ETSI TS 102 323 V1.1.1 (2004-09)

Technical Specification

Digital Video Broadcasting (DVB); Carriage and signalling of TV-Anytime information in DVB transport streams

Reference
DTS/JTC-DVB-163

Keywords
broadcasting, content, digital, DVB, TV

ETSI

650 Route des Lucioles
F-06921 Sophia Antipolis Cedex - FRANCE

Tel.: +33 4 92 94 42 00 Fax: +33 4 93 65 47 16

Siret N° 348 623 562 00017 - NAF 742 C
Association à but non lucratif enregistrée à la
Sous-Préfecture de Grasse (06) N° 7803/88

Contents

Intellectual Property Rights

IPRs essential or potentially essential to the present document may have been declared to ETSI. The information pertaining to these essential IPRs, if any, is publicly available for **ETSI members and non-members**, and can be found in ETSI SR 000 314: *"Intellectual Property Rights (IPRs); Essential, or potentially Essential, IPRs notified to ETSI in respect of ETSI standards"*, which is available from the ETSI Secretariat. Latest updates are available on the ETSI Web server (http://webapp.etsi.org/IPR/home.asp).

Pursuant to the ETSI IPR Policy, no investigation, including IPR searches, has been carried out by ETSI. No guarantee can be given as to the existence of other IPRs not referenced in ETSI SR 000 314 (or the updates on the ETSI Web server) which are, or may be, or may become, essential to the present document.

Foreword

This Technical Specification (TS) has been produced by Joint Technical Committee (JTC) Broadcast of the European Broadcasting Union (EBU), Comité Européen de Normalisation ELECtrotechnique (CENELEC) and the European Telecommunications Standards Institute (ETSI).

NOTE: The EBU/ETSI JTC Broadcast was established in 1990 to co-ordinate the drafting of standards in the specific field of broadcasting and related fields. Since 1995 the JTC Broadcast became a tripartite body by including in the Memorandum of Understanding also CENELEC, which is responsible for the standardization of radio and television receivers. The EBU is a professional association of broadcasting organizations whose work includes the co-ordination of its members' activities in the technical, legal, programme-making and programme-exchange domains. The EBU has active members in about 60 countries in the European broadcasting area; its headquarters is in Geneva.

European Broadcasting Union
CH-1218 GRAND SACONNEX (Geneva)
Switzerland
Tel: +41 22 717 21 11
Fax: +41 22 717 24 81

Founded in September 1993, the DVB Project is a market-led consortium of public and private sector organizations in the television industry. Its aim is to establish the framework for the introduction of MPEG-2 based digital television services. Now comprising over 200 organizations from more than 25 countries around the world, DVB fosters market-led systems, which meet the real needs, and economic circumstances, of the consumer electronics and the broadcast industry.

Introduction

Phase 1 of the work of the TV-Anytime Forum (TVAF) has defined key technologies for Personal Digital Recorders (PDRs). DVB has adopted these specifications as the basis of support for PDRs on DVB networks. The present document provides an implementation of the TV-Anytime phase one on DVB transport streams in a way that enhances and extends existing DVB functionality.

1 Scope

The present document describes a method of supporting Personal Digital Recorders (PDRs) in DVB broadcast networks. It was developed to fulfil the commercial requirements for supporting PDRs detailed in DVB commercial module document PDR008. The data types and protocols defined in the present document are based upon TV-Anytime phase 1 specifications TS 102 822-2 [3], TS 102 822-3-1 [4], TS 102 822-3-2 [5] and TS 102 822-4 [6].

2 References

The following documents contain provisions which, through reference in this text, constitute provisions of the present document.

- References are either specific (identified by date of publication and/or edition number or version number) or non-specific.
- For a specific reference, subsequent revisions do not apply.
- For a non-specific reference, the latest version applies.

Referenced documents which are not found to be publicly available in the expected location might be found at http://docbox.etsi.org/Reference.

[1] ETSI EN 300 468 (V1.4.1): "Digital Video Broadcasting (DVB); Specification for Service Information (SI) in DVB systems".

[2] IETF RFC 2396: "Uniform Resource Identifiers (URI): Generic Syntax".

[3] ETSI TS 102 822-2: "Broadcast and On-line Services: Search, select and rightful use of content on personal storage systems ("TV-Anytime Phase 1"); Part 2: System description".

[4] ETSI TS 102 822-3-1: "Broadcast and On-line Services: Search, select, and rightful use of content on personal storage systems ("TV-Anytime Phase 1"); Part 3: Metadata; Sub-part 1: Metadata schemas".

[5] ETSI TS 102 822-3-2: "Broadcast and On-line Services: Search, select, and rightful use of content on personal storage systems ("TV-Anytime Phase 1"); Part 3: Metadata; Sub-part 2: System aspects in a uni-directional environment".

[6] ETSI TS 102 822-4: "Broadcast and On-line Services: Search, select, and rightful use of content on personal storage systems ("TV-Anytime Phase 1"); Part 4: Content referencing".

[7] ISO/IEC 13818-1 (2000 - Amendment 1): "Carriage of metadata over ISO/IEC 13818-1 streams".

[8] ISO/IEC 13818-1 (2000): "Information technology - Generic coding of moving pictures and associated audio information: Systems".

[9] ISO/IEC 15938-1 (2001): "Information technology - Multimedia content description interface - Part 1: Systems".

[10] ETSI TS 102 812 (V1.2.1): "Digital Video Broadcasting (DVB); Multimedia Home Platform (MHP) Specification 1.1.1".

[11] IETF RFC 1591: "Domain Name System Structure and Delegation".

[12] ISO/IEC 13818-6: "Information technology - Generic coding of moving pictures and associated audio information - Part 6: Extensions for DSM-CC".

[13] ISO 8601 (2002): "Data elements and interchange formats - Information interchange - Representation of dates and times".

[14] W3C Recommendation: "XML Schema Part 2: Datatypes". http://www.w3.org/TR/xmlschema-2/.

[15] IETF RFC 1951: "DEFLATE Compressed Data Format Specification version 1.3".

[16] ETSI TS 102 822-6-1: "Broadcast and On-line Services: Search, select, and rightful use of content on personal storage systems ("TV-Anytime Phase 1"); Part 6: Delivery of metadata over a bi-directional network; Sub-part 1: Service and transport".

[17] IETF RFC 1945: "Hypertext Transfer Protocol - HTTP/1.0. T".

[18] IETF RFC 1950: "ZLIB Compressed Data Format Specification version 3.3".

[19] ISO 646: "Information technology - ISO 7-bit coded character set for information interchange".

[20] ISO/IEC 15938-2: "Information technology - Multimedia content description interface - Part 2: Description definition language".

[21] ISO 639-2: "Codes for the representation of names of languages - Part 2: Alpha-3 code".

3 Definitions and abbreviations

3.1 Definitions

For the purposes of the present document, the following terms and definitions apply:

authority: term used in TS 102 822-3-1 [4] and TS 102 822-3-2 [5] for organization that creates CRIDs

NOTE: In the present document the term *CRID authority* is used instead.

broadcaster (service provider): organization which assembles a sequence of events or programmes to be delivered to the viewer based upon a schedule

broadcast time: time and date on which an event is actually broadcast

EXAMPLE: A news programme may actually start at 18:04:38.21. See also *published time* and *scheduled time*.

complete set of CRI data: single set of CRI data that carries all CRI data available for a CRID authority

NOTE: See also *set of CRI data*.

context: grouping of resolution providers for the purposes of managing the discovery of TV-Anytime information

NOTE: A context could relate to a particular bouquet for which CRI is provided, or a network, etc.

CRID authority: organization that creates CRIDs

NOTE: This is termed in TS 102 822-4 [6] as simply an *authority*.

CRID resolution: location resolution

CRID result: the (possibly empty) set of locators or CRIDs that a CRID resolves to

CRI service: coherent set of CRI data provided by a resolution provider and describing one or more CRID authorities

location resolution: process for establishing the address (location and time) of content instance(s) from a CRID

NOTE: Sometimes termed "CRID resolution".

locator: URI reference to the location of content

NOTE: A locator can reference a file on the internet, an event on a DVB network, etc. For broadcast content a locator would include DVB service, date and time information.

metadata service: coherent set of TV-Anytime metadata carried within a DVB object carousel and distinguished by a metadata_service_ID

NOTE: See ISO/IEC 13818-1 (Amendment 1) [7].

promotional link: reference to material related to content currently being viewed

published time: time and date on which a service provider publishes that an event starts

EXAMPLE: A news programme may have a published time of 18:00. See also *broadcast time* and *scheduled time*.

related material: content that is related in some way to the present content

EXAMPLE: Other events in the same series or the web-page for the current programme.

resolution provider: body which provides location resolution

NOTE: In TS 102 822-4 [6] this term is a synonym for the term *resolving authority*, a term which is not used in the present document.

scheduled time: time and date on which a service provider has scheduled to start an event

EXAMPLE: A news programme may have a scheduled time of 18:02. See also *broadcast time* and *published time*.

set of CRI data: all CRI data for a single CRID authority at a single location

NOTE: See also *complete set of CRI data*.

3.2 Abbreviations

For the purposes of the present document, the following abbreviations apply:

BiM	Binary format for Multimedia description streams
CIT	Content Identifier Table
CRI	Content Referencing Information
CRID	Content Reference IDentifier
DNS	Domain Naming System
DSI	Download Server Initiate
DVB	Digital Video Broadcasting
EIT	Event Information Table
IP	Internet Protocol
MJD	Modified Julian Date
MPEG	Motion Picture Experts Group
PDR	Personal Digital Recorder
RAR	Resolving Authority Record
RCT	Related Content Table
RNT	RAR Notification Table
SI	Service Information
TVAF	TV-Anytime Forum
TVA_id	TV-Anytime event identifier
UML	Unified Modelling Language
URI	Uniform Resource Indicator
URL	Uniform Resource Locator
UTC	Co-ordinated Universal Time
XML	eXtensible Markup Language
TS	Transport Stream
IMI	Instance Metadata Identifier

4 Overview

The present document defines how PDR devices can be supported using TV-Anytime phase 1 specifications on DVB transport streams. The TV-Anytime phase 1 specifications relevant to the present document are:

- TS 102 822-2 [3]: "System description";
- TS 102 822-3-1 [4]: "Metadata schemas";
- TS 102 822-3-2 [5]: "System aspects in a uni-directional environment"; and
- TS 102 822-4 [6]: "Content referencing".

The TV-Anytime process for recording content is "search, select, acquire". Metadata is searched for the content the viewer wishes to record, the viewer selects the correct content and the PDR then records it. Figure 1 illustrates how this process is interpreted in a DVB transport stream environment.

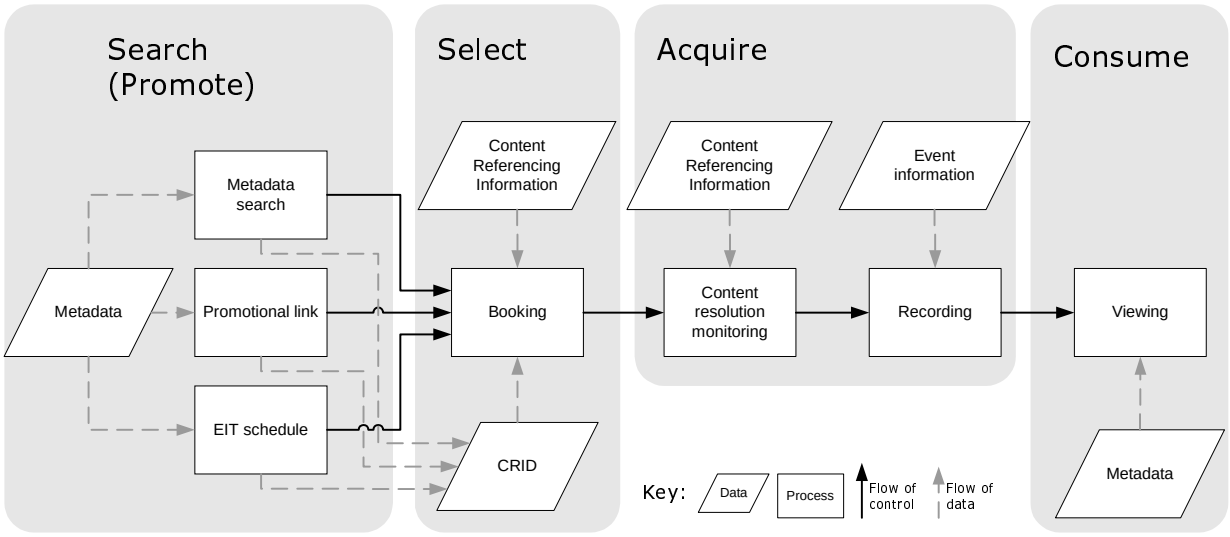

Figure 1: Overview of PDR process

A number of interrelated technologies are defined that can be used to enable various PDR applications. Figure 2 illustrates how the present document integrates these technologies into DVB transport streams. The main sections of the present document are listed below. Each section has a more detailed introduction as its first subsection.

a) **TV-Anytime information discovery (clause 5):** Defines how TV-Anytime metadata services and CRI services are discovered on DVB transport streams. Central to this is the resolution provider Notification Table (RNT), which lists resolution providers and the CRID authorities the resolution providers provide information on. The RNT and related descriptors implements the TV-Anytime RAR structure, as defined in TS 102 822-4 [6].

b) **CRIDs in DVB networks (clause 6):** Defines how URIs (including CRIDs) shall be encoded in the data structures defined in the present document. This clause also describes the rules for default authorities which can be used to abbreviate CRID strings, saving on bandwidth. CRIDs are defined in TS 102 822-4 [6].

c) **Content resolution (clause 7):** Defines how CRIDs shall be resolved (i.e. content resolution) in DVB transport streams and how this process works when more than one relevant CRI service is available. This clause also defines how CRI data, the information required for content resolution, shall be delivered in DVB transport streams.

d) **Profile of TVA metadata types (clause 8):** Defines restrictions on specific parts of the TVA metadata schema. These restrictions are necessary to provide an integrated and self-consistent TV-Anytime toolkit on DVB transport streams.

e) **Delivery of metadata (clause 9):** Defines how TV-Anytime metadata shall be carried on DVB transport streams. TV-Anytime metadata is defined in TS 102 822-3-1 [4]. Clause 9 is an implementation of TS 102 822-3-2 [5], which defines how TV-Anytime metadata is delivered in unidirectional environments. This clause includes a set of profiled metadata indices based on that specification.

f) **Promotional links (clause 10):** Defines a mechanism for referencing related material in a real-time manner. This mechanism is for providing the viewer with the opportunity to book content related to what is being viewed, e.g. booking a film whilst watching it being trailed.

g) **Accurate recording (clause 11):** Defines how the receiver uses the different recording modes signalled in locators in CRI data. The different modes are intended to allow easy adoption of TV-Anytime whilst providing increasing functionality for more advanced broadcaster infrastructures.

h) **Extensions to DVB SI (clause 12):** Defines how TV-Anytime technologies can be integrated into the DVB SI Event Information Table (EIT). This allows those platforms that use the EIT to provide schedule information, to use TV-Anytime content resolution and metadata.

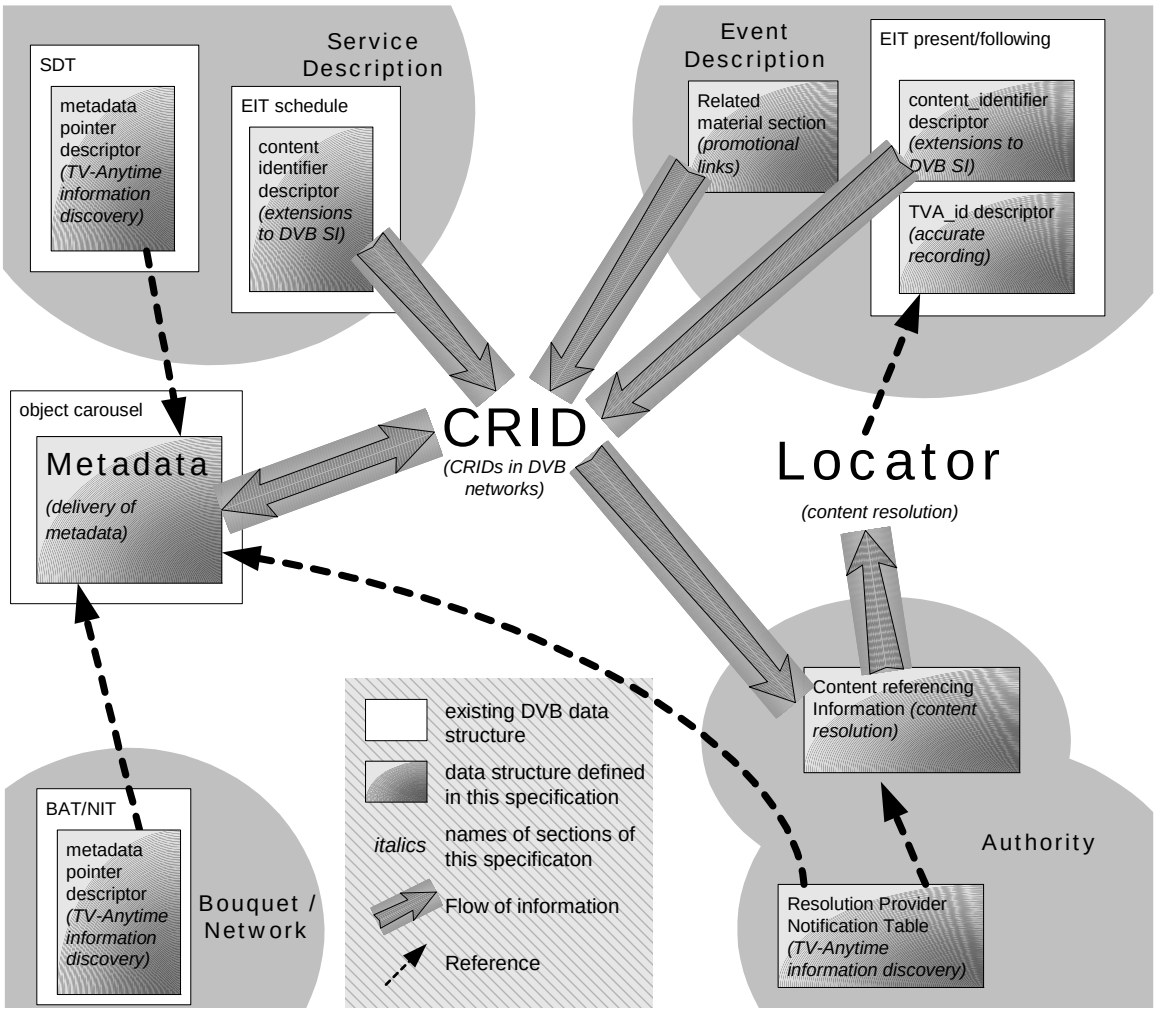

Figure 2: Overview of defined technologies

5 TV-Anytime information discovery

5.1 Introduction

The TV-Anytime Forum has defined a rich set of content referencing information and metadata. This information may be carried on a broadcast network or on other networks such as an IP network. On a broadcast network metadata is carried in metadata services and content referencing information is carried in content referencing services. A receiver accessing TV-Anytime information first has to discover the location of this data. Content referencing services are located by accessing the resolution provider Notification Table (RNT). Metadata services are located by following linkage that may be present in a number of places, depending on whether the metadata sought relates to content, a service's schedule, or another type of entity.

A key concept in TV-Anytime for discovering content referencing information is the resolution provider. On DVB transport streams a resolution provider is a separate entity described by a resolution provider Notification Table (RNT). The resolution provider entity is independent of networks, bouquets and DVB services. Therefore, a resolution provider may provide content resolving information for CRIDs that refer to content carried on one or more networks, bouquets or DVB services. Conversely, a network, bouquet or DVB service may deliver content described by more than one resolution provider.

NOTE: A resolution provider may carry information on CRIDs published by one or more CRID authorities. For details on CRID authorities and resolution providers, see TS 102 822-4 [6].

This clause defines how information regarding resolving authorities entities shall be represented and discovered on a DVB broadcast network and how content referencing services and metadata services shall be discovered.

5.2 Resolution providers

5.2.1 Discovering RARs

On a DVB transport stream, Resolving Authority Records (RARs) are represented by RAR descriptors, which are carried in the resolution provider Notification Table (RNT) (see clauses 5.2.2, 5.3.5 and 5.3.6). This clause describes the process for discovering RARs relevant to a given combination of resolution provider and CRID authority. See figure 3 for a graphical representation of an example implementation of this process.

NOTE 1: The term "resolving authority" is a synonym for "resolution provider." Therefore a Resolving Authority Record (RAR) refers to a particular resolution provider.

RNT subtables shall be carried on a fixed PID, with each RNT subtable being distinguished by its context_id, context_id_type and the transport stream the subtable is carried on. The context_id may be a bouquet ID, original network id, or another type of identifier. The type of value carried by the context_id is identified by the value of context_id_type (see table 2). Each RNT subtable can carry information on one or more resolution providers, each being distinguished by their resolution_provider_name. A receiver may be configured by means outside the scope of the present document to know the values of context_id, context_id_type and resolution_provider_name it should use.

NOTE 2: The use of the context_id_type allows a RNT subtable to be targeted at a particular population of receivers in the best way for a given circumstance. The best value of context_id_type to use may depend, for example, on whether or not a network uses bouquets, or the way network_ids are used.

If the RNT subtable on the current transport stream does not provide information on the desired combination of CRID authority and resolution provider, the RNT may include a RNT scan descriptor. This descriptor shall contain a complete list of transport streams where RNT subtables with the same context_id and context_id_type are available. If a RNT scan descriptor is not present in a RNT subtable, that RNT subtable shall list all CRI sets for all CRID authorities for all resolution providers for that combination of context_id and context_id_type.

It is possible that CRI sets are available from multiple resolution providers for a particular CRID authority.

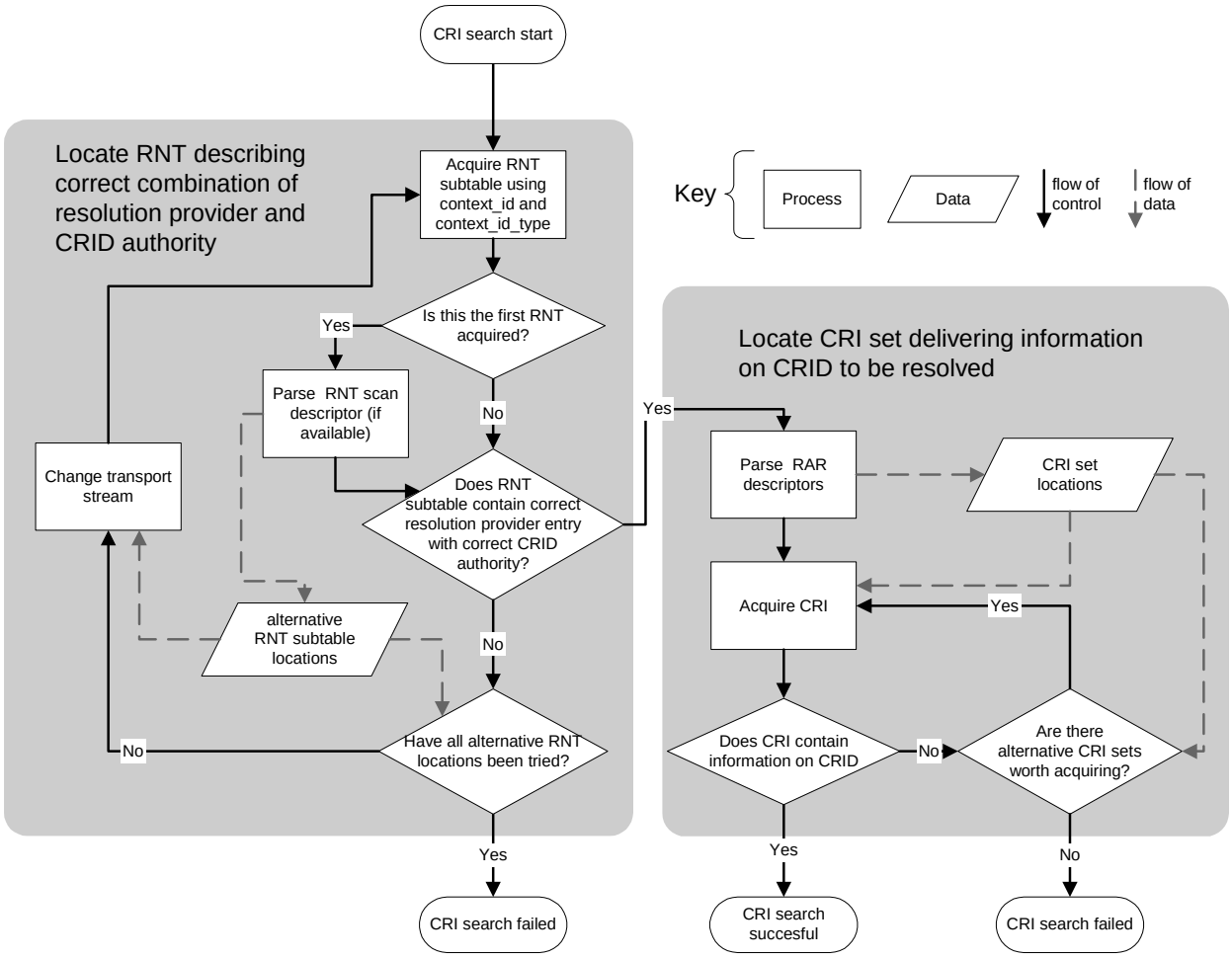

Figure 3: Example procedure for acquiring CRI (informative)

5.2.2 Resolution provider notification table

The resolution provider Notification Table (RNT) carries information provided by resolution providers relating to CRID authorities (see TS 102 822-4 [6], clause 6), providing the locations of CRI and metadata for those CRID authorities.

Each RNT subtable provides information on a particular context (see clause 5.2.1). An RNT subtable is distinguished by context_id, context_id_type and the transport stream the RNT is carried on. RNTs with the same context_id and context_id_type but carried on different transport streams shall not be considered the same subtable.

The RNT shall be segmented into sections using the syntax of table 1. Sections forming part of the RNT shall be carried in Transport Stream (TS) packets with a PID value of 0x0016.

