
43	43	107	1 700	171	36 000	235	1 000 000
44	44	108	1 800	172	37 000	236	1 100 000
45	45	109	1 900	173	38 000	237	1 200 000
46	46	110	2 000	174	39 000	238	1 300 000
47	47	111	2 100	175	40 000	239	1 400 000
48	48	112	2 200	176	41 000	240	1 500 000
49	49	113	2 300	177	42 000	241	1 600 000
50	50	114	2 400	178	43 000	242	1 700 000
51	60	115	2 500	179	44 000	243	1 800 000
52	70	116	2 600	180	45 000	244	1 900 000
53	80	117	2 700	181	46 000	245	2 000 000
54	90	118	2 800	182	47 000	246	2 100 000
55	100	119	2 900	183	48 000	247	2 200 000
56	110	120	3 000	184	49 000	248	2 300 000
57	120	121	3 100	185	50 000	249	2 400 000
58	130	122	3 200	186	60 000	250	2 500 000
59	140	123	3 300	187	70 000	251	2 600 000
60	150	124	3 400	188	80 000	252	2 700 000
61	160	125	3 500	189	90 000	253	2 800 000
62	170	126	3 600	190	100 000	254	2 900 000
63	180	127	3 700	191	110 000	255	3 000 000

Annex C
(normative)

CRC calculation

C.1 CRC calculation

The TPEG **<crc>** primitive is represented by a word **<intunli>** which itself represents the result of a 16-bit cyclic redundancy check (CRC) calculation upon a designated range of elements.

The calculation starts with the most significant bit of the first designated element field and ends with the least significant bit of the last byte of the last designated element.

The divisor polynomial used to generate the CRC is:

$$x^{16} + x^{12} + x^{5} + 1$$

The CRC is initialized by a value of FFFF hex, and the two check bytes are formed from the inverse of the result (1's complement). The eight most significant bits are represented by the first check field byte, and the eight least significant bits are represented by the last check field byte.

Example: When applied to a sequence of 47 bytes:

32 44 31 31 31 32 33 34 30 31 30 31 30 35 41 42 43 44 31 32 33 46 30 58 58 58 58 31 31 30 36 39 32 31 32 34 39 31 30 30 30 33 32 30 30 36 36 hex,

the CRC generated is 97 23 hex.

C.2 ITU-T (formerly CCITT) CRC calculation in PASCAL

Type STRING is a PACKED ARRAY of CHAR with zero'th element holding the length of the string.

SWAP is a library function that swaps the high- and low-order bytes of the argument.

EXAMPLE 1:

```
VAR X: WORD;
BEGIN
 X:= SWAP ($1234)        [$3412]
END;
```

LO is a library function which returns the low-order byte of the argument.

EXAMPLE:

```
VAR W: WORD;
BEGIN
 W:= LO ($1234)       [$34]
END;

FUNCTION CRCVALUE (STRINGTOEVAL : STRING): INTEGER;
VAR
COUNT: BYTE;
TEMPCRC: WORD;
BEGIN
 TEMPCRC:= $FFFF;
 FOR COUNT:= 1 TO LENGTH (STRINGTOEVAL) DO
```

```
        BEGIN
            TEMPCRC:= SWAP (TEMPCRC) XOR ORD (STRINGTOEVAL [COUNT]);
            TEMPCRC:= TEMPCRC XOR (LO (TEMPCRC) SHR 4);
            TEMPCRC:= TEMPCRC XOR (SWAP (LO (TEMPCRC)) SHL 4) XOR (LO (TEMPCRC) SHL 5)
        END;
        CRCVALUE:= TEMPCRC XOR $FFFF
       END; [OF FUNCTION CRCVALUE]
```

C.3 ITU-T (formerly CCITT) CRC calculation in C notation

```
#define swap(a) (((a)<<8)|((a)>>8))
//------------------------------------------------------------------------------
unsigned short usCalculCRC(unsigned char *buf,unsigned long lg)
//------------------------------------------------------------------------------
{
unsigned short crc;
unsigned long count;
crc= 0xFFFF;
for (count= 0; count < lg; count++)
    {
    crc = (unsigned short) (swap(crc) ^ (unsigned short)buf[count]);
    crc ^= ((unsigned char)(crc) >> 4);
    crc = (unsigned short) (crc ^ (swap((unsigned char)(crc)) << 4)
            ^ ((unsigned char)(crc) << 5));
    }

return((unsigned short)(crc ^ 0xFFFF));
}
```