Table 1: Resolution provider notification section

Name	Number of bits	Identifier
`resolution_authority_notification_section() {`		
`table_id`	8	uimsbf
`section_syntax_indicator`	1	bslbf
`reserved`	1	bslbf
`reserved`	2	bslbf
`section_length`	12	uimsbf
`context_id`	16	uimsbf
`reserved`	2	bslbf
`version_number`	5	uimsbf
`current_next_indicator`	1	bslbf
`section_number`	8	uimsbf
`last_section_number`	8	uimsbf
`context_id_type`	8	uimsbf

Name	Number of bits	Identifier
common_descriptors_length	12	uimsbf
reserved	4	bslbf
for (i=0; i<N1; i++) {		
descriptor()		
}		
for (i<0; i<N2; i++) {		
resolution_provider_info_length	12	uimsbf
reserved	4	bslbf
resolution_provider_name_length	8	uimsbf
for (j<0; j<resolution_provider_name_length; j++) {		
resolution_provider_name_byte	8	uimsbf
}		
resolution_provider_descriptors_length	12	uimsbf
reserved	4	bslbf
for (j=0; j<N3; j++) {		
descriptor()		
}		
for (j=0; j<N4; j++) {		
CRID_authority_name_length	8	uimsbf
for (k<0; k<CRID_authority_name_length; k++) {		
CRID_authority_name_byte	8	uimsbf
}		
CRID_authority_descriptors_length	12	uimsbf
reserved	4	bslbf
for (k=0; k<N5; k++) {		
CRID_authority_descriptor()		
}		
}		
}		
CRC_32	32	rpchof
}		

table_id: This field shall be set to 0x79.

section_syntax_indicator: This field shall be set to "1".

reserved: Fields marked as reserved shall have all bits set to "1".

section_length: This field specifies the number of bytes of the section, starting immediately following the section_length fields and including the CRC. The section_length shall not exceed 4 093 so that the entire section has a maximum length of 4 096.

context_id: This field identifies a particular context to which this subtable applies.

version_number: This 5-bit field is the version number of the subtable. The version_number shall be incremented by 1 when a change in the information carried within the subtable occurs. When it reaches value 31, it wraps around to 0.

current_next_indicator: This 1-bit indicator shall be set to "1".

section_number: This 8-bit field gives the number of the section. The section_number of the first section in the subtable shall be "0x00". The section_number shall be incremented by 1 with each additional section with the same table_id, context_id and context_id_type.

last_section_number: This 8-bit field indicates the number of the last section (that is, the section with the highest section_number) of the subtable of which this clause is part.

context_id_type: This field defines the type of the context to which this subtable applies. It shall be encoded according to table 2.

Table 2: context_id_type

Value	Semantics
0x00	context_id is a value of bouquet_id
0x01	context_id is a value of original_network_id.
0x02	context_id is a value of network_id
0x03-0x7F	DVB reserved
0x80-0xFF	user defined

common_descriptors_length: The total length in bytes of the following descriptors.

resolution_provider_info_length: This field specifies the number of bytes in this resolution provider entry, starting immediately after this field and including information on all CRID authorities within this entry.

resolution_provider_name_length: The total length in bytes of the resolution provider name.

resolution_provider_name_byte: This byte forms part of a sequence that is the resolution provider name for this entry. The resolution provider name is a registered internet domain name. See RFC 1591 [11] for DNS name registration. The resolution provider name is case insensitive and must be a fully qualified name according to the rules given by RFC 1591 [11]. See TS 102 822-4 [6], clause 11.1.

resolution_provider_descriptors_length: The total length in bytes of the following descriptors.

CRID_authority_name_length: The total length in bytes of the CRID authority name.

CRID_authority_name_byte: This byte forms part of a sequence that is the CRID authority name string. The encoding of this field follows the same rules as for the resolution provider name. See TS 102 822-4 [6], clause 6.

CRID_authority_descriptors_length: The total length in bytes of the following descriptors.

CRC_32: This is a 32-bit field that contains the CRC value that gives a zero output of the registers in the decoder defined in EN 300 468 [1] after processing the entire private section.

5.3 Descriptors

5.3.1 Parsing of descriptors

When parsing any one of the descriptors defined by the present document, receivers shall always use the descriptor length field to determine the length of the descriptor.

NOTE: In common with all DVB descriptors the receiver shall always use the encoded descriptor length when parsing or skipping a descriptor. This is because DVB may in the future extend the syntax beyond that currently understood by the receiver.

5.3.2 Descriptor identification and location

Table 3 lists the descriptors defined or profiled in the present document, giving their intended placement and their values of descriptor_tag.

Table 3: Possible locations of descriptors

Descriptor	Tag value	Clause	NIT 1	NIT 2	BAT1	BAT 2	SDT	PMT 1	PMT 2	EITs	EITpf	RNT 1	RNT 2	RNT 3
metadata pointer descriptor	0x25	5.3.3	see note		see note		see note					see note	see note	see note
metadata descriptor	0x26	5.3.4							see note					
RAR over DVB stream descriptor	0x40	5.3.5												see note
RAR over IP descriptor	0x41	5.3.6												see note
RNT scan descriptor	0x42	5.3.7										see note		
default authority descriptor	0x73	6.3.3	see note	see note	see note	see note	see note							
related content descriptor	0x74	10.3							see note					
TVA_id descriptor	0x75	11.2									see note			
content identifier descriptor	0x76	12.1								see note	see note			

NOTE:
NIT 1: common (outer) descriptor loop of the NIT.
NIT 2: transport stream descriptor loop of the NIT.
BAT 1: common descriptor loop of the BAT.
BAT 2: transport stream descriptor loop of the BAT.
PMT 1: common descriptor loop of the PMT.
PMT 2: elementary stream descriptor loop of the PMT.
EITs: the descriptor loop of the EIT schedule.
EITpf: the descriptor loop of the EIT present/following.
RNT 1: common descriptor loop of the RNT.
RNT 2: resolution provider descriptor loop of the RNT.
RNT 3: CRID authority descriptor loop of the RNT.

5.3.3 Metadata pointer descriptor

5.3.3.1 Usage

The metadata pointer descriptor defines linkage to a metadata service. The scope of that linkage is either global, a DVB service or a CRID authority. The metadata service should carry information relevant to the scope of the linkage. For example, if a metadata service linkage is located in the descriptor loop of an SDT for a particular DVB service, it should carry information relating to that DVB service. A single metadata service may be referenced from a number of different locations and may therefore carry information relevant to a number of scopes. Table 4 defines the intended locations of the metadata pointer descriptor and gives example fragment types (see TS 102 822-3-2 [5], clause 4.3.1) relevant to that location.

Table 4: Permitted locations of the metadata pointer descriptor

Scope	Linkage location	Example metadata fragment types
Global	BAT, NIT, common descriptor loop of the RNT, resolution provider descriptor loop of the RNT	BroadcastEvent, Schedule, ServiceInformation, ProgramInformation, GroupInformation, Review, SegmentInformation, SegmentGroupInformation, PersonName, OrganizationName.
DVB Service	Service loop of the SDT	BroadcastEvent, Schedule, ServiceInformation.
CRID authority	CRID authority descriptor loop of the RNT	ProgramInformation, GroupInformation, Review, SegmentInformation, SegmentGroupInformation, PersonName, OrganizationName.

5.3.3.2 Semantics

This clause defines the use and additional semantics of fields of the metadata pointer descriptor. See ISO/IEC 13818-1 (amendment 1) [7], clause 2.6.58, for the format and basic semantics of this descriptor. This descriptor shall be used with the constraints given below when associating an entity in a DVB network with a metadata service delivering relevant TV-Anytime metadata.

metadata_application_format: This field shall have the same value as encoded in the metadata_application_format field of the metadata descriptor describing the target metadata service (see clause 5.3.4), that is the metadata descriptor with matching metadata_service_id carried in the PMT subtable indicated by this descriptor.

metadata_application_format_identifier: This field shall not be used.

metadata_format: This field shall have the same value as encoded in the metadata_format field of the metadata descriptor describing the target metadata service (see clause 5.3.4), that is the metadata descriptor with matching metadata_service_id carried in the PMT subtable indicated by this descriptor.

metadata_format_identifier: This field shall not be used.

metadata_service_id: This field shall be used to identify which metadata service this descriptor references. When that metadata service is delivered in a DVB transport stream, the target metadata service shall be described by a metadata descriptor with a matching value of metadata_service_id in the PMT subtable identified by this descriptor (see clause 5.3.4).

metadata_locator_record_flag: This field shall be set to "1" if this descriptor references a metadata service not delivered in a DVB transport stream. Otherwise, this field shall be set to "0".

EXAMPLE: If delivered over bi-directional IP according to TS 102 822-6-1 [16].

metadata_locator_record_length, metadata locator record: An option that may be used to point to an arbitrary URL not within a DVB network. If it is used the metadata locator record shall contain a compliant URI (see RFC 2396 [2]) which shall be encoded as described in clause 6.2. If this field contains a HTTP URL as specified in RFC 1945 [17], then that URL shall reference a metadata service that conforms to TS 102 822-6-1 [16].

MPEG_carriage_flags: A 2-bit field that shall be coded according to table 5. If the metadata_locator_record_flag is set to "1" then this field shall be set to 3. Otherwise, this field shall be set to either 0 or 1.

Table 5: MPEG_carriage_flags

Value	Semantics
0	Carried in the "actual" DVB transport stream
1	Carried in another DVB transport stream
2	Not used in the present document
3	None of the above - may be used if there is no relevant metadata carried on the DVB network. In this case the metadata locator record shall be present

Together the following three fields make up the normal triple that define a DVB service.

program_number: This field shall be set to the service_id in which the referenced metadata service can be found.

transport_stream_location: This field shall be set to the original_network_id. If not present the program number refers to the actual transport stream.

transport_stream_id: This field shall be set to the transport_stream_id. If set to 0x0000 this field shall be ignored and the combination of transport_stream_location and program_number define the original_network_id and service_id that uniquely identify the service in which the reference metadata can be found.

private_data_bytes: If the metadata_locator_record_flag is set to "0" the first bytes of this field shall contain the metadata descriptors extension structure as defined in table 9. When this is the case, the DVB_carriage_format field of the metadata descriptors extension structure shall be set to the same value as encoded in the DVB_carriage_format field of the metadata descriptors extension structure in the metadata descriptor describing the target metadata service (see clause 5.3.4), that is the metadata descriptor with matching metadata_service_id carried in the PMT subtable indicated by this descriptor

5.3.4 Metadata descriptor

5.3.4.1 Usage

MPEG-2 part 1 amendment 1: Carriage of metadata over ISO/IEC 13818-1 [7] (amendment 1) streams defines the metadata descriptor (see clause 2.6.60 of ISO/IEC 13818-1 [7]). This descriptor with the constraints given below shall be carried in PMT subtables in the descriptor loop of an elementary stream that carries the DSI of an MHP object carousel delivering one or more metadata services (see clause 9.2). It is used to describe the format and download parameters of the metadata service carried on that elementary stream.

5.3.4.2 Semantics

This clause defines the use and additional semantics of fields of the metadata descriptor. See ISO/IEC 13818-1 [7] (amendment 1), clause 2.6.60, for the format and basic semantics of this descriptor.

metadata_application_format: This field shall have the value 0x0100. The use of this value signifies that the metadata service contains TVA metadata as profiled according to DVB (see clause 8).

metadata_application_format_identifier: This field shall not be used.

metadata_format: This field indicates the coding format of the metadata. It shall be encoded according to table 6. When the metadata_locator_record_flag is set to "1" this field shall be set to 0x3F.

Table 6: metadata_format

Value	Semantics
0x00 to 0x3E	Not used in the present document
0x3F	Defined by metadata application format
0x40 to 0xEF	User defined
0xF0	The encoding and encapsulation format as defined in clauses 9.3 and 9.4
0xF1 to 0xF7	DVB Reserved
0xF8 to 0xFE	User defined
0xFF	Not used in the present document

metadata_format_identifier: This field shall not be used.

metadata_service_ID: This field indicates the metadata service to which this descriptor applies. This value shall be unique within the scope of the PMT subtable in which this descriptor is carried.

decoder_config_flags: This field indicates the location of the DVB-TVA-init message (see clause 9.4.2.1).

If the DVB_carriage_format field is set to 0 then this field shall have the value "000" if the DVB-TVA-init message is carried in a BIOP::FileMessage object referenced by the default file name, or it shall have the value "011" if the DVB-TVA-init message is carried in a BIOP::FileMessage object referenced by the dec_config_identification_record_bytes. No other values of decoder_config_flags are permitted.

For other values of DVB_carriage_format the semantics of this field are undefined.

DSM-CC_flag: This field shall be set to "1" if the metadata is carried in an ISO/IEC 13818-6 [12] data or object carousel. Otherwise it shall be set to "0".

service_identification_record_byte: This field is part of a sequence that conveys the metadata service location string.

If the DVB_carriage_format field is set to 0 then the following rules apply:

- This string is the location of the metadata service within the object carousel associated with this descriptor. The location is described as a file path that references the directory within which the metadata service's containers can be found.

If the service_identification_record_byte sequence is a string with length greater than zero, then contents of this field define the service location string (mds_explicit_path). If this sequence is a zero-length string, then the default value of the metadata service location string shall be used (mds_default_path). The format of the metadata service location string is defined in table 7.

Table 7: Metadata service location string

```
metadata_service_location_string    = mds_explicit_path | mds_default_path
mds_explicit_path                   = "/" path_segments
mds_default_path                    = "/" metadata_service_ID_string
metadata_service_ID_string          = hex_string
hex_string                          = 4*hex
hex                                 = digit | "A" | "B" | "C" | "D" | "E" | "F" |
                                        "a" | "b" | "c" | "d" | "e" | "f"
digit                               = "0" | "1" | "2" | "3" | "4" | "5" | "6" |
                                        "7" | "8" | "9"
```

The format for path_segments is defined in RFC 2396 [2].

The format for metadata_service_ID_string is the value of metadata_service_ID carried in this descriptor encoded as a hex_string.

Below are example metadata service location paths, the first two examples are explicitly encoded in the service_identification_record_byte field, the last example is a default metadata service location path, defined by a zero-length service_identification_record_byte sequence:

```
"/metadata/service1"
"/"
"/579A"
```

For other values of DVB_carriage_format the semantics of this field are undefined.

dec_config_identification_record_byte: This field is part of a sequence that conveys the location of the decoder configuration.

If the DVB_carriage_format field is set to 0 then the following rules apply:

- The location is described as a file path that references the BIOP::FileMessage object within the metadata service referenced by this descriptor, which carries the DVB-TVA-init message (see clause 9.4.2.1). The format of this field is defined below:

  - If the dec_config_identification_record_byte sequence is a string with length greater than zero, then contents of this field define the location of the DVB-TVA-init message (the dti_explicit_path). If this sequence is a zero-length string, then the default value of the DVB-TVA-init message location shall be used (the dti_default_path). The format of the DVB-TVA-init message location is defined in table 8.

Table 8: DVB TVA init message location

```
DVB_TVA_init_msg_location       = dti_explicit_path | dti_default_path
dti_explicit_path               = "/" path_segments
dti_default_path                = "/" metadata_service_ID_string
metadata_service_ID_string      = metadata_service_location_string
```

The format for path_segments is defined in TS 102 812 [10], clause 14.1.

The format for the metadata service location string is conveyed by the service_identification_record_bytes. See semantics for that field, above.

For other values of DVB_carriage_format the semantics of this field are undefined.

private_data_bytes: This field shall contain the metadata_descriptors_extension structure as defined in table 9. Otherwise, the contents of this field is not defined.

Table 9: Metadata descriptors extension

Name	Number of bits	Identifier
metadata_descriptors_extension() {		
DVB_carriage_format	4	uimsbf
reserved	4	uimsbf
for (i=0; i<N; i++) {		
user_data_byte	8	bslbf
}		
}		

DVB_carriage_format: This field defines the precise delivery format used for this metadata service. It shall be encoded according to table 10.

Table 10: DVB_carriage_format

Value	Semantics
0	Delivery as defined in clause 9.2
1 to 7	DVB reserved
8 to 15	User defined

5.3.5 RAR over DVB stream descriptor

The RAR over DVB stream descriptor encapsulates a RAR for a CRID authority whose data can be found on a DVB stream. This descriptor is permitted in the CRID authority descriptor loop of a RNT section. The format for this descriptor is defined in table 11.

Table 11: RAR over DVB stream descriptor

Syntax	Number of bits	Identifier
RAR_over_DVB_stream_descriptor() {		
descriptor_tag	8	uimsbf
descriptor_length	8	uimsbf
first_valid_date	40	bslbf
last_valid_date	40	bslbf
weighting	6	uimsbf
complete_flag	1	bslbf
scheduled_flag	1	bslbf
transport_stream_id	16	uimsbf
original_network_id	16	uimsbf
service_id	16	uimsbf
component_tag	8	uimsbf
if (scheduled_flag == 1) {		
download_start_time	40	bslbf
download_period_duration	8	uimsbf
download_cycle_time	8	uimsbf
}		
}		

descriptor_tag: This field shall be set to 0x40.

descriptor_length: This field shall be set to the number of bytes in this descriptor immediately following this field.

first_valid_date: The first date when this CRID authority reference can be used, using UTC as the time reference.

last_valid_date: The first date when this CRID authority reference cannot be used, using UTC as the time reference.

NOTE: The reason for providing start and end dates for resolution is so that resolution providers can move their resolution URLs and be sure all PDRs have switched to the new URL once the last valid date of the old resolution record has passed.

weighting: The weighting field is a hint to the PDR as to the order to try multiple records for a single CRID authority from the same resolution provider. The largest weighting number shall be assigned to the URL that should be tried first. The weighting field is only used to provide ordering between resolution provider records for the same combination of resolution provider and CRID authority name and not for ordering one provider over another.

complete_flag: This flag indicates if the referenced CRI data is complete. It shall be set to "1" if the referenced CRI data is complete, otherwise it shall be set to "0". See clause 7.2.3 for details on the use of this field.

scheduled_flag: This flag indicates if the referenced CRI data is delivered at a scheduled time, rather than being delivered continuously. It shall be set to "1" if the referenced CRI data is scheduled, or "0" if the referenced CRI data is delivered continuously.

transport_stream_id: This field shall be set to the transport_stream_id of the DVB service in which the referenced CRI is carried. If set to 0x0000 then this field shall be ignored and the DVB service shall be uniquely identified by a combination of original_network_id and service_id.

original_network_id: This field shall be set to the original_network_id of the DVB service in which the referenced CRI is carried.

service_id: This field shall be set to the service_id of the DVB service in which the referenced CRI is carried.

component_tag: This field identifies the elementary stream on which the referenced CRI is carried. The stream_identifier_descriptor is mandatory for all components referenced by this descriptor (see EN 300 468 [1], clause 6.2.34).

download_start_time: This field describes the date and time at which the CRI service will start to be available. This field shall be encoded as MJD and UTC (see EN 300 468 [1], annex C). This 40-bit field is coded as 16 bits giving the 16 LSBs of MJD followed by 24 bits coded as 6 digits in the 4-bit BCD.

download_period_duration: This field describes the length of time from the start time during which the CRI service will be available. This field shall be encoded as a count of 6 minute periods.

download_cycle_time: This field shall be set to the minimum time required for one complete repetition of all data in the CRI service, measured in minutes.

5.3.6 RAR over IP descriptor

The RAR over IP descriptor encapsulates a RAR for a CRID authority whose data can be found via an IP network. This descriptor is permitted in the CRID authority descriptor loop of a RNT section. The syntax for this descriptor is defined by table 12.

Table 12: RAR over IP descriptor

Syntax	Number of bits	Identifier
RAR_over_IP_descriptor() {		
descriptor_tag	8	Uimsbf
descriptor_length	8	uimsbf
first_valid_date	40	bslbf
last_valid_date	40	bslbf
weighting	6	uimsbf
complete_flag	1	bslbf
reserved	1	bslbf
url_length	8	uimsbf
For (i=0;i < url_length< i++) {	8	uimsbf
url_char	8	uimsbf
}		
}		

descriptor_tag: This field shall be set to 0x41.

descriptor_length: This field shall be set to the number of bytes in this descriptor immediately following this field.

first_valid_date: The first date when this CRID authority reference can be used, using Universal Time Coordinates (UTC) as the time reference.

last_valid_date: The first date when this CRID authority reference cannot be used, using UTC as the time reference.

NOTE: The reason for providing start and end dates for resolution is so that resolution providers can move their resolution URLs and be sure all PDRs have switched to the new URL once the last valid date of the old resolution record has passed.

weighting: The weighting field is a hint to the PDR as to the order to try multiple records for a single CRID authority from the same resolution provider. The largest weighting number shall be assigned to the URL that should be tried first. The weighting field is only used to provide ordering between resolution provider records for the same combination of resolution provider and CRID authority name and not for ordering one provider over another.

complete_flag: This flag indicates if the referenced CRI data is complete. It shall be set to "1" if the referenced CRI data is complete, otherwise it shall be set to "0". See clause 7.2.3 for details on the use of this field.

reserved: Reserved bits shall be set to "1".

url_length: The length of the URL.

url_char: The URL describing the location where CRIDs belonging to this CRID authority can be resolved. This field shall be encoded according to clause 6.2.

5.3.7 RNT scan descriptor

The RNT scan descriptor carries references to transport streams that carrying RNTs. This descriptor is permitted in the common descriptor loop of an RNT section, see clause 5.2.1 for details on the use of this descriptor. The format for the RNT scan descriptor is defined by table 13.

Table 13: RNT scan descriptor

Syntax	Number of bits	Identifier
RNT_scan_descriptor() {		
descriptor_tag	8	uimsbf
descriptor_length	8	uimsbf
for (i=0; i<N; i++) {		
transport_stream_id	16	uimsbf
original_network_id	16	uimsbf
scan_weighting	8	uimsbf
}		
}		

descriptor_tag: This field shall be set to 0x42.

descriptor_length: This field shall be set to the number of bytes in this descriptor immediately following this field.

transport_stream_id: This field carries the value of transport stream id of the transport stream referenced by this entry.

original_network_id: This field carries the value of original network id of the transport stream referenced by this entry.

scan_weighting: This field defines the intended order of tuning to other transport streams to acquire RNTs. An entry with a larger weighting value should be inserted before entries with smaller weightings.

6 CRIDs and other URIs in DVB networks

6.1 Introduction

This clause defines how CRIDs and other URIs shall be encoded in the structures defined in the present document. It also defines a mechanism for defining a default authority and associated scoping rules for the purpose of improving the compaction of CRIDs.

6.2 Encoding of URI strings and the use of non-Latin characters

The URI format (see RFC 2396 [2]) consists of a sequence of a limited range of Latin characters plus a limited number of graphical characters (e.g. "@", "=", etc., but not including a space character). In order for non-Latin characters to be used in URIs, a standard mapping from those non-Latin characters is defined.

All characters not within the range of characters allowed in a URI must be encoded into UTF8 and included in the URI as a sequence of escaped octets. An escaped octet is encoded as a character triplet, consisting of the percent character "%" followed by the two hexadecimal digits representing the octet code.

The syntax of the CRID is URI compliant and is defined in TS 102 822-4 [6]. Its format is as follows:

- crid://<CRIDauthority>/<data>

An example being:

- crid://company.com/foobar

CRIDs are insensitive to the case of characters.

Where lexicographical ordering is applied to CRIDs in the present document, it shall be applied after characters not in the allowed range are converted into sequences of escaped octets.

6.3 Default authority and abbreviated CRIDs

6.3.1 Abbreviated CRID rules

In certain situations described in the present document a CRID string may use the following abbreviated forms. These reduce the overhead of a CRID string by leaving out information that can be inferred from the location of the CRID entry.

Firstly, the characters "crid://" may be omitted from the start of the string so that the string starts with the first character of the CRID authority. So the example CRID:

- crid://company.com/foobar

may be encoded as:

- company.com/foobar

Additionally, within the scope of the definition of a default authority (see clause 6.3.2), the CRID authority part of the string may also be omitted if the CRID's authority matches the current value of default authority. In this case the string starts with the delimiter between CRID authority and data parts of the CRID (i.e. "/"). Therefore, the example CRID:

- crid://company.com/foobar

may be encoded as:

- /foobar

6.3.2 Scope of a default authority definition

A default authority is defined by the presence of a default authority descriptor. The purpose of the default authority is to allow a CRID reference within the scope of such a definition to leave out the protocol and authority parts of a CRID URI, if the CRID authority part of that CRID reference is the same as the defined default authority.

The scope of a particular value of default authority is defined by the location of the default authority descriptor. A value of default authority defined in a scope overrides any value of default authority already defined for a wider, enclosing scope. See table 14 for definitions of the permitted locations of the default authority descriptor and which scope override which others.

Table 14: Permitted locations of default authority descriptor

Default authority descriptor location	Scope of definition	Scopes this definition overrides
First descriptor loop of NIT	network	none
Transport stream descriptor loop of NIT	transport stream	bouquet or network
First descriptor loop of BAT	bouquet	none
Transport stream descriptor loop of BAT	transport stream	bouquet or network
Service descriptor loop of SDT	service	transport stream, bouquet or network

The effect of defining a default authority in a BAT that conflicts with a definition of equivalent scope in a NIT is not defined by the present document.

EXAMPLE: If a default authority is defined at the scope of a network, this can be overridden for a single service on that transport stream by the inclusion of a default authority descriptor in the service loop of an SDT on that transport stream.

6.3.3 Default authority descriptor

The default authority descriptor is permitted in either the first of second descriptor loop of a NIT or BAT, or the service descriptor loop of a SDT. The syntax of this descriptor is defined in table 15.

Table 15: Default authority descriptor

Syntax	Number of bits	Identifier
default_authority_descriptor() {		
descriptor_tag	8	uimsbf
descriptor_length	8	uimsbf
for (i=0; i < descriptor length; i++) {		
default_authority_byte	8	uimsbf
}		
}		

descriptor_tag: This field shall be set to 0x73.

descriptor_length: This field shall be set to the number of bytes in this descriptor immediately following this field.

default_authority_byte: This field forms part of a sequence that is the default authority for this scope. The encoding of this field shall follow the rules defined in clause 5.2.2 for resolution_provider_name_byte. See also TS 102 822-4 [6], clause 6.

6.4 DVB locator extensions

The present document extends the DVB locator format as defined in TS 102 812 [10], clause 14.1. The syntax is extended as defined by table 16.

Table 16: DVB locator extension

```
dvb_event_constraint    = event_id_mode | tva_id_mode | time_constraint
event_id_mode           = ";" event_id [ time_constraint ]
tva_id_mode             = ";;" TVA_id [ time_constraint ]
time_constraint         = "@" time_duration
TVA_id                  = hex_string
time_duration           = string
hex_string              = 4*hex
hex                     = digit | "A" | "B" | "C" | "D" | "E" | "F" | "a" | "b" | "c" |
                            "d" | "e" | "f"
digit                   = "0" | "1" | "2" | "3" | "4" | "5" | "6" | "7" | "8" | "9"
```

The time_duration string shall be formatted according to the extended format defined in ISO 8601 [13], clause 5.5.4.3. Note that "/" must be replaced by two "-" characters, as in the example below.

EXAMPLE: CCYY-mm-ddThh:mm[:ss]Z--PThhHmmM[ssS].

When referencing a DVB service the DVB locator shall be restricted like so:

- dvb://<original_network>.[<transport_stream>].<service_id>

When referencing an event the DVB locator shall be restricted to any of the following:

- dvb://<original_network.>[<transport_stream>].<service_id>;<event_id> [@time_duration]
- dvb://<original_network.>[<transport_stream>].<service_id>;;<TVA_id> [@time_duration]
- dvb://<original_network.>[<transport_stream>].<service_id>@time_duration

A metadata fragment may contain a DVB locator referencing a file in an object carousel (see for example the Logo element in clause 8.6). When this occurs and the file is delivered in the same object carousel as the metadata service delivering the metadata fragment, the following syntax may be used for the DVB locator:

- dvb:/path_segments

This path shall be interpreted as being an absolute path, that is one that is relative to the ServiceGateway for the object carousel carrying the metadata service. The format of this shall follow the restrictions defined in TS 102 812 [10], clause 14.1.4.

If a metadata fragment references a file delivered in a different object carousel to the metadata service delivering that metadata fragment, the following syntax shall be used for the DVB locator:

- dvb://<original_network_id>.[<transport_stream_id>].<service_id>.<component_tag>{&<component_tag>}/path_segments.

7 Content resolution

7.1 Introduction

The purpose of content referencing is to allow acquisition of a specific item of content or a group of items of content. For example, if a consumer sees a promotion on TV saying, "there will be a new series on 'Foxes in the cold' around Christmas", he may want to instruct his Personal Digital Recorder (PDR) to record the whole series. However the actual time and channel of airing of the episodes might be unknown to the PDR. In fact, the broadcaster may not know yet, either. However, at the time when the viewer sees the promotion he will want to make sure that he does not miss the opportunity to acquire the content.

The ability to refer to content (in this example a series of programs) independent of its time or location will provide this capability desired by the consumer. Whether that location is on a particular broadcast channel on some date and time, or on a file server connected to Internet, or wherever.

In the current example of a series, the PDR system would be provided with a reference for that series. In due time, the information required to link this reference to the individual episodes would be supplied to the PDR. Subsequently, a specific location (channel, date and time) would be provided for each episode so that the PDR would then be able to acquire all of them.

Figure 4 demonstrates the purpose of content referencing: to provide the ability to refer to content independent of its location; and to provide the ability to subsequently resolve such a reference into one or more locations where the content can be obtained.

Location Resolution is not a once-only operation. Receivers may need to re-resolve CRIDs at intervals before and during recordings in response to changes in the content referencing information.

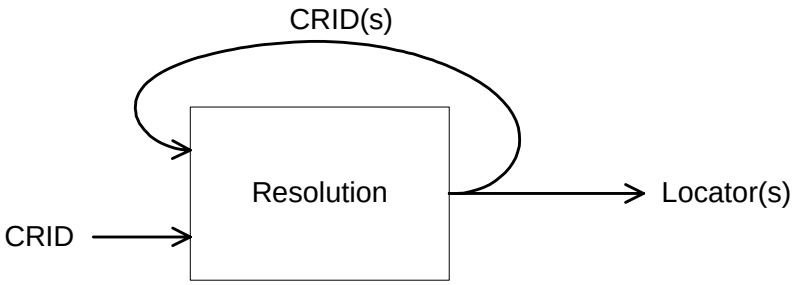

Figure 4: The location resolution process

The key concept in content referencing is the separation of the reference to a content item - the CRID - from the information needed to actually retrieve the content item - the locator. The separation provided by the CRID enables a one-to-many mapping between content references and the locations of the deliverables. From a system perspective, content referencing and resolution lies between search and selection and actually acquiring the content. From the content referencing perspective, search and selection yields a CRID, which is resolved into either a number of CRIDs or a number of locators (the number may be one). A full discussion of content referencing is beyond the scope of the present document; rather it is the intention here to show how content referencing fits into the overall system. In the examples below, the syntax of a CRID and the syntax of a locator are employed. The syntax of a CRID is:

- CRID://<CRIDauthority>/<data>

Where <CRIDauthority> takes the form:

- <DNS name><name_extension>

<DNS name> is a registered Internet domain name. (See RFC 1591 [11] for DNS name registration.) The <DNS name> is case insensitive and must be a fully qualified name according to the rules defined by RFC 1591 [11].

<name_extension> is an optional string (beginning with a ";" character) to enable multiple authorities to use the same DNS name. All <name_extension> elements which share the same <DNS name> must be unique. The <name_extension> section is case insensitive.

The syntax of the locator is:

- <transport mechanism>:<transport system specific>

The content referencing mechanism employs two key elements. The first is the resolution provider notification table that maps the CRID authority that issued the CRID to the Resolution Service Provider. The second is the actual Content Resolution Information (CRI), which maps a CRID to another CRID or to a location. The CRI may also contain information to link a locator to metadata describing that instance. See TS 102 822-4 [6] for a more detailed explanation of the concepts and tables involved.

7.2 Resolving CRIDs in a DVB network

7.2.1 DVB transport stream resolution handler

TS 102 822-4 [6], clause 10.1.1, describes a conceptual modular resolution system. The modular resolution system has different resolution handler modules, each handling a different protocol over which location resolving can occur. This clause defines the functionality of a DVB transport stream resolution handler module for the purpose of defining CRID resolving on DVB transport streams (see figure 5).

The use of other types of resolution handler is supported. For example, a PDR may additionally support CRID resolution over the internet via a return path connection.

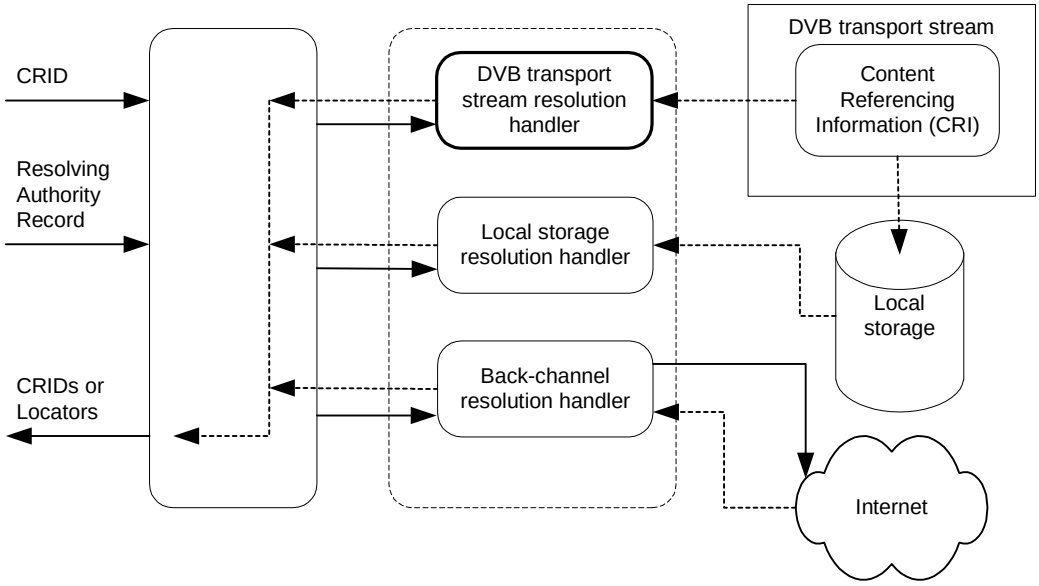

Figure 5: Modular location resolving system

7.2.2 CRI data sets

CRI data consists of the information required to resolve CRIDs into its result (i.e. CRIDs or locators). On DVB transport streams, CRI data for a particular CRID authority is organized into one or more *sets* of CRI data (or "CRI data sets"). A CRI data set is the CRI data for a CRID authority in a particular content resolution service or internet location. See figure 6 for an example of different CRI data sets. References to CRI data are carried in Resolving Authority Records (RARs) which, on a DVB transport stream, are embodied in RAR descriptors (see clauses 5.3.5 and 5.3.6). These descriptors are carried in the resolution provider notification table (see clause 5.2.2).

Each content resolution service shall only contain CRI provided by one resolution provider. A content resolution service may contain CRI regarding more than one CRID authority (i.e. more than one CRI data set).

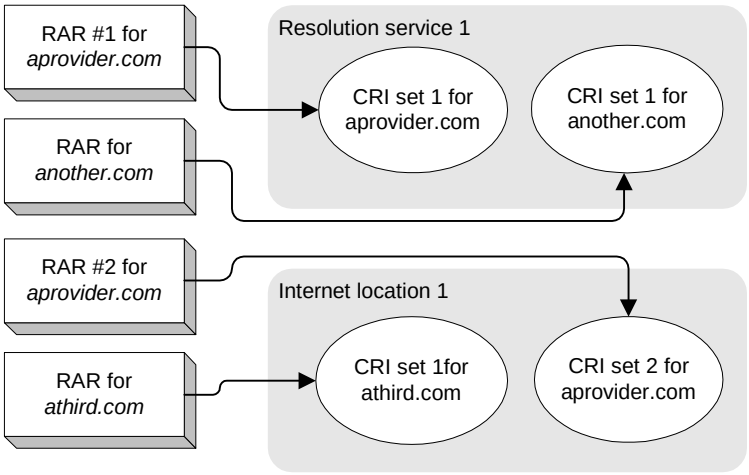

Figure 6: Sets of CRI data in different locations

7.2.3 Complete and incomplete CRI data sets

Sets of CRI data may be signalled as either being complete or incomplete. A complete set of CRI data contains information on all CRIDs provided by a particular resolution provider for a particular CRID authority.

Where a complete set of CRI data is supplied by a resolution provider for a CRID authority, incomplete sets of CRI data can be used in addition, for example to provide resolution information for CRIDs relevant to the local transport stream. Because of this a receiver may optimize location resolving by accessing an incomplete CRI data set in the current transport stream before falling back to accessing a complete CRI data set on another transport stream.

CRI data provided by a resolution provider for a CRID authority may alternatively be distributed between several incomplete sets of CRI data, with no complete set being available. In this case, a receiver resolving a CRID should access all CRI data sets available from a resolution provider for the relevant CRID authority before returning an error.

If a receiver is configured to access CRI provided by more than one resolution provider it should check both for CRI relating to a particular CRID authority.

NOTE: Failure to resolve a CRID may be due to resolving information not being available for that CRID, or it may be because of a network failure. Therefore, a receiver may wish to continue to attempt to resolve such a CRID over a period of time not determined by the present document.

7.3 Delivery of content referencing information

7.3.1 Container

7.3.1.1 Description

The container is the means by which all CRI structures shall be carried in a transport stream. A container shall contain one or more CRI structures. Each container is distinguished by a unique identifier, which is the container_id. Containers are mapped on to a table of MPEG 2 TS private sections, the headers of which carry the container_id. The container carrying the cri_index structure, which is the first structure required by the receiver, shall have its container_id set to 0x0000.

7.3.1.2 Classifications of CRI structures and containers

There are two classifications of CRI structures, CRI index structures and CRI results structures.

CRI Index Structures: cri_index, cri_prepend_index cri_leaf_index, data_repository.

CRI Results Structures: results_list, result_data, data_repository, services.

A container carrying index structures is termed an index container, whilst a container carrying results structures termed a results container. A container can carry either a mixture of classifications of structure, or just a single classification. Therefore, a container can be both an index container and a results container. Figure 7 shows an example configuration suitable for a large set of CRI data, with CRI index structures separated into different containers than CRI results structures. Figure 8 shows an example configuration suitable for a small set of CRI data, with CRI index structures carried in the same container as CRI results structures.

NOTE: The types of structures used in each example differ slightly because a cri_leaf_index structure does not require a results_list structure if it uses the local_result_locator.

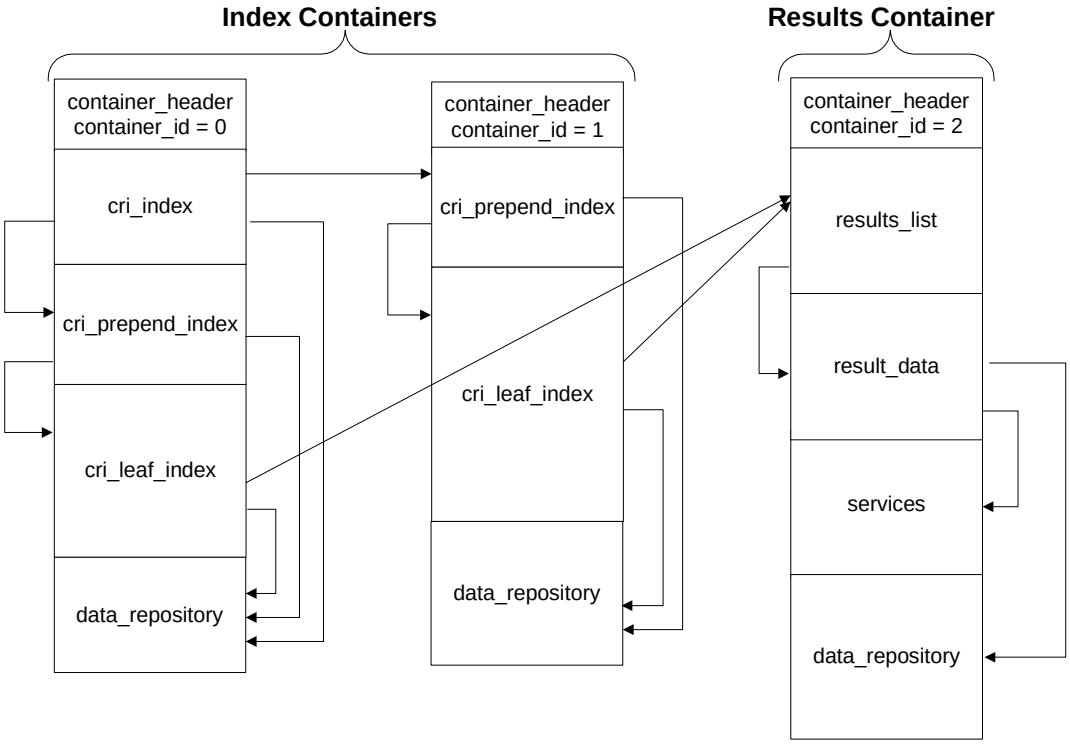

Figure 7: Example container structure for a large CRI data set

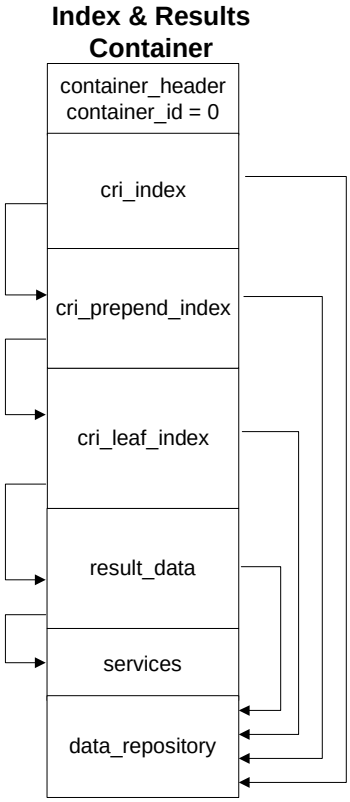

Figure 8: Example container structure for a small CRI data set

7.3.1.3 Container format

Entries within the container_header shall be ordered in ascending cri_structure_type and cri_structure_id. This enables a device to efficiently locate a particular structure. The maximum size of a container shall be 65 536 bytes. The format of the container is defined by table 17.

Table 17: Container

Syntax	Number of bits	Identifier
container() {		
container_header {		
num_cri_structures	8	uimsbf
for(j=0; j<num_cri_structures; j++) {		
cri_structure_type	8	uimsbf
cri_structure_id	8	uimsbf
cri_structure_ptr	24	uimsbf
cri_structure_length	24	uimsbf
}		
}		
for (j=0; j<num_cri_structures; j++) {		
cri_structure()		
}		
}		

num_cri_structures: This field specifies the number of structures in this container.

cri_structure_type: This field identifies the type of structure this entry relates to, it shall be encoded according to table 18.

Table 18: CRI structure type and structure id

cri_structure_type value	cri_structure_id value	Description
0x00	not defined	reserved
0x01	0x00	results_list
0x02	0x00	data_repository
0x03	not defined	reserved
0x04	0x00 - 0xFF	cri_index
0x05	0x00 - 0xFF	cri_prepend_index or cri_leaf_index
0x06 to 0x07	not defined	reserved
0x08	0x00	result_data
0x09	0x00	services
0x0A to 0xEF	not defined	DVB Reserved
0xF0 to 0xFF	not defined	User Private

cri_structure_id: An 8 bit field used to distinguish between multiple instances of the same structure in a single container. This field shall be encoded according to table 18. Where only one value of cri_structure_id is allowed for a particular value of cri_structure_type, this indicates that only one occurrence of that cri_structure_type is allowed in a container.

cri_structure_ptr: A 24 bit field giving the offset in bytes from the start of this container to the first byte of the CRI structure.

cri_structure_length: A 24 bit field which indicates the length in bytes of the CRI structure pointed to by cri_structure_ptr.

7.3.1.4 Container section

For delivery, a container is carried in a container_subtable split into a sequence of one or more blocks of container data. Each block is carried as a section, the numbering of each section corresponding to the position of the container data block in the sequence. The clauses for a container form a single container_subtable, distinguished by container_id. Before a container can be parsed, all sections forming a single container_subtable must be acquired. The container is reconstructed by appending container data blocks together in the order defined by the section number and reversing compression if this has been applied.

The container section is derived from the standard private_section syntax as defined in ISO/IEC 13818-1 [8]. A container is carried by one or more container sections. Every container section shall carry 4 084 bytes of container data, except the container_section with section_number equal to last_section_number, which shall carry however many bytes remain of the container subtable. The syntax of the container section is defined by table 19.

Table 19: Container section

Syntax	Number of bits	Identifier
container_section() {		
table_id	8	uimsbf
section_syntax_indicator	1	bslbf
private_indicator	1	bslbf
reserved	2	bslbf
private_section_length	12	uimsbf
container_id	16	uimsbf
reserved	2	bslbf
version_number	5	uimsbf
current_next_indicator	1	bslbf
section_number	8	uimsbf
last_section_number	8	uimsbf
container_data()		
CRC32	32	uimsbf
}		

table_id: This field shall be set to 0x75.

section_syntax_indicator: This shall be set to "1" to indicate that the private section follows the generic section syntax.

private_indicator: This flag shall be set to "1".

private_section_length: The number of remaining bytes in the private section immediately following the private_section_length field up to the end of the private_section.

container_id: This shall contain the container_id of the container carried by the table this section is part of.

version_number: The version of the table. The version shall be incremented by 1 modulo 32 when there is a change in the information.

current_next_indicator: This field shall be set to "1" to indicate that the section is currently valid.

section_number: This 8-bit field specifies the number of the private section. This section_number will be incremented by 1 with each additional section in the table.

last_section_number: This specifies the number of the last section making up this table.

container_data(): A sequence of bytes making up a portion of a compression_wrapper.

CRC32: This field contains the CRC value that gives a zero output of the registers in the decoder defined in EN 300 468 [1] after processing the entire private section.

7.3.1.5 Compression wrapper

The compression_wrapper allows a container to be carried in a compressed or uncompressed format. The syntax of the compression_wrapper is defined by table 20.

Table 20: Compression_wrapper

Syntax	Number of bits	Identifier
`compression_wrapper() {`		
`  compression_method`	8	uimsbf
`  if (compression_method == 0x00) {`		
`  container()`		
`  } else if (compression_method == 0x01) {`		
`  original_size`	24	uimsbf
`  compression_structure()`	N x 8	
`  }`		

compression_method: This field shall be encoded according to table 21.

Table 21: Compression method

Value	Meaning
0x00	The container is not compressed
0x01	The container is carried in a Zlib stream as defined in RFC 1950 [18]
0x02 to 0x7F	DVB reserved
0x80 to 0xFF	User private

container(): See clause 7.3.1.3 for the definition of the container structure.

original_size: This field indicates the size in bytes of the container prior to compression.

compression_structure: This shall contain a Zlib stream as defined in RFC 1950 [18]. When present, the Zlib stream shall have its compression method nibble set to "1000", indicating use of the Deflate compression algorithm as specified in RFC 1951 [15].

7.3.2 CRI results structures

7.3.2.1 Description

CRI results structures shall be used to carry Content Referencing Information on DVB transport streams. Individual CRID results conveyed within these structures may be identified and located using the CRI indexing structures. Every CRID result has a handle_value that is used by index CRI structures to refer to a result. The results_list maps those handles to the relevant CRID result data, which is carried in the result_data structure.

Of the following structures a results container shall contain those that are mandatory and may contain those that are optional:

- results_list (Mandatory if results in this container are referenced by a cri_sub_index in another container);
- result_data (Mandatory);
- services (Mandatory if DVB binary locators are used).

7.3.2.2 Results_list

The results_list CRI structure associates results data with a handle_value. Entries within the results_list structure shall be in order of ascending handle_value.

There shall be one instance of the results_list CRI structure within a results container if any CRI results in that container are referred to from other containers. The syntax of the results_list structure is defined by table 22.

Table 22: Results_list

Syntax	Number of bits	Identifier
`results_list () {`		
`  for(j=0; j<NumCRIDs; j++) {`		
`  handle_value`	16	uimsbf
`  result_ptr`	16	uimsbf
`  }`		
`}`		

handle_value: This field uniquely identifies a CRID result within the current container, this is for the purpose of referencing the CRID result from a cri_leaf_index. The value assigned to a CRID result should be persistent for the life of that CRID result so long as it is transmitted in the same container. If a CRID result moves from one container to another, the handle_value must then be unique within the new container. All entries shall be ordered by ascending value of handle_value.

result_ptr: The offset, in bytes, from the first byte of the result_data structure of the current container to the first byte of the relevant content referencing information.

7.3.2.3 Result_data

7.3.2.3.1 Usage

The result_data CRI structure carries the CRID results data. There shall be one instance of the result_data CRI structure within a results container.

7.3.2.3.2 Syntax

The syntax of the result_data structure is defined by table 23.

Table 23: Result_data

Syntax	Number of bits	Identifier
result_data () {		
year_offset	16	uimsbf
for(j=0; j<Table_size; j += sizeof(Result)) {		
status	2	uimsbf
acquisition_flag	1	bslbf
re_resolve_flag	1	bslbf
result_type	2	bslbf
imi_flag	1	bslbf
reserved	1	bslbf
if(status=="00") {		
num_results	8	uimsbf
for(r=0; r<num_results; r++) {		
if(result_type == "00") {		
CRID_prepend_ptr	16	uimsbf
result_CRID_data_ptr	16	uimsbf
}		
else if(result_type == "01") {		
dvb_binary_locator()		
}		
else if(result_type == "10") {		
locator_format	4	uimsbf
locator_length	12	uimsbf
if (locator_format == 0x01) {		
dvb_binary_locator()		
}		
else {		
for(j=0; j<locator_length; j++) {		
locator_byte	8	uimsbf
}		
}		
}		
else {		
DVB_reserved_length	16	uimsbf
for (i=0; i<DVB_reserved_length; i++) {		
DVB_reserved_byte	8	uimsbf
}		
}		
if(result_type != "00" && imi_flag == "1") {		
Imi_prepend_ptr	16	uimsbf
result_imi_data_ptr	16	uimsbf
}		
}		
}		
if(status == "01" || (status == "00" && re_resolve_flag == "1") {		
reserved	7	bslbf
reresolve_date	9	uimsbf
reresolve_time	16	uimsbf
}		
}		
}		

year_offset: The year relative to which date values in this structure shall be calculated. This field shall be encoded as the binary value of the year, for example "2003" would be encoded as 0x07D3.

status: A two bit field used to indicate the current status of the CRID result. This field shall be encoded according to table 24.

Table 24: CRID resolution status

status	Meaning
"00"	Valid result, CRID or locators follow
"01"	CRID is not yet resolvable
"10" to "11"	DVB Reserved

It is not possible to encode a result with status "discard CRID", as defined by TS 102 822-4 [6], clause 12.2. This value is assumed implicitly if a CRID has no entry in a relevant complete set of CRI data, or in all relevant CRI data sets should a complete set not be available. See clauses 7.2.2 and 7.2.3.

acquisition_flag: A flag to indicate what type of acquisition is required for this CRID. This field shall be encoded according to table 25.

Table 25: Acquisition flag

acquisition flag	Meaning
"0"	Acquire all items this CRID resolves into
"1"	Acquire any one of the items this CRID resolves into

re_resolve_flag: A flag to indicate whether resolution of the CRID is complete. This field shall be encoded according to table 26.

Table 26: Re-resolve flag

re_resolve_flag	Meaning
"0"	Resolution is complete
"1"	Resolution is incomplete, more resolution information to follow

result_type: Indicates whether this is a group or leaf CRID and how they are encoded. This field shall be encoded according to table 27.

Table 27: Result type

result_type	Meaning
"00"	Result is a list of CRIDs, from one or more CRID authorities
"01"	Result is a list of DVB binary encoded locators
"10"	Result is a list of locators, in mixed format (see table 29)
"11"	DVB Reserved

NOTE 1: If a CRID resolves to other CRIDs, rather than locators, it is possible that the CRID relates to a single programmes rather than a collection of programmes. This may occur, for example, when the options for capturing that single programme are too complex to be described as a list of locators. Therefore, where metadata is available, a receiver should try to access ProgramInformation for a CRID that resolves to other CRIDs, if accessing GroupInformation does not succeed. See clause 8.3.

imi_flag: A flag to indicate whether Instance Metadata Identifiers (IMIs) are provided. This shall not be set to "1" if result_type is equal to "00". This field shall be encoded according to table 28.

Table 28: IMI flag

imi_flag	Meaning
"0"	There are no IMIs for this result
"1"	IMIs are provided for this result

num_results: The number of resolution results for this entry.

CRID_prepend_ptr: The offset in bytes within the string data repository, where the CRID prepend string can be found. The first seven letters ("crid://") shall be omitted from the CRID prepend string.

result_CRID_data_ptr: The offset, in bytes, from the first bytes of the data repository within the current container, to the first byte of the result CRID data string. Concatenating the string "crid://" with the prepend string and the result CRID data string shall result in a valid CRID.

- crid://<CRID prepend string><result CRID data string>

locator_format: The format of the locator bytes. This field shall be encoded according to table 29.

Table 29: Locator format

Locator format	Meaning
0x00	URI compliant string followed by a byte of value 0x00
0x01	DVB binary locator (see clause 7.3.2.3.3)
0x02 to 0x0B	Reserved
0x0C to 0x0F	Private Use

locator_length: The number of bytes in the following locator.

locator_byte: One of a sequence of bytes that together form the locator.

DVB_reserved_length: The number of DVB_reserved_bytes.

DVB_reserved_byte: A sequence of bytes reserved for future use.

imi_prepend_ptr: The offset, in bytes, from the first bytes of the data repository within the current container, to the first byte of the prepend string for the IMI. Where the name portion of the IMI is the same as the CRID authority name part of the CRID that is being resolved, this field may point at a zero length string, in which case the prepend string shall be considered to be the CRID authority name. The "imi:" prefix shall be omitted from the IMI prepend string.

result_imi_data_ptr: The offset, in bytes, from the first bytes of the data repository within the current container, to the first byte of the string which is the remaining part of the IMI. The act of concatenating the prepend string with the string pointed to by the result_imi_data_ptr results in a valid IMI.

If the imi_flag is set to "1" but a result does not have an IMI, this condition shall be signalled by the result_imi_data_ptr for that result pointing to a zero-length string. This may occur in the case where some but not all of the results have been assigned IMIs.

reresolve_date: The first date on which the receiver should try to re-resolve this CRID. This field uses Universal Co-ordinated Time (UTC) as the time reference. It shall be encoded as the number of days from the beginning of the year indicated by the year_offset field, the value zero indicating the 1st of January of that year.

NOTE 2: The size of this field allows the encoded date to extend into the year following that encoded in the year_offset field.

reresolve_time: The time, on the date given by reresolve_date, after which the receiver should try to re-resolve this CRID, using UTC as the time reference. This field shall be encoded as the number of 2-second periods since midnight.

7.3.2.3.3 DVB binary locator

The syntax of the DVB binary locator sub-structure is defined by table 30.

Table 30: DVB_binary_locator

Syntax	Number of bits	Identifier
dvb_binary_locator() {		
identifier_type	2	bslbf
scheduled_time_reliability	1	bslbf
inline_service	1	bslbf
reserved	1	bslbf
start_date	9	uimsbf
if (inline_service == "0") {		
DVB_service_triplet_ID	10	uimsbf
} else {		
reserved	2	bslbf
transport_stream_id	16	uimsbf
original_network_id	16	uimsbf
service_id	16	uimsbf
}		
start_time	16	uimsbf
duration	16	uimsbf
if (identifier_type == "01") {		
event_id	16	uimsbf
}		
if (identifier_type == "10") {		
TVA_id	16	
}		
if (identifier_type == "00" && scheduled_time_reliability == "1")) {		
early_start_window	3	uimsbf
late_end_window	5	uimsbf
}		
}		

identifier_type: This field indicates the type of event identifier used in this DVB binary locator. This field shall be encoded according to table 31.

Table 31: Identifier type

value	Meaning
"00"	no event identifier field is present
"01"	event identifier is an event_id
"10"	event identifier is a TVA_id
"11"	DVB Reserved

See clause 11.1 for details on the semantics of this field.

scheduled_time_reliability: This field only has meaning when the value of identifier_type is "00", i.e. when no event identifier is provided and the receiver will need to use the scheduled time to control recording. When set, the early_start_window and late_end_window information shall be provided, which the receiver may use to improve the reliability of capture of the item of content (see clause 11.1). If the value of identifier_type is not "00" then this field shall be set to "0".

inline_service: This flag indicates how the original network ID, transport stream ID and service ID for the DVB service that this locator refers to may be found. If set to "1" this flag indicates that these values are carried in-line in this structure. If set to "0" these values are found in the services structure in the current container and are referenced by the DVB_service_triplet_ID.

reserved: Reserved bits shall be set to "1".

start_date: The date on which the content pointed to by this locator is scheduled to start. This field uses Universal Co-ordinated Time (UTC) as the time reference. It shall be encoded as the number of days from the beginning of the year indicated by the year_offset field in the enclosing structure. The value zero indicates the 1st of January of that year.

NOTE: The size of this field allows the encoded date to extend into the year following that encoded in the year_offset field.

DVB_service_triplet_ID: A zero-based index into the services structure present in the current container. The entry at this index shall contain the details of the DVB service (original network ID, transport stream ID and service ID) that this locator refers to. See clause 7.3.2.4 for the definition of the services structure.

transport_stream_id: The transport stream ID for the transport stream that carries the DVB service that this locator refers to. If set to 0x0000 then this field shall be ignored and the DVB service shall be uniquely identified by a combination of original_network_id and service_id.

original_network_id: The original network ID for the transport stream that carries the DVB service that this locator refers to.

service_id: The DVB service ID for the DVB service that this locator refers to.

start_time: The time at which the content pointed to by this locator is scheduled to start, using UTC as the time reference. This is encoded as the number of 2-second periods since midnight.

duration: The duration of the event encoded as a count of 2-second periods.

event_id: The value of event_id that may be used to detect the transmission of the content pointed to by this locator. See clause 11.1.

TVA_id: The value of TVA_id that may be used to detect the transmission of the content pointed to by this locator. See clause 11.1.

early_start_window: This field indicates a duration by which the start of transmission for the content pointed to by this locator may occur ahead of the scheduled start_time. This field shall be encoded as a count of 1 minute periods, giving a range of 0 min to 7 min.

late_end_window: This field indicates a duration by which the end of transmission for the content pointed to by this locator may occur after the end time (which is calculated by adding the start start_time to the duration). This field shall be encoded as a count of 2 minute periods, giving a range of 0 min to 62 min.

7.3.2.4 Services

The services CRI structure is used to provide an efficient means of identifying a DVB service by index. Items in this structure are referenced by the DVB_service_triplet_ID field of the results CRI structure (see clause 7.3.2.3). The index of the first entry is zero. There shall be at most one instance of this CRI structure within a container. The syntax of the services structure is defined by table 32.

Table 32: Services structure

Syntax	Number of bits	Identifier
`Services() {`		
`  for(j=0; j<NumServices; j++) {`		
`  transport_stream_id`	16	uimsbf
`  original_network_id`	16	uimsbf
`  service_id`	16	uimsbf
`  }`		
`}`		

transport_stream_id: The transport stream ID for the transport stream that carries the DVB service. If set to 0x0000 then this field shall be ignored and the DVB service shall be uniquely identified by a combination of original_network_id and service_id.

original_network_id: The original network ID for the transport stream that carries the DVB service.

service_id: The DVB service ID for the DVB service.

7.3.2.5 Data repository

The data repository is used by both CRI index structures and CRI results structures for holding variable length strings. It can be present in an index container or in a results container. All references to a data repository refer to the data repository carried in the same container. The string_encoding field defines the encoding used for the strings contained within. There shall be at most one data repository in a container. The structure_id field in the container_header entry referring to a data_repository shall be set to 0x00. The syntax of the data repository is defined by table 33.

Table 33: Data repository structure

Syntax	Number of bits	Identifier
data_repository() {		
string_encoding	8	uimsbf
for (i=0; i<item_count; i++) {		
if (string_encoding < 0x03) {		
for (j=0; j<stringlength; j++) {		
string_character	8	uimsbf
}		
if(string_encoding == 0x00){		
0x00	8	uimsbf
} else if(string_encoding == 0x01){		
0x00	8	uimbsf
} else if(string_encoding == 0x02){		
0x0000		
}		
else { /* string_encoding ≥ 0x03*/		
for (j=0; j<stringlength; j++) {		
private_byte	8	uimsbf
}		
}		
}		
}		

string_encoding: An 8 bit field used to define the character encoding system. This field shall be encoded according to table 34.

Table 34: String encoding

Value	Description	Termination value
0x00	8 bit ASCII (ISO 646 [19])	0x00
0x01	UTF-8	0x00
0x02	UTF-16	0x0000
0x03 to 0xE0	reserved	undefined
0xE1 to 0xFF	User Private	undefined

7.3.3 CRI index structures

7.3.3.1 Description

The CRI indexing format consists of a three level indexing system (see figure 9). The first level is described by a cri_index structure and the second by cri_prepend_index structures and the third by cri_leaf_index structures.

The first level of indexing, consisting of a single cri_index structure, lists ranges of CRID values, referencing one cri_prepend_index structure for each range.

The second level, consisting of cri_prepend_index structures, groups CRIDs together that have the same initial string, termed a prepend string. Each entry in the cri_prepend_index contains a prepend string and references the entries in the third indexing layer that represent CRIDs that start with that prepend string.

The third level, consisting of cri_leaf_index structures, lists references to results and describes the variable string (the last part of the CRID). Every cri_prepend_index references one subordinate cri_leaf_index structure. The entries in a cri_prepend_index structure point to the relevant entries in the cri_leaf_index structure.

The full CRID string is fully recoverable by concatenating the appropriate prepend string from the second level with the variable string of the third level.

When assigning prepend strings the CRID string is treated as a simple sequence of characters that can be split at any point. The point at which the CRID string is split will depend on the amount of commonality in the first characters of the CRID strings encoded.

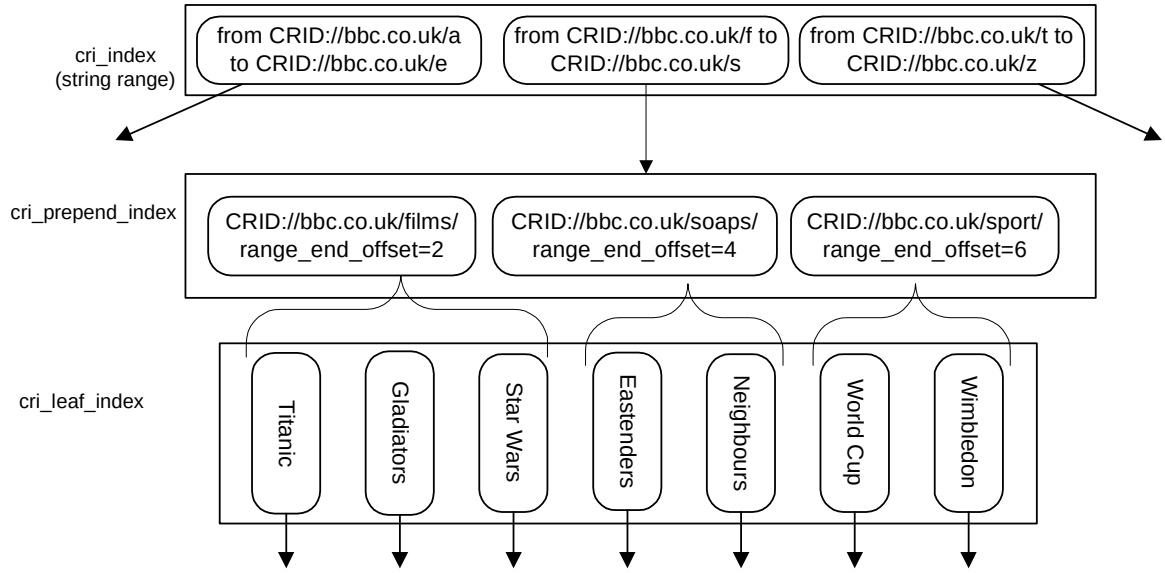

Figure 9: Example illustrating the use of string ranges, prepend strings and variable strings

7.3.3.2 Cri_index

The cri_index structure is the entry point for CRI carriage, it is the first structure that is located and decoded. It provides a list of all cri_prepend_index structures, describing the ranges of CRID values that those structures relate to.

When the overlapping_subindices flag is set to "1" the receiver should attempt to de-reference CRID results for a particular CRID by retrieving and parsing matching cri_prepend_index structures in the order that they are found in the cri_index structures.

Having overlapping ranges enables prioritization of particularly important CRIDs within the indexing structure. These CRIDs could be, for example, referencing programmes that are currently being broadcast or that will be broadcast very soon. Prioritization by grouping of results data is independent of prioritization (if any) within the indexing structures.

When cri_prepend_index ranges are not flagged as overlapping (overlapping_subindices set to "0"), the cri_prepend_index entries within the cri_index shall be in ascending lexicographical order. Lexicographical ordering shall be applied to the *encoded* CRID string, that is after converting characters outside of the standard Latin set into sequences of escaped octets, as defined in clause 6.2.

There shall only be one cri_index CRI structure within the CRI data delivered on a single PID. The syntax of the cri_index structure is defined by table 35.

Table 35: Cri_index structure

Syntax	Number of bits	Identifier
cri_index() {		
overlapping_subindices	1	bslbf
reserved_other_use	1	bslbf
reserved	6	bslbf
result_locator_format	8	uimsbf
for (i=0; i<sub_index_count; i++) {		
if (overlapping_subindices == 1) {		
low_key_value_CRID	16	uimsbf
}		
high_key_value_CRID	16	uimsbf
prepend_index_container	16	uimsbf
prepend_index_identifier	8	uimsbf
}		
}		

overlapping_subindices: When set to "1" indicates that one or more of the cri_prepend_index structures which form this index have ranges of values which overlap. Where cri_prepend_indices overlap entries in the cri_index structure are in descending order of search priority. When set to "0" indicates that the sub indices do not overlap, in which case the declared cri_prepend_index structures shall be ordered in ascending lexicographical order (see clause 6.2).

reserved_other_use: This field shall be set to "0".

reserved: All bits marked as being reserved shall be set to "1".

result_locator_format: Identifies the format of the result locator structure within cri_leaf_index CRI structures referenced from this structure. This field shall be encoded according to table 36.

Table 36 : result_locator_format

Value	Meaning
0x00	local_result_locator
0x01	remote_result_locator
0x02 to 0xFF	DVB reserved

See clause 7.3.3.5 for definitions of the local_result_locator and remote_result_locator.

low_key_value_CRID: This 16 bit field is the offset, in bytes, from the first byte of the data repository to the first byte of a CRID string. This CRID string must have a lexicographical value equal to or less than any CRID string referred to by the referenced cri_prepend_index structure (see clause 6.2).

If the overlapping_subindices field is set to "0" then this field shall not be used. In this case, every CRID string referred to by the referenced cri_sub_index structure must have an lexicographical value greater than the high_key_value_CRID of the previous entry in this structure.

high_key_value_CRID: This 16 bit field is the offset, in bytes, from the first byte of the data repository to the first byte of a CRID string. This CRID string must have a lexicographical value equal to or higher than any CRID string referred to by the referenced cri_prepend_index structure (see clause 6.2).

NOTE: When defining the range of values which a particular sub index shall cover, sufficient space can be left in containers to permit adding further CRID results without necessitating reallocation of the ranges of the sub index structures.

prepend_index_container: This field is the container ID of the container carrying the cri_prepend_index structure.

prepend_index_identifier: This 8 bit field shall be equal to the value of the cri_structure_id field of the target cri_prepend_index structure.

7.3.3.3 Cri_prepend_index

The cri_prepend_index CRI structure comprises the second level of indexing. Each cri_prepend_index structure provides references (via a cri_leaf_index structure) to CRID results that are within the lexicographical range of values specified by the cri_index structure.

For every cri_prepend_index structure there shall be a cri_leaf_index structure in the same container. These structures have the same cri_structure_type but are differentiated by their cri_structure_id values. All references to a cri_prepend_index or cri_leaf_index contain the cri_structure_id of the target structure, preventing confusion. Cri_prepend_index and cri_leaf_index CRI structures are also differentiable by the value of the leaf_flag field, which is present in both structures.

All entries within the sub index shall be ordered in ascending lexicographical order (see clause 6.2). When trying to select the appropriate prepend string used for a CRID being resolved, the receiver shall select the longest matching prepend string. A matching prepend string is defined as one where all characters in the prepend string match the corresponding characters in the CRID.

All entries that share the same prepend string shall be grouped together in the subordinate cri_leaf_index structure. These groups shall be in the same order as defined in the cri_prepend_index structure. For a given prepend string the correct range of cri_leaf_index entries is defined as being from one after the range_end_offset for the previous prepend string to the range_end_offset for the current prepend string. The start of the range for the first prepend string listed in a cri_prepend_index is the first entry in the subordinate cri_leaf_index.

The syntax of the cri_prepend_index structure is defined by table 37.

Table 37: Cri_prepend_index structure

Syntax	Number of bits	Identifier
`cri_prepend_index() {`		
`  leaf_flag`	1	bslbf
`  reserved`	7	uimsbf
`  sub_index_ref`	8	uimsbf
`  for (j=0; j<reference_count; j++) {`		
`  prepend_CRID_data`	16	uimsbf
`  range_end_offset`	16	uimsbf
`  }`		
`}`		

leaf_flag: This shall be set to "0" indicating that this structure is a cri_prepend_index.

reserved: All fields marked as being reserved shall have all bits set to "1".

sub_index_ref: This 8 bit field identifies the cri_structure_id of the cri_leaf_index that is the final level. The target cri_leaf_index shall be in the same container as this cri_prepend_index structure.

prepend_CRID_data: This 16 bit field gives the offset, in bytes, from the first byte of the data repository to the first byte of the prepend string.

range_end_offset: This 16 bit field gives the zero-based index of the last entry within the relevant cri_leaf_index structure that shares the current prepend string.

7.3.3.4 Cri_leaf_index

The cri_leaf_index CRI structure comprises the third level of indexing. Each entry in a cri_prepend_index structure provides the variable string part of a CRID and a reference to the results for that CRID.

All entries within this structure shall be ordered in ascending lexicographical order. The syntax of the cri_leaf_index structure is defined by table 38.

Table 38: Cri_leaf_index structure

Syntax	Number of bits	Identifier
cri_leaf_index() {		
leaf_flag	1	bslbf
reserved	7	uimsbf
for (j=0; j<reference_count; j++) {		
variable_CRID_data	16	uimsbf
result_locator()	variable	
}		
}		

leaf_flag: This 1 bit field shall be set to "1", indicating that this is a cri_leaf_index structure.

reserved: All fields marked as being reserved shall have all bits set to "1".

variable_CRID_data: This 16 bit field gives the offset, in bytes, from the first byte of the data repository to the first byte of the variable CRID string for this entry.

result_locator: This sub-structure references the results for this CRID. The format of this locator is dependent on the result_locator_format defined for this index within the cri_index structure. See clause 7.3.3.5 for the format of this field.

7.3.3.5 Result_locator formats

7.3.3.5.1 local_result_locator

If the CRI results structures are located in the same container as the cri_prepend_index CRI structure that refers to them then the local_result_locator format may be used. The syntax of the local_result_locator structure is defined by table 39.

Table 39: Local_result_locator

Syntax	Number of bits	Identifier
local_result_locator {		
result_ptr	16	uimsbf
}		

result_ptr: The offset, in bytes, from the first byte of the result_data CRI structure to the first byte of the relevant CRID results.

7.3.3.5.2 remote_result_locator

If the CRI results structures are located in a separate container than the cri_leaf_index CRI structure then the remote_result_locator format shall be used. The syntax of the remote_result_locator structure is defined by table 40.

Table 40: Remote_result_locator

Syntax	Number of bits	Identifier
remote_result_locator {		
target_container_id	16	uimsbf
target_handle	16	uimsbf
}		

target_container_id: The ID of the container holding the CRID content referencing result.

target_handle: This field identifies a CRID result within the target container. This value of this field shall match the value of the handle_value field that corresponds to the relevant CRID result in the target container's results_list structure (see clause 7.3.3.2).

8 Profile of TVA metadata over DVB transport streams

8.1 Introduction

The TV-Anytime metadata specification provides several options for how to structure descriptions of programme, group and schedule information. This clause defines the options that are either mandatory, optional or not used, for TV-Anytime metadata delivered over DVB transport streams.

NOTE: The present document does not provide any profiling for metadata delivered by other means.

8.2 Summary

Fragment	DVB profile
ProgramInformation	Required for support of metadata searching. In addition, a ProgramInformation fragment (with the same CRID) shall be present for each ScheduleEvent element that does not contain an InstanceDescription element. Otherwise optional.
GroupInformation	A GroupInformation fragment shall be present for each group CRID that is referenced by other fragments. Otherwise optional.
BroadcastEvent	Not used in the present document.
Schedule	Optional.
ServiceInformation	A ServiceInformation fragment shall be present for each serviceID referenced by other fragments. Otherwise optional.
PersonName (from CreditsInformationTable)	A PersonName fragment shall be present for each person that is referenced from other fragments. Otherwise optional.
OrganizationName (from CreditsInformationTable)	An OrganizationName fragment shall be present for each organization that is referenced from other fragments. Otherwise optional.
SegmentInformation	Not specified by this version of the present document.
Review (from ProgramReviewTable)	Optional.
OnDemandProgram	Optional.
OnDemandService	Optional.
ClassificationScheme	Mandatory if any classification scheme other than those defined in TS 102 822-3-1 [4] is referenced by any other fragment. Otherwise optional.

8.3 ProgramInformation fragment

When a program is a member of a series, the EpisodeOf element should be present. When a program is a member of a group that is not considered to be a series, the MemberOf element should be present.

When one or more Synopsis elements are present, the length attribute for each Synopsis element shall be present. The lang attribute of the Title and Synopsis elements should be present. The programID attribute of the ProgramInformation element shall contain a CRID. This CRID should be available in the CRI and will normally resolve to locators, but it may resolve to other CRIDs.

8.4 GroupInformation fragment

The groupId attribute of the GroupInformation element shall contain a CRID that resolves to other CRIDs and not locators. The Title element shall be present. When multiple synopsis elements are present, the length attribute of each Synopsis element shall be present.

8.5 Schedule fragment

The ScheduleEvent elements within the Schedule element shall be in chronological order, with the earliest item first. The Schedule element shall contain a start attribute that contains a time equal to or earlier to the PublishedStartTime of the first ScheduleEvent element and an end attribute that contains a time that is equal to or greater than the end of the last ScheduleEvent element. This end time is calculated by adding PublishedDuration to PublishedStartTime. Temporal gaps may also exist between consecutive ScheduleEvent elements.

The start and end times in Schedule elements shall not overlap with other Schedule elements for the same service. Taken together, the Schedule elements for a particular service shall form a contiguous chronological sequence.

The PublishedStartTime and PublishedDuration elements in the ScheduleEvent element shall be used, and the PublishedEndTime element shall not be present. The Program element shall be present and shall contain a CRID that can be found in the ProgramInformationTable and this CRID should also be present in the CRI.

The ProgramURL element may be present, to provide an indication of the expected broadcast location. If the ProgramURL refers to an event delivered in a DVB transport stream it shall contain a DVB locator that refers to an event, using the syntax as defined in clause 6.4. The CRI shall be considered the authoritative source of CRID to location information.

The InstanceMetadataId may be present and when present the same IMI shall be available in the CRI.

8.6 ServiceInformation fragment

Optional elements that are absent in a ServiceInformation element may be taken from DVB-SI information.

The ServiceURL element shall be present and shall contain a valid DVB locator that refers to a service, using the syntax as defined in clause 6.4.

The Name element shall either be an empty element, or contain the same name as specified by the SDT subtable for that service. If the Name element is empty, its contents shall be inferred from the SDT subtable for that service.

If the Logo element is present, it shall contain a URL that points to an image file.

8.7 Other types

Elements of type anyURI, as defined by XML Schema Part 2: Datatypes [14], may contain a DVB locator (see clause 6.4).

9 Delivery of metadata

9.1 Introduction

The TV-Anytime forum has defined how metadata shall be encoded and encapsulated for delivery over a unidirectional network (see TS 102 822-3-2 [5]).

There are several aspects to this technology (see TS 102 822-3-2 [5], clause 4.2):

- Fragmentation: splitting the metadata document into a number of self-contained fragments.
- Encoding: binary encoding of fragments for efficient delivery.
- Encapsulation: structures for identification of fragments and concatenation of fragments into containers.
- Indexing: structures for identification of fragments according to the value of an attribute or element.

Both metadata and indices are carried in containers. However, TS 102 822-3-2 [5] does not define how containers are to be delivered in a particular environment, such as DVB transport streams. In clause 4.5.1 of that specification the following requirements are defined for the carriage of containers:

- Containers are to be classified into data containers or index containers (or containers that carry both).
- Containers are to be identifiable by a 16 bit identifier.

This clause defines how TV-Anytime metadata may be encoded and delivered on DVB transport streams. This clause implements TS 102 822-3-2 [5] and defines a number of additional extensions and constraints. These include:

- Carriage of metadata containers by object carousel, including classification and identification.
- Additional semantics for encapsulation of fragments.
- Additional semantics and codecs for encoding of fragments using BiM.
- A set of profiled indices.

9.2 Delivery of containers

9.2.1 Delivery by MHP object carousel

The metadata_pointer_descriptor and the metadata_descriptor both contain a DVB_carriage_format field, which signals the transport protocol being used for the metadata service being described (see clause 5.3.4). If the value of this field is set to 0x00 then the metadata shall be delivered in an object carousel compliant with the object carousel profile as defined in annex B of TS 102 812 [10] with the additional constraints defined in this clause.

Containers, as defined in TS 102 822-3-2 [5], clause 4.5.1, shall be carried within BIOP::FileMessages (file objects). All containers for a single metadata service shall be referenced from the root directory (BIOP::DirectoryMessage or BIOP::ServiceGatewayMessage) for that metadata service, as indicated by the relevant metadata descriptor (see clause 5.3.4). Containers for a metadata service shall not be carried in sub-directories of the directory signalled by the metadata descriptor for that metadata service. One object carousel may contain several metadata services.

Since the MHP profile of the object carousel provides a number of features that are not required for the delivery of TVA metadata, within the context of the present document the constraints defined in table 41 shall be observed.

Table 41: MHP object carousel constraints

MHP section		
B.2.2.4.1	Label descriptor	Not used by the present document (see note).
B.2.2.4.2	Caching priority descriptor	Not used by the present document (see note).
B.2.3.4	Content type descriptor	Not used by the present document (see note).
B.2.3.7.2	LiteOptionsProfileBody	All containers of a metadata service shall be carried in a single object carousel. Therefore, LiteOptionsProfileBody may not form part of the reference to any such container. However, external assets, e.g. images, audio clips, referenced by the metadata service may be delivered in a different object carousel, the LiteOptionsProfileBody may be used as part of the reference to any such external assets.
B.2.3.8	BIOP StreamMessage	Not used by the present document (see note).
B.2.3.9	BIOP StreamEventMessage	Not used by the present document (see note).
B.2.4	Stream Events	Not relevant to the present document.
B.2.10.2	DVB-J mounting of an object carousel	Not relevant to the present document.
B.3.2	DSM-CC association_tags to DVB component_tags	All DIIs and DDBs used in the delivery of a metadata service shall be carried in elementary streams that are listed in the PMT that carries the metadata descriptor for that metadata service (see clause 5.3.5). Therefore, use of the deferred_association_tags_descriptor is not required.
B.3.1.2	TapUse is BIOP_PROGRAM_USE	Not used by the present document (see note).
B.5	Caching	Informative for receiver manufacturers.
NOTE: Metadata services shall not include such information in the object carousel. Receivers may ignore the presence of such information in the object carousel when accessing metadata services.		

9.2.2 Container file names

The format for the file name for container files (i.e. the format of the NameComponent ID given in the binding that references a container file) is defined by table 42.

Table 42: File name format for container files

```
container_file_name         = mapped_container_id "." classification_indicator
mapped_container_id         = hex_string
classification_indicator    ="d" | "i" |"b"
hex_string                  = 4*hex
hex                         = digit | "A" | "B" | "C" | "D" | "E" | "F" | "a" | "b" | "c" |
                                "d" | "e" | "f"
digit                       = "0" | "1" | "2" | "3" | "4" | "5" | "6" | "7" | "8" | "9"
```

The mapped_container_id carries the container ID for this container expressed as a hex_string.

The classification_indicator is a single character that indicates the classification of the container (see TS 102 822-3-2 [5], clause 4.5.1.2). It shall be encoded according to table 43.

Table 43: classification_indicator

Value	Semantics
"d"	The container is a data container
"i"	The container is an index container
"b"	The container is both a data container and an index container
other values	reserved

Table 44 contains example file names.

Table 44: Example file names

Example	Container ID	Container classification
"001F.d"	0x001F	data container
"0DB6.b"	0x0DB6	data container and index container
"0000.i"	0x0000	index container which carries index list

9.3 Fragment encapsulation

9.3.1 Introduction

TS 102 822-3-2 [5], clause 4.6, defines how fragments shall be encapsulated within containers. As part of that encapsulation the encapsulation structure uses fragment references as a means to reference fragments and provide version information for those fragments. The present document defines a DVB BiM fragment reference, which is necessary for supporting the dvbStringCodec defined in clause 9.4.3.3. The DVB BiM fragment reference differs from the fragment reference defined in TS 102 822-3-2 [5] by containing an additional external string buffer reference.

Figure 10 illustrates the format of a data container where the encapsulation structure uses the DVB fragment reference.

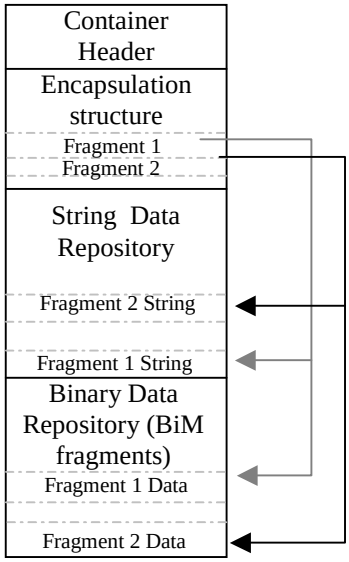

Figure 10: Example illustrating container when using DVB fragment references

9.3.2 Encapsulation structure

This clause defines additional semantics of fields of the encapsulation structure as defined in TS 102 822-3-2 [5], clause 4.6.1.1.

fragment_reference_format: This field defines the format and the interpretation of the fragment_reference field. This field shall be encoded according to table 45.

Table 45: Fragment reference format

Value	Semantics
0x00 to 0xE0	Defined by TS 102 822-3-2 [5], clause 4.6.1.1
0xE1	DVB_BiM_fragment_reference
0xE2 to 0xEF	DVB Reserved
0xF0 to 0xFF	User defined

If a container carries fragments that use the dvbStringCodec then the fragment_reference_format field shall be set to 0xE1.

9.3.3 DVB BiM fragment reference

The format of the DVB BiM fragment reference is defined by table 46.

Table 46: DVB BiM fragment reference

Syntax	Number of bits	Mnemonic
DVB_BiM_fragment_reference () {		
BiM_fragment_ptr	16	uimsbf
external_string_buffer_ptr	16	uimsbf
}		

BiM_fragment_ptr: The zero-based offset in bytes from the start of the binary repository within this container to the first byte of the FragmentUpdateUnit() enclosing the intended fragment.

external_string_buffer_ptr: The zero-based offset in bytes from the start of the string repository within this container to the first byte of the external string buffer for the fragment.

The structure of the data repositories of type string data (or "string repository") is specified in TS 102 822-3-2 [5], clause 4.8.4.1. The structure of the data repositories of type binary data (or "binary repository") is defined in clause 9.4.3.2.

9.4 Fragment encoding

9.4.1 Introduction

In order to simplify the implementation of BiM decoders and to improve the compression of TV-Anytime fragments, the present document imposes a number of restrictions on the TVA MPEG-7 BiM profile, as defined in TS 102 822-3-2 [5].

9.4.2 Rules for BiM encoding

9.4.2.1 DVB-TVA-init message

The present document defines a DVB profile of the TVA MPEG-7 profile defined in TS 102 822-3-2 [5]. A new encoding version is introduced and a number of restrictions have been imposed on the initialization of the BiM decoders. The DVB-TVA-init message shall be delivered as specified in clause 5.3.4. The DVB-TVA-init message is adapted from the TVA-init message defined in TS 102 822-3-2 [5], clause 4.4.1, it shall be encoded according to table 47.

Table 47: DVB-TVA-init

Syntax	Number of bits	Mnemonic
DVB-TVA-init {		
EncodingVersion	8	uimsbf
IndexingFlag	1	bslbf
reserved	7	
DecoderInitptr	8	bslbf
if(EncodingVersion == "0x01" ||		
EncodingVersion == "0xF0") {		
BufferSizeFlag	1	bslbf
PositionCodeFlag	1	bslbf
reserved	6	
CharacterEncoding	8	uimsbf
if (BufferSizeFlag == "1") {		
BufferSize	24	uimsbf
}		
}		
if(IndexingFlag) {		
IndexingVersion	8	uimsbf
}		
Reserved	0 or 8+	
DecoderInit( )		bslbf
}		

The semantic of all terms of the DVB-TVA-init message is as defined for the TVA-init message defined in TS 102 822-3-2 [5], excepted for the PositionCodeFlag and EncodingVersion fields.

EncodingVersion: This field indicates the method of encoding used to represent the TVA metadata fragments. This field shall be encoded according to table 48.

Table 48: Encoding version

Value	Encoding version
0x00 to 0xEF	TVA reserved
0xF0	DVB profile of TVA MPEG_7 profile (BiM) ISO/IEC 15938-1 [9] as defined in the present document
0xF1 to 0xF7	DVB reserved
0xF8 to 0xFF	User defined

PositionCodeFlag: This field indicates if the BiM contextPath Position Code is used in the encoded fragment. This field shall be set to "0".

NOTE 1: This value indicates that the position code is not used within the contextPath, as defined in clause 9.4.2.3. This means that the canonical format of the instance description is not preserved, i.e. the relative ordering of fragments is not preserved.

CharacterEncoding: This field shall be encoded as defined by TS 102 822-3-2 [5], clause 4.4.1.

NOTE 2: For compatibility it is advised that receivers should at least support the UTF-8 character encoding format.

9.4.2.2 DecoderInit and default TVAMain fragment

In BiM, the DecoderInit is used to configure parameters required for the decoding of the binary fragments and to transmit the initial state of the decoder.

At least one schema URN shall be transmitted in the DecoderInit. Consequently, the field NumberOfSchemas of the DecoderInit shall be greater or equal to 1 and the field SchemaURI[0] of the DecoderInit shall be set to "urn:tva:metadata:2002", indicating the use of the schema defined by TS 102 822-3-1 [4].

Additional schemas may be signalled if the TV-Anytime metadata types are extended. If this is the case the zero-based index of the entry of the schemas also serves as a means of defining namespace prefixes in index XPaths. This is an extension to the predefined prefixes defined in TS 102 822-3-2 [5], clause 4.8.5.3. The resulting namespace prefixes shall be generated according to the format defined in table 49. An example of the use of namespace prefixes for extended schemas is given in clause D.3. Since the first schema signalled in the DecoderInit is always that defined by TS 102 822-3-1 [4] and that schema has a predefined prefix (see TS 102 822-3-2 [5], clause 4.8.5.3), the prefix "d0" is disallowed.

Table 49: Format for namespace prefixes

```
namespace_prefix            = "d" namespace_index
namespace_index             = * digit
digit                       = "0" | "1" | "2" | "3" | "4" | "5" | "6" | "7" | "8" | "9"
```

The initial state of a BiM decoder for a binary description tree is given by an initial description. In a TVA fragment stream, the initial description is given by the TVAMain fragment.

In the present document, the transmission of the initial description to the decoder is not mandatory. If the TVAMain fragment is not delivered to the decoder, the decoder is initialized with a default TVAMain fragment. The default TVAMain fragment is defined in table 50.

Table 50: Default TVAMain fragment

```
<TVAMain xmlns='urn:tva:metadata:2002'>
    <ClassificationSchemeTable />
    <ProgramDescription>
        <ProgramInformationTable />
        <GroupInformationTable />
        <ProgramLocationTable />
        <ServiceInformationTable />
        <CreditsInformationTable />
        <ProgramReviewTable />
        <SegmentInformationTable>
            <SegmentList />
            <SegmentGroupList />
        </SegmentInformationTable>
    </ProgramDescription>
</TVAMain>
```

If the TVAMain fragment is delivered to the decoder, it shall be encapsulated in a data container as defined in the clause 9.3, and this container shall be delivered as defined in the clause 9.2 with the following restriction: the name of the file name for the container which conveys the TVAMain fragment shall be </TVAMain>.

As a consequence, the InitialDescription() field of the DecoderInit message as specified in ISO/IEC 15938-1 [9], shall always be empty.

9.4.2.3 DVB BiM access unit

In BiM, a path is encoded at the beginning of each fragment. This path specifies the type of the element encoded in the fragment. The TVA MPEG-7 BiM profile specify the fixed set of possible TV-Anytime fragments. The present document replaces the resulting ContextPath expressions in the fragment encoding, whose format is defined in ISO/IEC 15938-1 [9], clause 7.6.5, by a set of fixed ContextPath values identifying the type of the TVA fragments.

As a consequence, the definition of the binary_ repository table as specified in TS 102 822-3-2 [5], clause 4.6.1.4.1.1, is profiled according to table 51.

Table 51: Binary repository carrying DVB BiM access unit

Syntax	Number of bits	Identifier
binary_repository() {		
DVBBiMAccessUnit {		
NumberOfFUU	8+	vluimsbf8
for(i=0; i< NumberOfFUU; i++) {		
FUULength	8+	vluimsbf8
DVBContextPath	16	uimsbf
FragmentUpdatePayload(startType)		
}		
}		
for (i=0; i<N; i++) {		
private_byte	8	uimsbf
}		
}		

startType: This flag is set to the EquivalentStartType associated to the value of the DVBContextPath flag as defined in table 52.

FragmentUpdatePayload: Triggers the decoding of the TV-anytime fragment of type startType as defined in ISO/IEC 15938-1 [9], clause 8.3.

DVBContextPath: This field identifies the type of the fragment. It shall be encoded according to table 52. This table defines the DVBContextPath value for each fragment type defined by TS 102 822-3-2 [5], clause 4.3. The EquivalentStartType type names are namespace qualified and use the following namespace prefix:

- tva urn:tva:metadata:2002.

Table 52: DVBContextPath

Value	Description	EquivalentStartType
0x0000	reserved	
0x0001	ProgramInformation fragment	tva:ProgramInformationType
0x0002	GroupInformation fragment	tva:GroupInformationType
0x0003	OnDemandProgram fragment	tva:OnDemandProgramType
0x0004	BroadcastEvent fragment	tva:BroadcastEventType
0x0005	Schedule fragment	tva:ScheduleType
0x0006	ServiceInformation fragment	tva:ServiceInformationType
0x0007	PersonName fragment	LocalType(tva:CreditsInformationTableType, PersonName)
0x0008	OrganizationName fragment	LocalType(tva:CreditsInformationTableType, OrganizationName)
0x0009	Review fragment	LocalType(tva:ProgramReviewTableType, Review)
0x000A	CSAlias fragment	LocalType(tva:ClassificationSchemeTableType, CSAlias)
0x000B	ClassificationScheme fragment	LocalType(tva:ClassificationSchemeTableType, ClassificationScheme)
0x000C	SegmentInformation fragment	tva:SegmentInformationType
0x000D	SegmentGroupInformation fragment	tva:SegmentGroupInformationType
0x000E	TVAMain fragment	tva:TVAMainType
0x000F	OnDemandService fragment	tva:OnDemandServiceType
0x0010 to 0x00EF	DVB Reserved	
0x00F0 to 0xFFFF	User defined	

LocalType(T, E): represents the local type of the child element whose name is E in the content model of the type T. In XML Schema, local types defined within the content model of a type T are anonymous (no global name for T is defined). As none W3C tool is currently available in order to point to such types, the function LocalType has been defined.

EXAMPLE: LocalType(tva:CreditsInformationTableType, PersonName) represents the local type of the child elements whose name are "PersonName" in the type tva:CreditsInformationTableType. See figure 11.

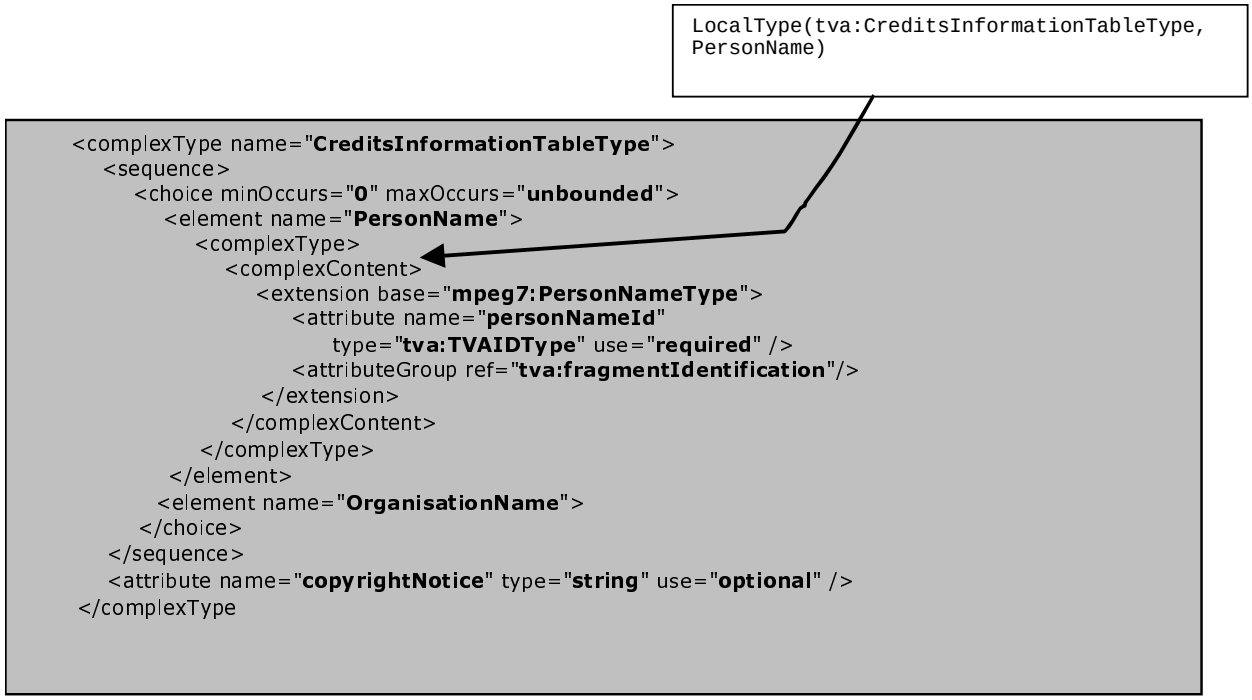

Figure 11: Example for localType function

9.4.3 Codec definitions

9.4.3.1 Introduction

The present document defines a set of codecs that shall be used by default for the encoding of TV-Anytime fragment in the DVB-GBS context.

9.4.3.2 Classification scheme of DVB codecs

In the MPEG-7 framework, the use of a specific codec for a specific type is signalled using the codec configuration mechanism defined in ISO/IEC 15938-1 [9]. This mechanism associates a codec using its URI with a list of schema types. For that purpose, a URI is assigned to each codec in a classificationScheme, which defines the list of the specific codecs.

In the present document, this list is composed of the following codecs: dvbStringCodec, dvbDateTimeCodec, dvbDurationCodec, dvbLocatorCodec, and dvbControlledTermCodec. The following figure gives the standard ClassificationScheme used by the present document.

```
<ClassificationScheme uri="urn:tva:metadata:2002:cs:CodecTypeCS ">
    <Term termID="1">
        <Name xml:lang="en">dvbStringCodec</Name>
        <Definition xml:lang="en">Encodes string by using an external string
             buffer</Definition>
    </Term>
    <Term termID="2">
        <Name xml:lang="en">dvbDateTimeCodec</Name>
        <Definition xml:lang="en">Encodes date using Modified Julian Date & Time in
            Millisecond and differential encoding</Definition>
    </Term>
    <Term termID="3">
        <Name xml:lang="en">dvbDurationCodec</Name>
        <Definition xml:lang="en">Encodes duration using strings or
    approximation with an accuracy of 1 minute </Definition>
    </Term>
    <Term termID="4">
        <Name xml:lang="en">dvbLocatorCodec</Name>
        <Definition xml:lang="en">Encodes DVB Locator using prefix dictionary </Definition>
    </Term>
    <Term termID="5">
        <Name xml:lang="en">dvbControlledTermCodec</Name>
        <Definition xml:lang="en">Encodes Controlled Terms using indices</Definition>
    </Term>
</ClassificationScheme>
```

Figure 12: DVB codec classification

9.4.3.3 dvbStringCodec

9.4.3.3.1 Introduction

The dvbStringCodec codec, as defined in this clause, may be used for the encoding of strings.

9.4.3.3.2 Rationale and encoding process (informative)

9.4.3.3.2.1 Rationale

Most TV-Anytime fragments are likely to have a rather small size. This implies a limited redundancy over the string data, and therefore the efficiency of the zlibCodec string codec, as specified by TS 102 822-3-2 [5], might be poor. On another hand some redundancy in the strings across different fragments can be expected. Therefore, to achieve good compression ratio, the optimized dvbStringCodec codec gather the strings of all binary fragments of a TV-Anytime container in a single string repository within this container. The delivery method used for carrying the TV-Anytime data containers applies a compression method to the whole container. For instance, a metadata container may be transmitted within a compressed object carousel module. Therefore the statistical compression can take advantage of the inter-fragment redundancy.

9.4.3.3.2.2 Encoding

At the encoding time, the dvbStringCodec gathers all the strings from a BiM fragment in an external string buffer. The external string buffers of all fragments of a TV-Anytime container shall be stored in a string repository in the same container. This dvbStringCodec is reinitialized for each TV-Anytime fragment. The syntax of the string repository is as specified in TS 102 822-3-2 [5], clauses 4.6.1.4 and 4.8.4.1.

Figure 13 represents how a regular BiM bitstream can be converted into a BiM bitstream as supported by the present dvbStringCodec and its associated external string repository.

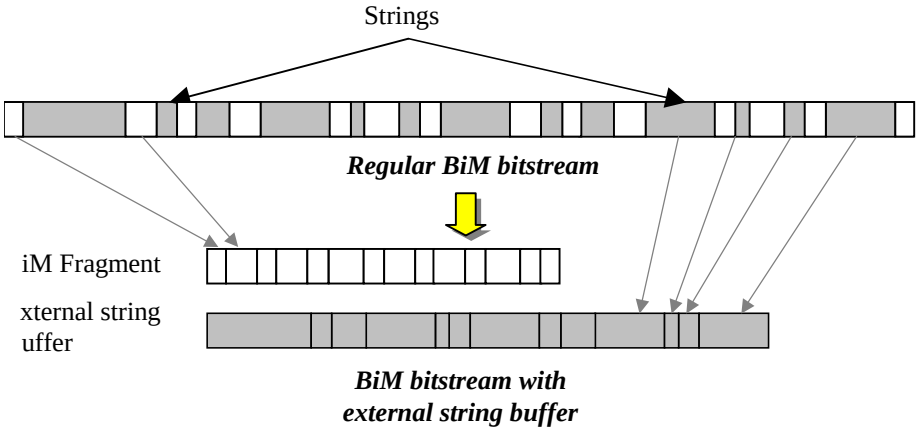

Figure 13: BiM bitstream with an external string buffer

The external string buffers are sets of consecutive strings separated by a string_terminator as defined TS 102 822-3-2 [5], clause 4.8.4.1.

9.4.3.3.2.3 Decoding principle

The principle of the decoding a BiM fragment with an external string buffer is the same as the decoding of a regular BiM bitstream. The main difference so far is that the string codec is reading the string data from the external string buffer instead of reading the strings from the main regular BiM bitstream.

At the beginning of the decoding of a BiM Fragment unit, the string codec is initialized with a reference to the first byte of the first string of the external string buffer.

9.4.3.3.2.4 Managing schema compatibility and skippable chunks

Things get a bit complicated when one want the string codec to support schema compatibility (in that case, the decoder can skip extended elements - see clause 9.4.4) or allow the user to skip parts of the bitstream. Indeed, at decoding time after a chunk of the bitstream is skipped, the string codec must be re-synchronized at the appropriate place in the external string buffer.

This is done by inserting in the BiM bitstream, for each string immediately following the end of a skippable element, the offset of this string in the external string buffer. When decoding the string, the decoder reads this offset and use it to re-synchronize the string codec at the appropriate location in the external string buffer.

Figure 14 illustrates how the bitstream shall be encoded in order to support the skip of an element which preceded a string element.

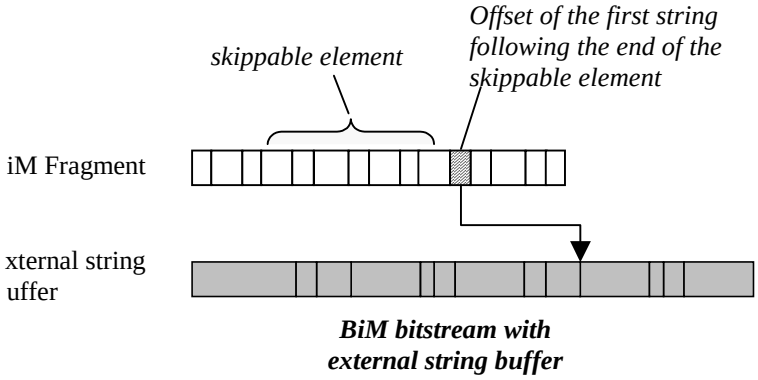

Figure 14: BiM bitstream with a skippable element before a string element

9.4.3.3.3 Decoding

The format for this codec is defined in table 53.

Table 53: dvbStringCodec

Syntax	Number of bits	Identifier
dvbStringCodec () {		
if(isFollowingSkippableElement == 1) {		
string_offset	16	uimsbf
resynchronizeCodec(string_offset)		
}		
getNextStringFromBuffer()		
}		

isFollowingSkippableElement: This field shall be set to "1" when the decoder is decoding the first of the strings following the end of a skippable element. In this case, this means that the offset of this string in the external string repository shall be read in the bitstream.

string_offset: The offset in bytes, from the beginning of the external string buffer, of the first character of the string being decoded.

resynchronizeCodec(string_offset): Synchronizes the dvbStringCodec on the first byte, in the external string buffer, of the string to be decoded. The offset, in bytes, from the beginning of the external string buffer, is given by the string_offset parameter.

getNextStringFromBuffer(): Reads the current string in the external string buffer, and jumps to the first byte of the next string of the buffer.

9.4.3.4 dvbLocatorCodec

9.4.3.4.1 Usage

The dvbLocatorCodec is used by default for the encoding of elements or attributes of type anyURI in order to efficiently encode the DVB Locator as defined in clause 6.4.

9.4.3.4.2 Rationale and encoding process (informative)

Within a single TV-Anytime fragment a given DVB locator prefix, composed of the original networks ID, a transport stream ID and a service ID, is often reused by several CRIDs. In order to efficiently encode this repetition, the dvbLocatorCodec reuses the prefix of the decoded DVB locator which is immediately preceding in the fragment if it has the same prefix.

9.4.3.4.3 Decoding

The format of the DVBLocatorCodec is defined by table 54.

Table 54: DVBLocatorCodec

Name	Number of bits	Identifier
DVBLocatorCodec() {		
optimized_codec_flag	1	bslbf
if (optimized_codec_flag == 1) {		
OptimizedDVBLocator()		
} else {		
dvbStringCodec()		
}		
}		

optimized_codec_flag: A flag which indicates whether the optimized codec is used or not. If not, the default string encoding codec is used.

dvbStringCodec(): For the definition of the dvbStringCodec() see clause 9.4.3.3.

OptimizedDVBLocator(): This field encodes the DVB locator in a optimized way. The format for this field is defined by table 55.

Table 55: OptimizedDVBLocator

Name	Number of bits	Identifier
OptimizedDVBLocator() {		
prefix_flag	1	bslbf
if (prefix_flag ==1) {		
DVBLocatorPrefix()		
}		
ctag_flag	1	bslbf
if (ctag_flag ==1) {		
component_tags()		
}		
eventOrTVAflag	2	bslbf
if (eventOrTVAflag == 01) {		
event_id	16	uimsbf
}		
else if (eventOrTVAflag == 10) {		
TVA_id	16	uimsbf
}		
time_flag	1	bslbf
if (time_flag = 1){		
day_flag	1	bslbf
if (day_flag = 1) {		
day	16	uimsbf
}		
time	17	
duration	17	
}		
path_segments_flag	1	bslbf
if (path_segments_flag ==1) {		
path_segments()		
}		
}		

prefix_flag: If the prefix_flag is set to "1" the DVBLocatorPrefix is encoded. For the first occurrence in the fragment, this flag shall be set to 1. For subsequent occurrences, the prefix_flag may be set to "0" in which case the previous value of DVBLocatorPrefix encoded in this fragment is reused.

If this field is set to "1" within a skippable encoded chunk relating to a schema extension, then the next occurrence of this codec outside of that encoded chunk shall set this field to "1". For extending the TV-Anytime schema see clause 9.4.4.

NOTE 1: This is to prevent problems with the reused information being set in part of the BiM bitstream inaccessible by a receiver that doesn't understand the extended schema. This receiver would use the wrong value of the reused information for the next occurrence.

DVBLocatorPrefix: This field encodes the original_network, the transport_stream and the service_id. If the transport_stream_id field is set to 0x0000 then it shall be ignored and the DVB service shall be uniquely identified by a combination of original_network_id and service_id. The format of this field is defined by table 56.

Table 56: DVBLocatorPrefix

Name	Number of bits	Identifier
DVBLocatorPrefix() {		
transport_stream_id	16	uimsbf
original_network_id	16	uimsbf
service_id	16	uimsbf
}		

ctag_flag: If the ctag_flag is set to "1" the one or more component_tags() field of the dvb locator are encoded.

component_tags(): This field encodes the component_tag fields of the dvbLocator in a string representation. The format of this field is defined by table 57.

Table 57: component_tag

Name	Number of bits	Identifier
component_tags() {		
dvbStringCodec()		
}		

eventOrTVAFlag: This field indicates if an event_id or a TVA_id or none of them is encoded. This field shall be encoded according to table 58.

Table 58: eventOrTVAFlag

Value	Description
"00"	event-id and TVA-id are not encoded
"01"	event_id is encoded
"10"	TVA_id is encoded
"11"	reserved

time-flag: This flag shall be set to "1" if time information is encoded. Otherwise it shall be set to "0".

day_flag: If the day_flag is set to "1", the day is encoded. For the first occurrence in a fragment this flag shall be set to "1". For subsequent occurrences the day_flag may be set to "0" in which case the previous value of day encoded in this fragment is reused.

If this field is set to "1" within a skippable encoded chunk relating to a schema extension, then the next occurrence of this codec outside of that encoded chunk shall set this field to "1". For extending the TV-Anytime schema see clause 9.4.4.

NOTE 2: This is to prevent problems with the reused information being set in part of the BiM bitstream inaccessible by a receiver that doesn't understand the extended schema. This receiver would use the wrong value of the reused information for the next occurrence.

day: The date to which this locator refers. This field is coded as 16 bits giving the 16 LSBs of MJD. See EN 300 468 [1], annex C.

time: The time of day to which this locator refers. This field is represented using 17 bits expressed as the number of elapsed seconds since midnight.

duration: Is represented using 17 bits expressed as the number of elapsed seconds.

path_segments(): This field encodes the path_segments component of the dvbLocator in a string representation. The format of this field is defined by table 59.

Table 59: path_segments

Name	Number of bits	Identifier
path_segments() {		
dvbStringCodec()		
}		

dvbStringCodec(): For the definition of the dvbStringCodec see clause 9.4.3.3.

9.4.3.5 dvbDateTimeCodec

9.4.3.5.1 Rationale and encoding process (informative)

The XML Schema primitive simple type dateTime is used widely within the TVA metadata Schema, and so a specific codec has been designed to represent date time elements or attributes.

Times shall be based on UTC, with no provision provided for maintaining the local time offset information. Any requirement to localize time values shall be performed by the receiving terminal.

The dateTime type is used by the following TV-Anytime types: TVAMainType, ScheduleType, ScheduleEventType, OnDemandProgramType, and ServiceRefType.

Within a single TV-Anytime fragment a given date is often reused. In order to efficiently encode this repetition, the dvbDateTimeCodec can reuse the value of the immediately preceding date instead of re-encoding it.

When the date is different from the previously encoded one, the date is represented using 2 bytes as a Modified Julian Date.

In order to further improve the compression ratio of dateTime elements or attributes, the time information are represented using 11 bits, with an accuracy of 1 min.

9.4.3.5.2 Decoding

The format of the dvbDateTimeCodec is defined by table 60.

Table 60: dvbDateTimeCodec

Name	Number of bits	Identifier
dvbDateTimeCodec() {		
dateTime_Flag	2	bslbf
if (dateTime_flag==00) {		
dateTimeOfTVA	64	bslbf
}		
if (dateTime_flag==01) {		
PublishedTime()		
}		
}		

dateTime_flag: This flag is used to state the encoding mode of the dateTime element or attribute. When an accuracy of one minute is sufficient, publishedTime() should be used. PublishedTime() is recommended for encoding the publishedStartTime and publishedEndTime sub-elements of the ScheduleEventType type. This field shall be encoded according to table 61.

Table 61: dateTime_flag

Value	Semantic
00	the dateTime codec as defined in TS 102 822-3-2 [5], clause 4.4.2.4.2 is used
01	A published time is encoded
10 to 11	reserved

dateTimeOfTVA: The dateTime elements or attributes shall be encoded with the dateTimeCodec as defined in TS 102 822-3-2 [5], clause 4.4.2.4.2.

PublishedTime(): Optimized encoding of the dateTime elements or attributes as defined in the present document. This field shall be formatted according to table 62.

Table 62: PublishedTime

Name	Number of bits	Identifier
PublishedTime() {		
date_flag	1	bslbf
if (date_flag == 1) {		
date	16	bslbf
}		
time	11	uimsbf
}		

date_flag: If the this field is set to "1", the date is encoded. For the first occurrence in the fragment, this field shall be set to "1". For subsequent occurrences, the date_flag field may be set to "0" in which case the previous value of date encoded in this fragment is reused.

If this field is set to "1" within a skippable encoded chunk relating to a schema extension, then the next occurrence of this codec outside of that encoded chunk shall set this field to "1". For extending the TV-Anytime schema see clause 9.4.4.

NOTE: This is to prevent problems with the reused information being set in part of the BiM bitstream inaccessible by a receiver that doesn't understand the extended schema. This receiver would use the wrong value of the reused information for the next occurrence.

date: Modified Julian Date represented using 2 bytes as defined in EN 300 468 [1], annex C.

time: Time of the day represented using 11 bits, expressed as the number of elapsed minutes since midnight.

9.4.3.6 dvbDurationCodec

9.4.3.6.1 Rationale and encoding process (informative)

The XML Schema primitive simple type duration is used widely within the TVA metadata Schema, and so a specific codec has been designed to represent date time elements or attributes.

The XML Schema primitive simple type duration is used by the following TV-Anytime types: ScheduleEventType, and OnDemandProgramType.

In order to efficiently encode duration elements or attributes, it is recommended to represent the duration information are represented using 11 bits, with an accuracy of 1 min.

9.4.3.6.2 Decoding

The format of the dvbDurationCodec is defined by table 63.

Table 63: dvbDurationCodec

Name	Number of bits	Identifier
dvbDurationCodec() {		
encoding_flag	1	bslbf
if (encoding_flag == 0) {		
dvbStringCodec()		
}		
if (encoding_flag == 1) {		
minutes	11	uimsbf
}		
}		

encoding_flag: If the encoding_flag is set to 0, the duration is encoded as a string as specified by XML Schema part 2 [14], clause 3.2.6. If the encoding_flag is set to 1, the duration is encoded as a number of minutes.

dvbStringCodec(): For the definition of the dvbStringCodec see clause 9.4.3.3.

minutes: The duration is encoded as a number of minutes and represented using 11 bits.

9.4.3.7 dvbControlledTermCodec

9.4.3.7.1 Usage

The dvbControlledTermCodec codec is used by default for encoding elements and attributes of type termReferenceType.

9.4.3.7.2 Rationale and encoding process (informative)

Default classification schemes have been developed by TV-Anytime to provide a universally applicable default set of classification terms.

The following dvbControlledTermCodec shall be used by default for encoding the fixed values defined by TV-Anytime. It encodes references to the classification scheme and a controlled term from that scheme.

When using the dvbControlledTermCodec for fixed terms defined by TV-Anytime classification schemes, it is recommended that elements of type ControlledTermType should not include Name or Definition child elements.

9.4.3.7.3 Decoding

The format of the ControlledTermCodec is defined by table 64.

Table 64: ControlledTermCodec

Name	Number of bits	Identifier
`dvbControlledTermCodec () {`		
`encoding_flag`	1	bslbf
`if (encoding_flag == 0) {`		
`dvbStringCodec()`		
`}`		
`if (encoding_flag == 1) {`		
`ClassificationSchemeID`	8	uimsbf
`termID`	8+	vluismbf8
`}`		
`}`		

encoding_flag: If the encoding_flag is set to 0, the term reference is encoded as a string. If the encoding_flag is set to 0, the term reference as a pair of classification schema identifier and term identifier.

dvbStringCodec(): For the definition of the dvbStringCodec see clause 9.4.3.3.

ClassificationSchemeID: An identifier for the classification scheme defined in TS 102 822-3-1 [4] annex A. This field shall be encoded according to table 65.

Table 65: ClassificationSchemeID

Value	Classification scheme URI
0x00	reserved
0x01	urn:tva:metadata:cs:ActionTypeCS:2002
0x02	urn:tva:metadata:cs:AtmosphereCS:2002
0x03	urn:tva:metadata:cs:ContentAlertCS:2002
0x04	urn:tva:metadata:cs:ContentCommercialCS:2002
0x05	urn:tva:metadata:cs:ContentCS:2002
0x06	urn:tva:metadata:cs:FormatCS:2002
0x07	urn:tva:metadata:cs:HowRelatedCS:2002
0x08	urn:tva:metadata:cs:IntendedAudienceCS:2002
0x09	urn:tva:metadata:cs:IntentionCS:2002
0x0A	urn:tva:metadata:cs:LanguageCS:2002
0x0B	urn:tva:metadata:cs:MediaType:2002
0x0C	urn:tva:metadata:cs:OriginationCS:2002
0x0D	urn:mpeg:mpeg7:cs:RoleCS:2001
0x0E	urn:tva:metadata:cs:TVARoleCS:2002
0x0F to 0xEF	DVB Reserved
0xF0 to 0xFF	User Private

termID: The rank of the element in the classification Scheme, according to the document order defined in TS 102 822-3-1 [4], annex A. This termID is coded as a vluimsbf8 in order to support extension to the classification schemes. The rank of the first term according to document order shall be zero.

NOTE: The rank of an element should not be calculated before processing all import statements and ordering all imported elements.

Guidelines for extending classification schemes with new controlled terms, while maintaining compatibility with the existing classification schemes, are provided in annex F.

9.4.4 Forward compatibility

9.4.4.1 Use of forward compatible mode

A decoder shall be able to decode all information related to the schema identified by the urn "urn:tva:metadata:2002" and shall be able to skip information related to any schema extensions.

If the schema "urn:tva:metadata:2002" is extended, instance documents using those extensions shall be encoded in a forward compatible manner.

9.4.4.2 Overview (informative)

As defined in ISO/IEC 15938-1 [9], with some constraints, interoperability is provided between different versions of ISO/IEC 15938 schema definitions, without the full knowledge of all schema versions being required.

It is assumed that the updated version of a schema imports the previous version of that schema. Forward compatibility allows a decoder only aware of a previous version of a schema to partially decode a description conformant to an updated version of that schema.

Forward compatibility is ensured by a specific syntax defined in ISO/IEC 15938-1 [9], clauses 7 and 8. Its main principle is to use the namespace of the schema, i.e. the Schema URI, as a unique version identifier. The binary format allows one to keep parts of a description related to different schema in separate chunks of the binary description stream, so that parts related to unknown schema may be skipped by the decoder.

In order for this approach to work, an updated schema should not be defined using the ISO/IEC 15938-2 [20] "redefine" construct but should be defined in a new namespace. The Decoder Initialization identifies schema versions with which compatibility is preserved by listing their Schema URIs. A decoder that knows at least one of the Schema URIs will be able to decode at least part of the binary description stream.

In case an updated version of the schema is used, an element is coded in several version-consistent bitstream chunks i.e. ElementContentChunks. All elements in an ElementContentChunk are decoded using a single schema. A schema identifier is present before each ElementContentChunk. These identifiers are generated on the basis of URIs conveyed in the DecoderInit (see ISO/IEC 15938-1 [9], clause 7.2). A Length is present when the element is coded in several ElementContentChunks, allowing the decoder to skip ElementContentChunks related to unknown schema.

NOTE: The decoder keeps track of a SchemaModeStatus. It is used to improve coding efficiency. The decoder can "freeze" the schema needed to decode the description. In this case no overhead is induced by the multiple-version element coding for the elements contained in the element being decoded, i.e. the entire sub-tree.

9.4.4.3 Multiple version encoding of an element (informative)

Each XML element is associated to a type which defines its content model. Derived types are defined by restriction or extension of existing types. When managing different versions of a schema, a version 2 type might extend a version 1 type as shown in figure 15. In this case, a multiple-version coding can be used to provide a forward compatible coding of this element. For example, the type T2.6 can be coded in two ElementContentChunks. The first ElementContentChunk could encode those parts of T2.6 which were derived from T1.4 (see figure 16). Encoding would be done exactly as if it were type T1.4. The second ElementContentChunk then encodes the difference between types T1.4 and T2.6. A "Schema-1-decoder" will be able to decode the first part of the element content and skip the second part using the Length information. Figure 17 shows the same element encoded in a non forward compatible way.

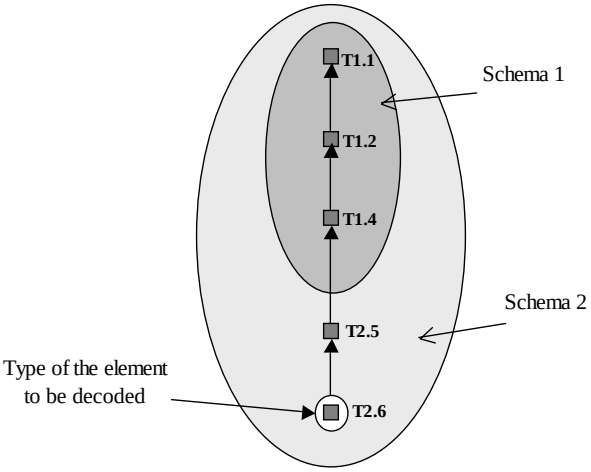

Figure 15: Example of a type hierarchy defined across versions

Figure 16: Example of a forward compatible encoding

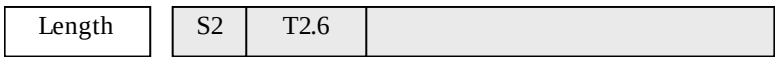

Figure 17: Example of a non forward compatible encoding

9.5 TV-Anytime structures

9.5.1 Profiled index structures

9.5.1.1 Introduction

TS 102 822-3-2 [5], clause 4.8.5 defines a means to index metadata delivered on a uni-directional network. If a metadata service is delivered indexing may be included that conforms to that specification. The present document defines a number of profiled indices; if a metadata service carries an index that is of the same type as one of the profiled indices defined in the present document, that is with the same fragment XPath and field XPath(s), then that index must conform to the profiled format defined herein. A metadata service may contain indices of other types not profiled in the present document, in which case those indices must conform to TS 102 822-3-2 [5].

The profiled indices are defined in the following way: structures are expressed in an informative, simplified form that removes options not used; normative values and options are defined. The structure definitions and field semantics are normatively defined in TS 102 822-3-2 [5], the profiled indices conform to that specification.

See clause 9.4.2.2 of the present document for creating indices for extended schemas.

9.5.1.2 Field identifier values

Table 66 defines the permitted values of the field identifier field in the index list structure.

Table 66: Allowed values of field_identifier field

Value	Equivalent field XPath expression
0x0000	"@tva:groupId"
0x0001	"tva:BasicDescription/tva:Title.text()"
0x0002	"@tva:programid"
0x0003	"@tva:end"
0x0004	"@tva:ServiceIDRef"
0x0005	"tva:ScheduleEvent/tva:InstanceDescription/tva:Title.text()"
0x0006 to 0x7FFF	DVB reserved
0x7FFF to 0xFFFE	user private
0xFFFF	Field Xpath expression is carried as a string

9.5.1.3 Index list

Table 67 is informative. It illustrates how the ListEntry profiled index structures defined in the present document fit into the index list structure, which is defined normatively in TS 102 822-3-2 [5].

If a receiver does not recognize the index definition for any entry in the index_list that entry shall be ignored.

Table 67: Index list

Syntax	Number of bits	Identifier	Value	Comment
index_list() {				
for (i=0; i<N; i++) {				
Either GroupInfoCridIndexListEntry()				entries for profiled indices
OR GroupInfoTitleIndexListEntry()				
OR ProgramInfoCridIndexListEntry()				
OR ProgramInfoTitleIndexListEntry()				
OR ScheduleTimeServiceIndexListEntry()				
OR ScheduleTitleIndexListEntry()				
OR {				generic index entry
index_descriptor_length	8	uimsbf	+ (see note 1)	
fragment_type	16	uimsbf	+	
if (fragment_type == 0xFFFF) {				
fragment_xpath_ptr	16			if fragment_type==0xFFFF this is ref. (see note 2) to XPath string
}				
num_fields	8	uimsbf	+	
for (i=0; i<num_fields; i++) {				
field_identifier	16	uimsbf	+	0xFFFF indicates use of W3C Xpath expression for field.
if (field_identifier == 0xFFFF) {				
field_xpath_ptr	16	uimsbf	* (see note 3)	if field_identifier==0xFFFF this is ref. to XPath string
}				
field_encoding	16	uimsbf	+	
}				
container_id	16	uimsbf	+	.
index_identifier	8	uimsbf	+	.
}				
}				
}				

NOTE 1: '+' indicates that the value is assigned (e.g. an identifier value).
NOTE 2: References to strings are all offsets from the start of the string_repository carried in the same container.
NOTE 3: '*' indicates that the value is calculated (e.g. a length or an offset value).

9.5.1.4 GroupInformation index by CRID

9.5.1.4.1 Index definition

An index of GroupInformation by CRID is defined by the XPath values in table 68.

Table 68: XPath values for GroupInformation index by CRID

	Value
fragment Xpath	/tva:TVAMain/tva:ProgramDescription/tva:ProgramInformationTable/tva:GroupInformation
field Xpath	@tva:groupId

The following structures define how an index of GroupInformation by CRID shall be constructed if present. These structures are compatible with the index and multi_field_sub_index structures defined by TS 102 822-3-2 [5].

9.5.1.4.2 Index list entry

Table 69 defines the index list entry that must be included in the index list structure if an index of this type is delivered.

Table 69: Index list entry for GroupInformation index by CRID

Syntax	Number of bits	Value	Comments
GroupInfoCridIndexListEntry() {			
index_descriptor_length	8	0x0C	
fragment_type	16	0x0002	indicates GroupInformation fragment
num_fields	8	0x01	single key index
field_identifier	16	0xFFFF	indicates use of W3C Xpath expression for field
field_xpath_ptr	16	*	ref. to string "@tva:groupId"
field_encoding	16	0x0000	indicates no encoding for field entries in GroupInfoCridIndex or GroupInfoCridSubIndex structures
container_id	16	+	the ID of the container carrying the GroupInfoCridIndex structure
index_identifier	8	+	the instance_id of the GroupInfoCridIndex structure
}			

9.5.1.4.3 Index structure

Table 70 defines the profile of index structure that must be used if an index of this type is delivered.

Table 70: Index structure for GroupInformation index by CRID

Syntax	Number of bits	Value	Description
GroupInfoCridIndex() {			
Overlapping_subindices	1	"0"	no overlapped indexing
Single_layer_sub_index	1	"0"	single layer only
Reserved	6	"111111"	
fragment_locator_format	8	0x01	remote fragment_locators
for (i=0; i<num_sub_indicies; i++) {			
high_field_value	16	*	ref. to CRID string
GroupInfo_sub_index_container	16	+	the ID of the container carrying the GroupInfoCridSubIndex structure
GroupInfo_sub_index_identifier	8	+	the instance_id of the GroupInfoCridSubIndex structure
}			
}			

9.5.1.4.4 Sub index structure

Table 71 defines the profile of multi_field_sub_index structure that must be used if an index of this type is delivered.

Table 71: Sub index structure for GroupInformation index by CRID

Syntax	Number of bits	Value	Description
GroupInfoCridSubIndex() {			
{			multi_field_header
leaf_field	1	"1"	Only one index layer so this is the leaf field
multiple_locators	1	"0"	fragment locators are in-line
Reserved	6	"111111"	
}			
for (j=0; j<num_entries;j++) {			repeat for each CRID indexed
field_value	16	*	ref. to GroupInformation CRID string
{			inline fragment locator structure referencing remote (see note) fragments
target_container	16	+	container carrying GroupInfo fragment
target_fragment	24	+	unique fragment ID
}			
}			
}			
NOTE: "remote fragments" are fragments not in the same container as the sub index structure. This format of fragment locator is the more flexible, but is not optimized for when data fragments are delivered in the same containers as the index structures.			

9.5.1.5 GroupInformation index by title

9.5.1.5.1 Index definition

An index of GroupInformation by title is defined by the XPath values in table 72.

Table 72: XPath values for GroupInformation index by title

	Value
fragment Xpath	/tva:TVAMain/tva:ProgramDescription/tva:ProgramInformationTable/tva:GroupInformation
field Xpath	tva:BasicDescription/tva:Title.text()

The following structures define how an index of GroupInformation by title shall be constructed if present. These structures are compatible with the index and multi_field_sub_index structures defined by TS 102 822-3-2 [5].

9.5.1.5.2 Index list entry

Table 73 defines the index list entry that must be included in the index list structure if an index of this type is delivered.

Table 73: Index List entry for GroupInformation index by title

Syntax	Number of bits	Value	Description
GroupInfoCridIndexListEntry() {			
index_descriptor_length	8	0x0C	
fragment_type	16	0x0002	indicates GroupInformation fragments
num_fields	8	0x01	single key index
field_identifier	16	0xFFFF	indicates use of W3C Xpath expression for field
field_xpath_ptr	16	*	ref. to string "tva:BasicDescription/tva:Title.text()"
field_encoding	16	0x0000	indicates no encoding for field entries in GroupInfoTitleIndex or GroupInfoTitleSubIndex structures
container_id	16	+	the ID of the container carrying the GroupInfoTitleIndex _index structure
index_identifier	8	+	the instance_id of the GroupInfoTitleIndex _index structure
}			

9.5.1.5.3 Index structure

Table 74 defines the profile of index structure that must be used if an index of this type is delivered.

Table 74: Index structure for GroupInformation index by CRID

Syntax	Number of bits	Value	Description
GroupInfoCridIndex() {			
Overlapping_subindices	1	"0"	no overlapped indexing
Single_layer_sub_index	1	"0"	single layer only
Reserved	6	"111111"	
fragment_locator_format	8	0x01	remote fragment_locators
for (i=0; i<num_sub_indicies; i++) {			
high_field_value	16	*	ref. to title string
GroupInfo_sub_index_container	16	+	the ID of the container carrying the GroupInfoTitleSubIndex structure
GroupInfo_sub_index_identifier	8	+	the instance_id of the GroupInfoTitleSubIndex structure
}			
}			

9.5.1.5.4 Sub index structure

Table 75 defines the profile of multi_field_sub_index structure that must be used if an index of this type is delivered.

Table 75: Sub index structure for GroupInformation index by title

Syntax	Number of bits	Value	Description
GroupInfoTitleSubIndex() {			
{			multi_field_header
leaf_field	1	"1"	Only one index layer so this is the leaf field
multiple_locators	1	"0"	fragment locators are in-line
Reserved	6	"111111"	
}			
for (j=0; j<num_entries;j++) {			repeat for each title indexed
field_value	16	*	ref. to GroupInformation title string
{			inline fragment locator structure referencing remote fragments
target_container	16	+	container carrying GroupInformation fragment
target_fragment	24	+	unique fragment ID
}			
}			
}			

9.5.1.6 ProgramInformation index by CRID

9.5.1.6.1 Index definition

An index of ProgramInformation by CRID is defined by the XPath values in table 76.

Table 76: XPath values for ProgramInformation index by CRID

	Value
fragment Xpath	/tva:TVAMain/tva:ProgramDescription/tva:ProgramInformationTable/tva:ProgramInformation
field Xpath	@tva:programid

The following structures define how an index of ProgramInformation by CRID shall be constructed if present. These structures are compatible with the index and multi_field_sub_index structures defined by TS 102 822-3-2 [5].

9.5.1.6.2 Index list entry

Table 77 defines the index list entry that must be included in the index list structure if an index of this type is delivered.

Table 77: Index List entry for ProgramInformation index by CRID

Syntax	Number of bits	Value	Description
ProgramInfoCridIndexListEntry() {			
index_descriptor_length	8	0x0C	
fragment_type	16	0x0002	indicates ProgramInformation fragment
num_fields	8	0x01	single key index
field_identifier	16	0xFFFF	indicates use of W3C Xpath expression for field
field_xpath_ptr	16	*	ref. to string "@tva:groupId"
field_encoding	16	0x0000	indicates no encoding for field entries in ProgramInfoCridIndex or ProgramInfoCridSubIndex structures
container_id	16	+	the ID of the container carrying the ProgramInfoCridIndex structure
index_identifier	8	+	the instance_id of the ProgramInfoCridIndex structure
}			

9.5.1.6.3 Index structure

Table 78 defines the profile of index structure that must be used if an index of this type is delivered.

Table 78: Index structure for ProgramInformation index by CRID

Syntax	Number of bits	Value	Description
ProgramInfoCridIndex() {			
Overlapping_subindices	1	"0"	no overlapped indexing
Single_layer_sub_index	1	"0"	single layer only
Reserved	6	"111111"	
fragment_locator_format	8	0x01	remote fragment_locators
for (i=0; i<num_sub_indicies; i++) {			
high_field_value	16	*	ref. to CRID string
ProgramInfo_sub_index_container	16	+	the ID of the container carrying the ProgramInfoCridSubIndex structure
ProgramInfo_sub_index_identifier	8	+	the instance_id of the ProgramInfoCridSubIndex structure
}			
}			

9.5.1.6.4 Sub index structure

Table 79 defines the profile of multi_field_sub_index structure that must be used if an index of this type is delivered.

Table 79: Sub index structure for ProgramInformation index by CRID

Syntax	Number of bits	Value	Description
ProgramInfoCridSubIndex() {			
{			multi_field_header
leaf_field	1	"1"	Only one index layer so this is the leaf field
multiple_locators	1	"0"	fragment locators are in-line
Reserved	6	"111111"	
}			
for (j=0; j<num_entries;j++) {			repeat for each CRID indexed
field_value	16	*	ref. to ProgramInformation CRID string
{			inline fragment locator structure referencing remote fragments
target_container	16	+	container carrying ProgramInfo fragment
target_fragment	24	+	unique fragment ID
}			
}			
}			

9.5.1.7 ProgramInformation index by title

9.5.1.7.1 Index definition

An index of ProgramInformation by title is defined by the XPath values in table 80.

Table 80: XPath values for GroupInformation index by title

	Value
fragment Xpath	/tva:TVAMain/tva:ProgramDescription/tva:ProgramInformationTable/tva:ProgramInformation
field Xpath	tva:BasicDescription/tva:Title.text()

The following structures define how an index of ProgramInformation by title shall be constructed if present. These structures are compatible with the index and multi_field_sub_index structures defined by TS 102 822-3-2 [5].

9.5.1.7.2 Index list entry

Table 81 defines the index list entry that must be included in the index list structure if an index of this type is delivered.

Table 81: Index List entry for ProgramInformation index by title

Syntax	Number of bits	Value	Description
ProgramInfoCridIndexListEntry() {			
index_descriptor_length	8	0x0C	
fragment_type	16	0x0002	indicates ProgramInformation fragments
num_fields	8	0x01	single key index
field_identifier	16	0xFFFF	indicates use of W3C Xpath expression for field
field_xpath_ptr	16	*	ref. to string "tva:BasicDescription/tva:Title.text()"
field_encoding	16	0x0000	indicates no encoding for field entries in ProgramInfoTitleIndex or ProgramInfoTitleSubIndex structures
container_id	16	+	the ID of the container carrying the ProgramInfoTitleIndex _index structure
index_identifier	8	+	the instance_id of the ProgramInfoTitleIndex _index structure
}			

9.5.1.7.3 Index structure

Table 82 defines the profile of index structure that must be used if an index of this type is delivered.

Table 82: Index structure for ProgramInformation index by title

Syntax	Number of bits	Value	Description
ProgramInfoCridIndex() {			
Overlapping_subindices	1	"0"	no overlapped indexing
Single_layer_sub_index	1	"0"	single layer only
Reserved	6	"111111"	
fragment_locator_format	8	0x01	remote fragment_locators
for (i=0; i<num_sub_indicies; i++) {			
high_field_value	16	*	ref. to title string
ProgramInfo_sub_index_container	16	+	the ID of the container carrying the ProgramInfoTitleSubIndex structure
ProgramInfo_sub_index_identifier	8	+	the instance_id of the ProgramInfoTitleSubIndex structure
}			
}			

9.5.1.7.4 Sub index structure

Table 83 defines the profile of multi_field_sub_index structure that must be used if an index of this type is delivered.

Table 83: Sub index structure for ProgramInformation index by title

Syntax	Number of bits	Value	Description
ProgramInfoCridSubIndex() {			
{			multi_field_header
leaf_field	1	"1"	Only one index layer so this is the leaf field
multiple_locators	1	"0"	fragment locators are in-line
Reserved	6	"111111"	
}			
for (j=0; j<num_entries;j++) {			repeat for each title indexed
field_value	16	*	ref. to ProgramInformation title string
{			inline fragment locator structure referencing remote fragments
target_container	16	+	container carrying ProgramInformation fragment
target_fragment	24	+	unique fragment ID
}			
}			
}			

9.5.1.8 Schedule index by time and DVB service

9.5.1.8.1 Index definition

An index of Schedule by time and DVB service is defined by the XPath values in table 84.

Table 84: XPath values for Schedule index by time and DVB service

	Value
fragment Xpath	/tva:TVAMain/tva:ProgramDescription/tva:ProgramLocationTable/tva:Schedule
first field Xpath	@tva:end
second field Xpath	@tva:ServiceIDRef

The following structures define how an index of Schedule by DVB service and time shall be constructed if present. These structures are compatible with the index and multi_field_sub_index structures defined by TS 102 822-3-2 [5].

The index of Schedule by time and DVB service is a two-layer index, indexing first by the value of the end time of the period covered by a Schedule fragment and second by the value of serviceIdRef.

9.5.1.8.2 Index list entry

Table 85 defines the index list entry that must be included in the index list structure if an index of this type is delivered.

Table 85: Index List entry for Schedule index by time and DVB service

Syntax	Number of bits	Value	Description
ScheduleTimeServiceIndexListEntry() {			
index_descriptor_length	8	0x12	
fragment_type	16	0x0002	indicates Schedule fragment
num_fields	8	0x02	two key index
field1_identifier	16	0xFFFF	indicates use of W3C Xpath expression for first key field
field1_xpath_ptr	16	*	ref. to string "@tva:end"
field1_encoding	16	0x0000	indicates no encoding for first key field entries in ScheduleTimeServiceIndex or ScheduleTimeServiceSubIndex structures
field2_identifier	16	0xFFFF	indicates use of W3C Xpath expression for second key field
field2_xpath_ptr	16	*	ref. to string "@tva:serviceIdRef"
field2_encoding	16	0x0000	indicates no encoding for second key field entries in ScheduleTimeServiceIndex or ScheduleTimeServiceSubIndex structures
container_id	16	+	the ID of the container carrying the ScheduleTimeServiceIndex structure
index_identifier	8	+	the instance_id of the ScheduleTimeServiceIndex structure
}			

9.5.1.8.3 Index structure

Table 86 defines the profile of index structure that must be used if an index of this type is delivered.

Table 86: Index structure for Schedule index by DVB service and time

Syntax	Number of bits	Value	Description
ScheduleCridIndex() {			
Overlapping_subindices	1	"0"	no overlapped indexing
Single_layer_sub_index	1	"0"	single layer only
Reserved	6	"111111"	
fragment_locator_format	8	0x01	remote fragment_locators
for (i=0; i<num_sub_indices; i++) {			
high_field_value1	16	*	ref. to serviceIdRef string
high_field_value2	16	*	ref. to date string
Schedule_sub_index_container	16	+	the ID of the container carrying the ScheduleCridSubIndex structure
Schedule_sub_index_identifier	8	+	the instance_id of the ScheduleCridSubIndex structure
}			
}			

9.5.1.8.4 Sub index structure layer 1

Table 87 defines the profile of multi_field_sub_index structure that must be used for the first layer of indexing if an index of this type is delivered.

The first layer sub index structure for a schedule index by time and DVB service lists all values of the "end" attribute for entries in the current container. A first layer sub index structure must be carried in the same container as the corresponding second layer sub index structure. Entries in this first layer sub index must be ordered by incrementing value of time.

Table 87: First layer sub index structure for Schedule index by DVB service and time

Syntax	Number of bits	Value	Description
ScheduleCridSubIndex1() {			
{			multi_field_header
leaf_field	1	"0"	this is not the last sub index layer
multiple_locators	1	"0"	fragment locators are in-line
Reserved	6	"111111"	
}			
child_sub_index_ref	8	+	The structure_id of the associate 2nd layer sub index structure
for (j=0; j<num_entries;j++) {			repeat for each DVB service indexed in this container
field_value	16	*	ref. to Schedule service string
range_end_offset	16	+	container carrying Schedule fragment
}			
}			

9.5.1.8.5 Sub index structure layer 2

Table 88 defines the profile of multi_field_sub_index structure that must be used for the second layer of indexing if an index of this type is delivered.

The second layer sub index structure for a schedule index by DVB service and time lists references to fragments, giving the value of ServiceIdRef of referenced fragments. Every first layer sub index structure must be carried in the same container as its corresponding second layer sub index structure. Entries in this second layer sub index relating to the same first layer sub index entry must be ordered by ascending lexicographical value.

Table 88: Second layer sub index structure for Schedule index by DVB service and time

Syntax	Number of bits	Value	Description
ScheduleCridSubIndex2() {			
{			multi_field_header
leaf_field	1	"1"	this is the last sub index layer
multiple_locators	1	"0"	fragment locators are in-line
Reserved	6	"111111"	
}			
for (j=0; j<num_entries;j++) {			repeat for each schedule fragment indexed in this container
field_value	16	*	ref. to end time string
{			inline fragment locator structure referencing remote fragments
target_container	16	+	container carrying Schedule fragment
target_fragment	24	+	unique fragment ID
}			
}			
}			

9.5.1.9 Schedule index by title

9.5.1.9.1 Index definition

An index of Schedule by title is defined by the XPath values in table 89.

Table 89: XPath values for Schedule index by title

	Value
fragment Xpath	/tva:TVAMain/tva:ProgramDescription/tva:ProgramLocationTable/tva:Schedule
field Xpath	tva:ScheduleEvent/tva:InstanceDescription/tva:Title.text()

The following structures define how an index of Schedule by title shall be constructed if present. These structures are compatible with the index and multi_field_sub_index structures defined by TS 102 822-3-2 [5].

NOTE: One Schedule fragment may contain more than one ScheduleEvent element, therefore each Schedule fragment will be referenced once for each ScheduleEvent it contains.

9.5.1.9.2 Index list entry

Table 90 defines the index list entry that must be included in the index list structure if an index of this type is delivered.

Table 90: Index List entry for Schedule index by Title

Syntax	Number of bits	Value	Description
`ScheduleCridIndexListEntry() {`			
`index_descriptor_length`	8	0x0C	
`fragment_type`	16	0x0002	indicates Schedule fragments
`num_fields`	8	0x01	single key index
`field_identifier`	16	0xFFFF	indicates use of W3C Xpath expression for field
`field_xpath_ptr`	16	*	ref. to string "tva:ScheduleEvent/tva:InstanceDescription/tva:Title.text()"
`field_encoding`	16	0x0000	indicates no encoding for field entries in ScheduleTitleIndex or ScheduleTitleSubIndex structures
`container_id`	16	+	the ID of the container carrying the ScheduleTitleIndex _index structure
`index_identifier`	8	+	the instance_id of the ScheduleTitleIndex _index structure
`}`			

9.5.1.9.3 Index structure

Table 91 defines the profile of index structure that must be used if an index of this type is delivered.

Table 91: Index structure for Schedule index by title

Syntax	Number of bits	Value	Description
`ScheduleCridIndex() {`			
`Overlapping_subindices`	1	"0"	no overlapped indexing
`Single_layer_sub_index`	1	"0"	single layer only
`Reserved`	6	"111111"	
`fragment_locator_format`	8	0x01	remote fragment_locators
`for (i=0; i<num_sub_indicies; i++) {`			
`high_field_value`	16	*	ref. to title string
`Schedule_sub_index_container`	16	+	the ID of the container carrying the ScheduleTitleSubIndex structure
`Schedule_sub_index_identifier`	8	+	the instance_id of the ScheduleTitleSubIndex structure
`}`			
`}`			

9.5.1.9.4 Sub index structure

Table 92 defines the profile of multi_field_sub_index structure that must be used if an index of this type is delivered.

Table 92: Sub index structure for Schedule index by title

Syntax	Number of bits	Value	Description
ScheduleCridSubIndex() {			
{			multi_field_header
leaf_field	1	"1"	Only one index layer so this is the leaf field
multiple_locators	1	"0"	fragment locators are in-line
Reserved	6	"111111"	
}			
for (j=0; j<num_entries;j++) {			repeat for each title indexed
field_value	16	*	ref. to Schedule title string
{			inline fragment locator structure referencing remote fragments
target_container	16	+	container carrying Schedule fragment
target_fragment	24	+	unique fragment ID
}			
}			
}			

9.5.2 Additional structures

9.5.2.1 Structure types

TS 102 822-3-2 [5], clause 4, defines a number of structure types for the purpose of carriage of TV-Anytime information. These structures shall be delivered by container, each structure having a structure_type and structure_id used to reference it from the container_header (see clause 4.5.2.1 of TS 102 822-3-2 [5]).

Structure types extending TS 102 822-3-2 [5], clause 4 are defined in the present document. The structure type and field of the container_header (see TS 102 822-3-2 [5], clause 4.5.2.1) shall be encoded as defined in table 93 if such a structure is included in a container.

Table 93: Structure type

Value	Meaning
0x00 to 0x7F	TVA structures
0x80	type list
0x81 to 0xAF	DVB reserved
0xB0 to 0xFF	private use

9.5.2.2 Type list

The type list structure is an extension to the TV-Anytime structures for carriage of metadata in unidirectional environments (see TS 102 822-3-2 [5], clause 4.5), its purpose is to define which containers carry certain types of metadata.

At most one type list structure may be carried in a metadata service. If present it shall be carried in the container with container ID equal to 0x0000. When the type list structure is present in a container, the structure_type and structure_id fields in the container_header (see TS 102 822-3-2 [5], clause 4.5.2.1) shall be set to 0x80 and 0x00, respectively.

This structure may contain entries for any number of types, but for each type there must be at most one entry. Additionally, for each entry, all containers in the current metadata service should be listed that carry fragments of the relevant type.

Table 94 defines the type list structure.

Table 94: Type list structure

Syntax	Number of bits	Identifier
type_list_structure() {		
num_types;	16	uimsbf
for (i=0; i<num_types; i++) {		
reserved	4	uimsbf
type_description_length	12	uimsbf
fragment_type	16	uimsbf
if (fragment_type == 0xFFFF) {		
fragment_xpath_ptr	16	uimsbf
}		
num_containers	8	uimsbf
for (j=0; j< num_containers; j++) {		
container_id	16	uimsbf
}		
}		
}		

num_types: This field shall be set to the number of fragment types listed in this structure.

reserved: This field shall be set to "1111".

type_description_length: This field shall be set to the number of bytes immediately following it in this fragment type entry.

fragment_type: This field identifies the type of fragment this entry pertains to. It shall be encoded according to the fragment_type field of the index list structure defined in clause 4.8.5.3 of TS 102 822-3-2 [5].

fragment_xpath_ptr: If the fragment_type is set to 0xFFFF then this field shall provide a reference to the XPath string that describes the fragment type for this entry. This reference shall be set to the offset, in bytes, from the start of the string repository in the current container to the first character of the fragment type XPath string.

num_containers: This field shall be set to the number of container_ids immediately following.

container_id: This field shall be set to the container_id of a container that carries at least one fragment that matches the fragment type of this entry.

10 Promotional links

10.1 Introduction

The purpose of promotional links is to provide the means to record material related to what the viewer is currently watching. For instance, if the viewer is currently watching a trailer for a film a promotional link can be used to give the viewer the opportunity to record the film.

Promotional links relating to a DVB service shall be carried by a Related Content Table (RCT), see clause 10.4 for the definition of the RCT. Each link consists of a reference (either a URI string which is likely to be a CRID, a DVB binary locator or both) and promotional text in at least one language. The format of this information is based on the tva:ExtendedRelatedMaterialType (see clause 10.2). The presence of a related content subtable for a particular service is indicated by the related content descriptor (see clause 10.3).

Promotional links are intended to be used in a real-time manner. While a promotional link is present for the current DVB service, a compliant receiver may make the viewer aware of the opportunity to book related material in some way. The receiver should present the viewer with the choice of related material, each item being described by the promotional text. When the viewer makes his selection the receiver should acquire the related content using the provided reference for the selected link. The subsequent removal of promotional links indicates that the receiver should cease to provide this opportunity to the viewer.

Figure 18 shows an example receiver implementation of promotional links. An actual receiver implementation may differ in many ways, for example by waiting for a short period before removing information from the screen.

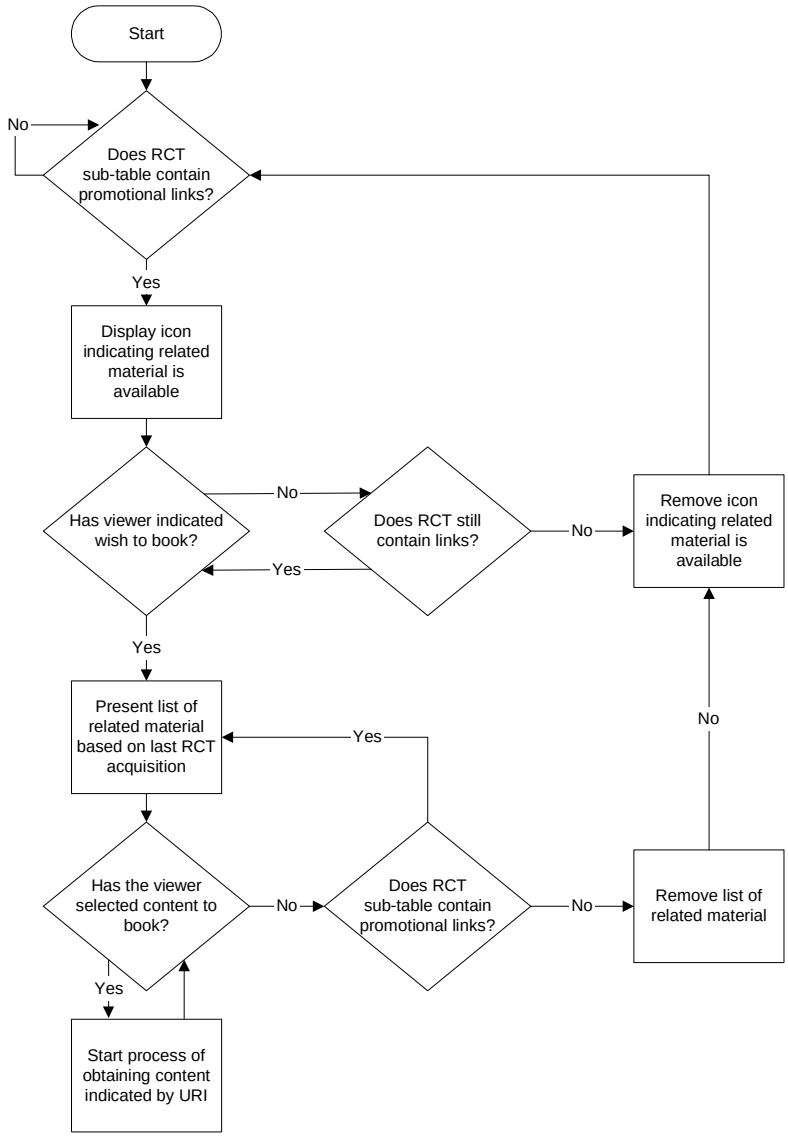

Figure 18: Example receiver implementation of promotional links (informative)

10.2 Restriction of tva:ExtendedRelatedMaterialType

The format for delivering promotional links is based upon a restriction of the tva:RelatedMaterialType type (see TS 102 822-3-1 [4], clause 6.3.4). Table 95 defines these restrictions.

It is not mandatory to provide metadata for CRIDs included in promotional links.

Table 95: Restrictions on types relevant to promotional links

Element	Restriction
mpeg7:MediaLocatorType	
MediaURI	One instance of this element is present
InlineMedia	Not present
StreamID	Not present
tva:RelatedMaterialType	
Format	Not present
PromotionalText	Present
SourceMediaLocator	Not present

10.3 Related content descriptor

The related content descriptor identifies an elementary stream that delivers a related content subtable. This descriptor may be carried in a PMT subtable in the descriptor loop for an elementary stream. Only one related_content descriptor is permitted in a single PMT subtable. The syntax of the related content descriptor is defined by table 96.

Table 96: Related content descriptor

Syntax	bits	format
related content descriptor() {		
descriptor_tag	8	uimsbf
descriptor_length	8	uimsbf
}		

descriptor_tag: This eight bit field shall be set to 0x74.

descriptor_length: This eight bit field shall be set to the number of bytes that follow it.

10.4 Related Content Table (RCT)

10.4.1 Description

The Related Content Table (RCT) carries promotional links related to the content currently being broadcast. Each subtable of the RCT relates to a single DVB service.

An RCT subtable is carried in an elementary stream whose PID is identified by the presence of a related content descriptor in the corresponding elementary stream descriptor loop of the current service''s PMT (see clause 10.3). The stream_type for an elementary stream carrying an RCT subtable must be set to 0x05, indicating private sections (see ISO/IEC 13818-1 [8], table 2-29).

When the related content descriptor is present in the PMT an RCT subtable shall be present and it shall be delivered at least once every two seconds. Receivers that are capable of using related material data may constantly monitor for its presence.

When an RCT subtable is obtained that has a link count greater than zero, the receiver should indicate to the viewer that there is the opportunity to book content. The way in which the user is prompted, or the conditions under which the list of available content is presented, are not part of the present document.

The time at which a related material is no longer being promoted is signalled by removing the specific link information from the related content table. A section with link_count set to zero shall be transmitted if there are currently no related materials.

10.4.2 Syntax

Each RCT subtable shall be delivered as sections as defined by table 97.

Table 97: Related content section

Syntax	Number of bits	format
related content section() {		
table_id	8	uimsbf
section_syntax_indicator	1	bslbf
reserved	1	bslbf
reserved_future_use	2	bslbf
section_length	12	uimsbf
reserved	16	uimsbf
reserved	2	bslbf
version_number	5	uimsbf
current_next_indicator	1	bslbf
section_number	8	uimsbf
last_section_number	8	uimsbf
year_offset	16	uimsbf
link_count	8	uimsbf
for (j=0; j<link_count; j++) {		
reserved	4	uimsbf
link_info_length	12	uimsbf
link_info()		
}		
CRC_32	32	rpchof
}		

table_id: This is an eight bit field that shall be set to 0x76.

section_syntax_indicator: This is a one-bit indicator which shall be set to "1".

reserved, reserved_future_use: All bits of fields marked as being reserved or reserved_future_use shall be set to "1".

section_length: This is a twelve bit field. It specifies the number of bytes of the section, starting immediately following the section_length field and up to the end of the section. The maximum allowed value of section_length is 4 093.

version_number: This 5-bit field is the version number of the subtable. The version_number shall be incremented by 1 when a change in the information carried within the subtable occurs. When it reaches value 31, it wraps around to 0.

current_next_indicator: This 1-bit indicator shall be set to "1".

section_number: This 8-bit field gives the number of the section. The section_number of the first section in the subtable shall be "0x00". The section_number shall be incremented by 1 with each additional section with the same table_id, context_id and context_id_type.

last_section_number: This 8-bit field indicates the number of the last section (that is, the section with the highest section_number) of the subtable of which this section is part.

year_offset: The year relative to which date values in this structure shall be calculated. This field shall be encoded as the binary value of the year, for example "2003" would be encoded as 0x07D3.

link_count: This is an eight bit field. It specifies the number of related material references in this clause.

CRC_32: This is a 32-bit field that contains the CRC value that gives a zero output of the registers in the decoder defined in annex B of EN 300 468 [1] after processing the entire section.

10.4.3 Link info structure

The syntax of the link info structure of the related content section is defined by table 98.

Table 98: Link info structure

Syntax	Number of bits	Identifier
link_info() {		
link_type	8	uimsbf
how_related	8	uimsbf
if (link_type == 0x00 || link_type == 0x02) {		
media_uri_length	8	
for (k=0; k<media_uri_length; k++) {		
media_uri_byte	8	uimsbf
}		
}		
if (link_type == 0x01 || link_type == 0x02) {		
dvb_binary_locator()		
}		
number_items	8	uimsbf
for (m=0; m<number_items; m++) {		
ISO 639-2_language_code	24	bslbf
promotional_text_length	8	uimsbf
for (n=0; n< promotional_text_length; n++) {		
promotional_text_char	8	uimsbf
}		
}		
reserved	4	uimsbf
descriptor_loop_length	12	uimsbf
for (p=0; p<descriptor_loop_length; p++) {		
descriptor()	8	uimsbf
}		
}		

link_type: This field indicates the format of the link information contained within this structure. This link information may consist of a URI (e.g. a CRID), a DVB binary locator or both a URI and a DVB binary locator.

If the link information consists of a CRID URI and a binary locator, the binary locator should be considered an indication of the likely broadcast time and location, the location the CRID resolves to should be considered the authoritative broadcast time and location. The semantics of the link information consisting of a different URI type and a binary locator are not defined.

This field shall be encoded according to table 99.

Table 99: link_type

Value	Description
0x00	Link information is a URI string only
0x01	Link information is a binary locator only
0x02	Link information is both a binary locator and a URI string

how_related: This is an eight bit field that defines the relationship between the content being broadcast and the content referenced by this link. This field shall be encoded according to table 100.

Table 100: how_related

Value	Description
0x00 to 0x1F	The numerical value of the TermID field in the urn:tva:metadata:cs:HowRelatedCS:2002 schema as defined in TS 102 822-3-1 [4], clause A.3.
0x20 to 0xAF	DVB reserved
0xB0 to 0xFF	User private

media_uri_length: If this field is present it shall be set to the number of media_uri_bytes that follow.

media_uri_byte: This field is part of a sequence that forms the MediaUri string. This string shall be a URI. If it is a CRID this string may use the abbreviated format (see clause 6.3). No other URIs may be abbreviated and must include the protocol part. The CRID authority part of the CRID may only be omitted if there is a default authority defined for the scope of the DVB service this related content section relates to.

dvb_binary_locator(): See clause 7.3.2.3.3 for definition. The year_offset value used by the DVB binary locator is defined in the enclosing related content section (see clause 10.4.2). The inline_service flag of the DVB binary locator shall be set to "1".

number_items: This field shall be set to the number of items in the following multilingual promotional text loop.

ISO 639-2_language_code: This field shall be set to the ISO 639-2 [21] code that describes the language of the following text field.

promotional_text_length: This field shall be set to the number of bytes in the following promotional text string.

promotional_text_char: This field is part of a sequence that forms the promotional text for the language given. Text information shall be encoded using the character sets and methods described in annex A of EN 300 468 [1].

descriptor_loop_length: This field shall be set to the total length in bytes of the following descriptors.

NOTE: The descriptor loop is included for future extension.

11 Accurate recording

11.1 Modes of operation

The DVB binary encoding for locators supports two modes for signalling the broadcast of an event. The mode in use is signalled by the identifier_type field of the DVB_binary_locator sub-structure (see clause 7.3.2.3.3).

The first mode is to use scheduled time and is signalled when the identifier_type is "00". In this case the receiver uses its internal clock to control the start and end of recording based on the times indicated by the start_time and duration fields. The start_time and duration fields offer the best estimate of when the content will be broadcast. Ideally, such CRI data should be updated to reflect any changes shortly before or during broadcast to provide more accurate information.

To offer increased reliability of capture a receiver may employ guard intervals either side of the scheduled start_time and implicit scheduled end time. In determining the size of these guard intervals the receiver may consider the early_start_window and late_end_window, if provided by the broadcaster for the particular item of content to be recorded. The intention is that the early_start_window and late_end_window fields should be considered an estimate of the maximum by which the broadcast time may differ from the scheduled time.

The scheduled time information provided in CRI, including the early_start_window and late_end_window, does not imply anything about previous and subsequent events. For example, the scheduled start of an event (from information in the CRI data) does not imply that the previous event on this service has finished.

The second mode relies on detecting the presence of event_identifiers in the broadcast stream and is signalled when the identifier_type is either "01" or "10". In its simplest form this mode relies on monitoring the presence of the relevant event_id in the present event of EIT p/f. In a refinement of this mode the receiver may monitor for the presence of the TVA_id, also carried in the present event of EIT p/f. The main advantage of the TVA_id over the event_id is that the service provider is free to control the presence of a TVA_id in the broadcast stream independently of the flow of EIT events. This allows the broadcaster to accurately identify when particular content is being delivered. In addition there can be more than one active TVA_id at any instance, enabling nesting and overlapping of content.

NOTE: One of the issues for a receiver implementation in providing support for the this mode is the strategy for how and when to look for the presence of the relevant event_identifier in the broadcast stream. An obvious candidate is to use the scheduled start_time and duration to define a "monitoring window". However, it should not be assumed that a service provider is always able to update an item of content's scheduled start time and duration before changes to the actual start time and duration occur. In extreme cases the programme may start before its scheduled start time or after it is scheduled to have finished. Despite this, the broadcast of either event_id and/or TVA_id may still enable accurate recording.

It is possible that a broadcast may contain information apparently supporting more than one of these modes for a particular event. For example, the service provider may provide start_time and duration even when signalling the use of an event_identifier simply to support clash detection during booking. However, there is no requirement to ensure consistency between this information and broadcast signalling should be used as a guide to the most reliable mode.

11.2 TVA_id descriptor

This clause defines the mechanism by which a TVA_id shall be carried in the broadcast stream, so creating an association with content currently being transmitted. The advantages of the use of the TVA_id for identifying content to record over the use of the EIT event_id are as follows:

- It allows the broadcaster to identify an item of content at a granularity that is finer than that of an EIT event.
- It allows broadcasters to indicate the actual transmission of an item of content more accurately than may be possible using EIT event_id.
- Since multiple TVA_ids may be present for a single DVB service it allows broadcasters to indicate the start and stop of a number of overlapping items of content.

The association of an item of content with a particular TVA_id is made within a DVB locator as carried in the CRI (see clause 7.3.2.3.3). The association of the same TVA_id with some specific content, and so by inference the item of content with this content, is made using the TVA_id_descriptor.

When carried, the TVA_id_descriptor shall be placed in the descriptor loop for the present event in the EIT present/following. Specific values of TVA_id shall be carried by a TVA_id_descriptor placed in the EIT present/following subtable for the DVB service in which the item of content is broadcast. Where a TVA_id is being signalled, placement of an appropriate TVA_id_descriptor in EIT present/following actual TS is mandatory, whilst placement of an appropriate TVA_id_descriptor in EIT present/following other TS is optional.

The TVA_id_descriptor allows a state to be associated with a specific value of TVA_id. This can be used by the receiver as part of its strategy of managing the recording process.

More than one TVA_id can be carried in a single instance of the TVA_id_descriptor and more than one instance of the TVA_id_descriptor can be placed in a single descriptor loop for the present event in EIT present/following. This enables the signalling of overlapping or nested events.

The syntax of the TVA_id descriptor is defined by table 101.

Table 101: TVA_id descriptor

Syntax	Number of bits	Identifier
`TVA_id_descriptor() {`		
`  descriptor_tag`	8	uimsbf
`  descriptor_length`	8	uimsbf
`  for (i=0; i<N; i++) {`		
`  TVA_id`	16	uimsbf
`  Reserved`	5	uimsbf
`  running_status`	3	uimsbf
`  }`		
`}`		

descriptor_tag: This 8-bit field shall be set to the value 0x75.

descriptor_length: This 8-bit field specifies the total number of bytes of the data portion of the descriptor following the byte defining the value of this field.

TVA_id: This 16-bit field shall be set to the value of TVA_id carried in the CRI locator that refers to an item of content current being broadcast.

running_status: This 3-bit field indicates the status of the item of content associated with a particular value of TVA_id. The possible values for this field are defined in table 102.

Table 102: Running status

Value	Meaning	Description
0	Reserved	
1	Not yet running	Receivers shall treat the item of content as not yet running. This can be used when the item of content is still to be broadcast, but is unlikely to start until some time after the most recently indicated scheduled start_time.
2	Starts (or restarts) shortly	Receivers shall prepare for the change of running_status to "running" to occur shortly. This optional mode can be used to assist receivers in preparing their resources for recording. If used this value should be signalled for 30 s before changing to "Running".
3	Paused	Receivers shall treat the item of content as paused. This can be used when broadcast of the item of content has already started, but at this time the content being broadcast is not a part it. It is assumed that the transmission of relevant content will resume at a later time. It is recommended that the paused state is only used for short interruptions not appearing in the schedule.
4	Running	Receivers shall treat the item of content as running. This can be used to indicate that at this time the content being broadcast is part of the item of content.
5	Cancelled	Receivers shall treat the item of content as cancelled. This can be used to indicate that the item of content has been pulled either before commencement of, or part way through transmission.
6 to 7	Reserved	

The relationship between the various states is illustrated by figure 19.

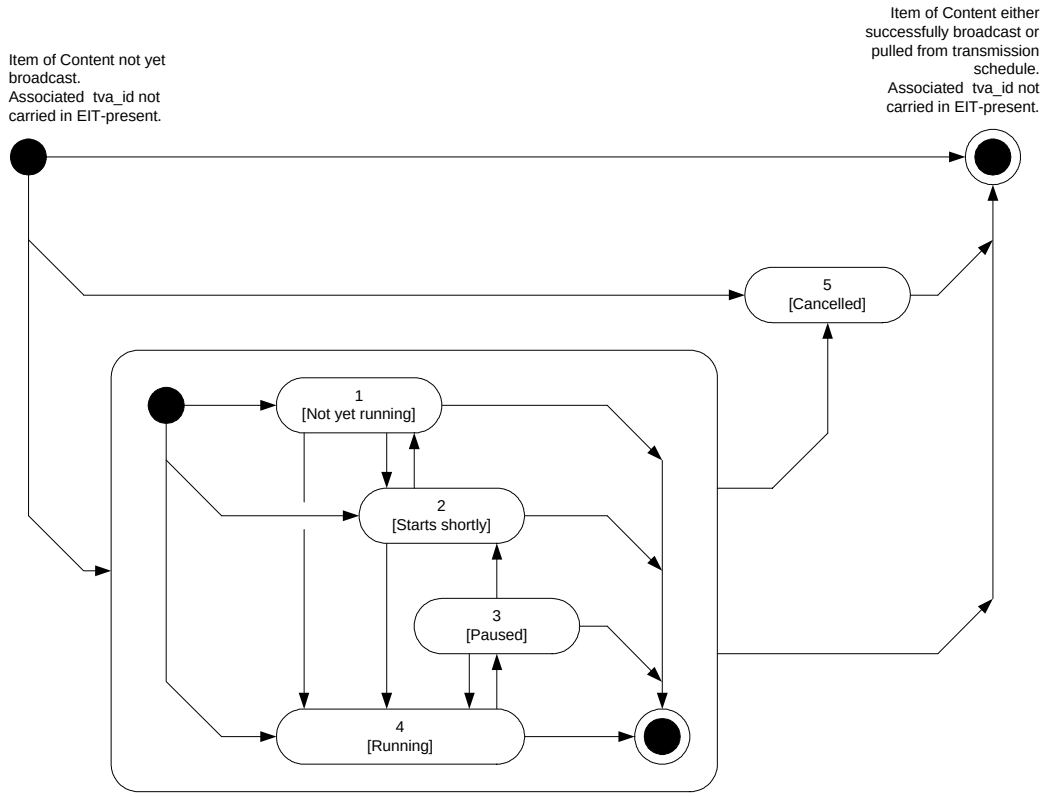

Figure 19: TVA_id UML state diagram

12 Extensions to DVB SI

12.1 Content identifier descriptor

12.1.1 Introduction

The content identifier descriptor allows a CRID to be assigned to an event entry in an EIT subtable, so providing a link to CRI or to metadata for that event's content. One or more instance of this descriptor may be carried in the event descriptor loop of an EIT schedule section or an EIT present/following section. There is no requirement for all EIT events to have a CRID assigned to them.

NOTE 1: The selection of events for recording using event entries in an EIT subtable that do not contain a CRID is not considered in the present document.

The content identifier descriptor supports two methods for defining the CRID to be associated, the CRID can be explicitly included in the descriptor or the descriptor can refer to a CRID carried in a separate subtable (an "indirect definition"). These methods of definition are interchangeable and each CRID included in a content identifier descriptor may be defined using any of these methods.

The use of explicit CRID definition is recommended for interoperability.

NOTE 2: A CRID is simply an identifier. The encodings supported by the content identifier descriptor are only provided to give the broadcaster some flexibility about how a particular CRID is defined. Once a receiver has extracted the CRID from a particular encoding it should simply treat this, like any other CRID, as just an identifier.

12.1.2 Explicit CRID definition

The first method is to explicitly carry the encoded CRID in the descriptor. The CRID may be encoded using the abbreviated CRID rules (see clause 6.3.1). If the CRID authority part is omitted there shall be a default authority defined for a scope encompassing the DVB service that the EIT relates to (see clause 6.3.2).

12.1.3 Indirect CRID definition

The second method is to refer to a CRID entry in a separate structure that is associated with the DVB service that the event forms part of. In this case the descriptor carries an identifier that uniquely identifies the relevant CRID in a subtable of the Content Identifier Table (CIT).

12.1.4 Syntax

The syntax of the content identifier descriptor is defined by table 103.

Table 103: Content identifier descriptor

Syntax	Number of bits	Identifier
content_identifier_descriptor() {		
descriptor_tag	8	uimsbf
descriptor_length	8	uimsbf
for (i=0;i<N;i++) {		
crid_type	6	uimsbf
crid_location	2	uimsbf
if (crid_location == "00") {		
crid_length	8	uimsbf
for (j=0;j<crid_length;j++) {		
crid_byte	8	uimsbf
}		
}		
if (crid_location == "01") {		
crid_ref	16	uimsbf
}		
}		
}		

descriptor_tag: This field shall be set to 0x76.

descriptor_length: This field shall be set to the number of bytes in this descriptor following this field.

crid_type: This field defines the type of CRID that this content labelling descriptor describes. This field shall be encoded according to table 104.

Table 104: CRID type

Value	Semantics
0x00	No type defined
0x01	CRID references the item of content that this event is an instance of
0x02	CRID references a series that this event belongs to
0x03	CRID references a recommendation. This CRID can be a group or a single item of content
0x04 to 0x1F	DVB reserved
0x20 to 0x3F	User private

crid_location: This field describes the location of the CRID information. This field shall be encoded according to table 105.

Table 105: CRID location

Value	Semantics
"00"	Carried explicitly within descriptor
"01"	Carried in Content Identifier Table (CIT)
"10"	DVB reserved
"11"	DVB reserved

crid_length: This field shall be set to the number of crid_bytes that follow.

crid_byte: This field forms part of a sequence of bytes that defines an explicitly encoded CRID (see clause 12.1.2).

NOTE: When the CRID authority part of the CRID URL is not present the forward-slash character ("/") immediately following the CRID authority part must be present.

This field may carry an IMI in addition to the CRID. In this case, the CRID is terminated by a "#" character which is followed immediately by the IMI. The first four bytes of the IMI (i.e. "imi:") shall be omitted.

crid_ref: When crid_location is set to "01" this field defines the identifier by which the relevant CRID can be discovered in the Content Identifier Table for this DVB service.

12.2 Content Identifier Table (CIT)

Event label data may be carried in CIT subtables comprised of content_identifier_sections. The format for this type of section is derived from the standard private section syntax as defined in ISO/IEC 13818-1 [7].

A CIT subtable provides CRID labels for EIT schedule subtables relating to one DVB service. One CIT schedule subtable is defined by a combination of service_id, transport_stream_id and original_network_id.

The semantics of the CIT are defined by table 106. Sections forming part of the CIT shall be carried in TS packets with a PID value of 0x0012.

Table 106: Content identifier section

Syntax	Number of bits	Identifier
Content_identifier_section() {		
table_id	8	uimsbf
section_syntax_indicator	1	bslbf
private_indicator	1	bslbf
reserved	2	bslbf
section_length	12	uimsbf
service_id	16	uimsbf
reserved	2	bslbf
version_number	5	uimsbf
current_next_indicator	1	bslbf
section_number	8	uimsbf
last_section_number	8	uimsbf
transport_stream_id	16	uimsbf
original_network_id	16	uimsbf
prepend_strings_length	8	uimsbf
for (i=0; i< prepend_strings_length ; i++) {		
prepend_strings_byte	8	uimsbf
}		
for (j=0; J<N; j++) {		
crid_ref	16	uimsbf
prepend_string_index	8	uimsbf
unique_string_length	8	uimsbf
for (k=0; k<unique_string_length; k++) {		
unique_string_byte	8	uimsbf
}		
}		
CRC32	32	
}		

table_id: This field shall be set to 0x77.

section_syntax_indicator: This shall be set to "1" to indicate that the private section follows the generic section syntax.

private_indicator: This flag shall be set to "1".

section_length: The number of remaining bytes in the private section immediately following the section_length field up to the end of this event label map section.

service_id: This shall contain the container_id of the container carried by the table this section is part of.

version_number: The version of the table.

current_next_indicator: This field shall be set to "1" to indicate that the section is currently valid.

section_number: This field specifies the number of the section. This section_number will be incremented by 1 with each additional section with the same service_id, transport_stream_id and original_network_id.

last_section_number: This specifies the number of the last section making up this table.

transport_stream_id: This field identifies the TS that carries the indicated DVB service.

original_network_id: This field identifies the network_id of the originating delivery system.

prepend_strings_length: This field gives the total number of bytes of prepend strings that follow this field.

prepend_strings_bytes: This field forms part of a sequence that is a concatenation of prepend strings. Each prepend string contained with this partition shall be partitioned by a byte with value 0x00. Prepend strings are referenced by index, the first prepend string having an index value of 0 and the second an index value of 1.

crid_ref: This field assigns a reference value for this CRID. This value is referenced from the content identifier descriptor of the EIT. Only one CRID may be assigned this value of crid_ref in all CIT sections with the same service_id and original_network_id.

prepend_string_index: This field gives the index of the relevant prepend_string in the list of prepend_strings carried in this section. If this field is set to 0xFF then there is no prepend_string and the unique_string shall hold the complete CRID string. The complete CRID string need not contain the first 7 characters common to every CRID (i.e. "CRID://"), if this is the case then their presence is implied.

unique_string_length: This field gives the length, in bytes, of the unique_string immediately following this field.

unique_string_byte: This field forms part of a sequence that together forms the unique_string. The unique_string shall not be null terminated. Concatenating the prepend_string for this entry with the unique_string gives a full CRID reference which may include an IMI. When an IMI is included the CRID shall be terminated by a "#" character which is followed immediately by the IMI. The first four bytes of the IMI ("imi:") are always omitted.

CRC32: This is a 32-bit field that contains the CRC value that gives a zero output of the registers in the decoder defined in ETSI 300 468 annex B after processing the entire private section.

Annex A (informative): Example recorder behaviour

Clause 11 specifies the different signalling that the broadcaster may provide for recording programme items based on the identifier_type field of the result_data CRI structure. This informative annex does not form a normative part of the present document but gives an illustrative example of a possible receiver implementation.

Figure A.1 shows these different modes of operation based on whether the identifier_type field is set to "not used", event_id or TVA_id. A receiver needs to be able to deal with a mixture of signalling modes present in a network as different broadcasters and programmes may support some modes and not others.

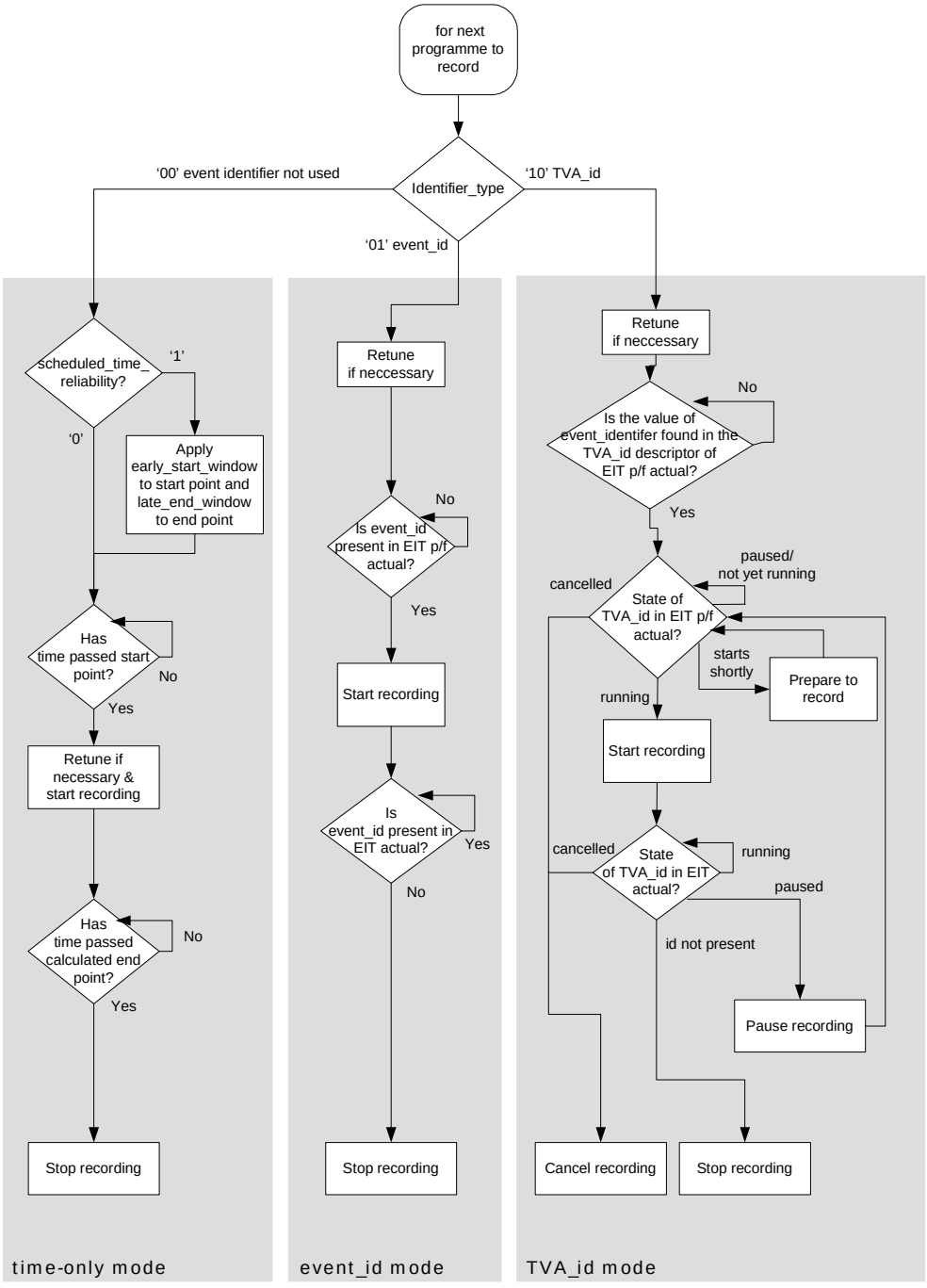

Figure A.1: Example strategy for controlling recording

However, this strategy does not represent the only solution. For example:

- If the actual programme transmission starts before the receiver has begun to monitor for a programme it will not be completely recorded.
- If two programmes are to be recorded "back-to-back" it may not be possible to blindly apply the early_start_window and late_end_window as provided by the broadcaster.
- In an alternative model a receiver may apply different adjustments to those suggested by the broadcaster or no adjustments at all.
- The "starts shortly" state of TVA_id signals to a receiver that it should prepare resources to start recording, this could also include increasing the rate of TVA_id monitoring.
- It does not indicate how the receiver should deal with an item in the list of programmes to record after the monitoring for its event_identifier was terminated by the receiver before starting recording, e.g. because it was time to record the next item in the list.
- This does not indicate what a receiver should do in the paused recording state, it may, for example, be able to record a different programme during the pause.

From this it should be clear that a range of strategies may be implemented based on the structure and size of the network, whether full cross-carriage of information is available and the physical resources of the receiver.

Annex B (informative): Example BiM format for ScheduleEvent fragment

B.1 TVA schedule schema

The following is a portion of the TVA Schema as defined in TS 102 822-3-1 [4]. It defines the Schedule type used in this example.

```
<element name="Schedule" type="tva:ScheduleType">

<complexType name="InstanceDescriptionType">
 <sequence>
  <element name="Title" type="mpeg7:TitleType" minOccurs="0"/>
  <element name="Synopsis" type="tva:SynopsisType" minOccurs="0"/>
  <element name="AVAttributes" type="tva:AVAttributesType" minOccurs="0"/>
 </sequence>
</complexType>

<complexType name="ScheduleEventType">
 <complexContent>
  <sequence>
   <element name="Program" type="tva:CRIDRefType"/>
   <element name="ProgramURL" type="anyURI" minOccurs="0"/>
   <element name="InstanceMetadataId" type="tva:InstanceMetadataIdType" minOccurs="0"/>
   <element name="InstanceDescription" type="tva:InstanceDescriptionType" minOccurs="0"/>
   <element name="PublishedStartTime" type="dateTime" minOccurs="0"/>
   <element name="PublishedEndTime" type="dateTime" minOccurs="0"/>
   <element name="PublishedDuration" type="duration" minOccurs="0"/>
   <element name="Live" type="tva:FlagType" minOccurs="0"/>
   <element name="Repeat" type="tva:FlagType" minOccurs="0"/>
   <element name="FirstShowing" type="tva:FlagType" minOccurs="0"/>
   <element name="LastShowing" type="tva:FlagType" minOccurs="0"/>
   <element name="Free" type="tva:FlagType" minOccurs="0"/>
  </sequence>
 </complexContent>
</complexType>

<attributeGroup name="fragmentIdentification">
 <attribute name="fragmentId" type="tva:TVAIDType" use="optional"/>
 <attribute name="fragmentVersion" type="unsignedLong" use="optional"/>
</attributeGroup>

<complexType name="ScheduleType">
 <sequence>
   <element name="ScheduleEvent" type="tva:ScheduleEventType" maxOccurs="unbounded"/>
 </sequence>
 <attribute name="serviceIDRef" type="tva:TVAIDRefType" use="required"/>
 <attribute name="start" type="dateTime" use="optional"/>
 <attribute name="end" type="dateTime" use="optional"/>
 <attributeGroup ref="tva:fragmentIdentification"/>
</complexType>
```

B.2 TVA dchedule instance: Textual coding

The following is a portion of a TV-Anytime instance document that describes a schedule fragment.

```
<Schedule   serviceIDRef="dvb://1234..0001"
        start="2003-03-04T12:00"
        end="2003-03-04T17:00">
 <ScheduleEvent>
  <Program crid="crid://1a2b3c4d" />
  <ProgramURL>dvb://1234..0001;1001@2003-03-04T14:01:30Z/PT00H58M15S</ProgramURL>
  <InstanceDescription>
   <Title>Footie</Title>
   <Synopsis>Kicking around a pig's bladder.</Synopsis>
  </InstanceDescription>
  <PublishedStartTime>2003-03-04T14:00:00</PublishedStartTime>
  <PublishedDuration>PT01H00M00S</PublishedDuration>
 </ScheduleEvent>
</Schedule>
```

B.3 TVA schedule instance: Binary coding

Table B.1 is an example encoding of the schedule fragment given in clause B.2.

Table B.1: Binary encoding of example schedule instance

Syntax	Value	Notes
FragmentUpdatePayload(){		
DecodingModes(){		
lengthCodingMode	00	No element lengths are encoded
hasDeferredNodes	0	No
hasTypeCasting	0	No
reservedBits	1111	
}		
Element(**Schedule**){	-	PayloadTopLevelElement so no transition to encode
Attributes(){		
Attr(**End**){	1	Optional attribute encoded
dvbDateTimeCodec()	10 0 [16 bits] [11 bits]	a Published Time is encoded don't reuse previous date date time (minutes since 00:00:00)
}		
Attr(**FragmentId**)	0	Optional attribute not encoded
Attr(**FragmentVersion**)	0	Optional attribute not encoded
Attr(**ServiceIDRef**){	-	Mandatory attribute
dvbStringCodec()	-	see dvbStringCodec(); data taken from string buffer
}		
Attr(**Start**){	1	Optional attribute is encoded
dvbDateTimeCodec()	10 1 [11 bits]	a Published Time is encoded reuse previously encoded date time (minutes since 00:00:00)
}		
}		
Content(){		ComplexContent
Sequence(){		
ScheduleEvent(){		
NumberOfOccurrences	0 0001	Unbounded in schema but only one in this instance
Element(**ScheduleEvent**){		
Attributes()		No attributes
Content(){		ComplexContent
Element(**Program**){	-	Mandatory element
Attributes(){		
Attr(**crid**){		Mandatory attribute

Syntax	Value	Notes
dvbDVBLocatorCodec(){		dvbLocatorCodec used since CRIDs are of type any URI. Data encoded
Optimized_codec_flag	0	CRID is encoded as a string
dvbStringCodec()	-	see dvbStringCodec(); data taken from string buffer
}		
}		
Content()		no content
}		
Element(**ProgramURL**){	1	Optional element is encoded
Attributes()	-	No attributes
Content(){		SimpleType
dvbDVBLocatorCodec(){		See dvbLocatorCodec
Optimized_codec_flag	1	Use the OptimizedDVBLocator encoding
	1	DVBLocatorPrefix is encoded
	[16 bits]	transport_stream_id
	[16 bits]	original_network_id
	[16 bits]	service_id
	0	No component_tag field encoded
	01	event_id is encoded
	[16 bits]	event_id data
	1	Time and date are encoded
	1	day is encoded
	[16 bits]	Date data
	[17 bits]	time data
	[17 bits]	duration data
	0	No path_segment data encoded
}		
}		
}		
Element(**InstMetadataId**)	0	Optional element not encoded, i.e. shunt transition
Element(**InstDescription**){	1	Optional element is encoded
Attributes()	-	No attributes
Content(){		ComplexContent
Sequence(){		
Element(**Title**){	1	Optional element is encoded
Attributes()		No attributes
Content(){		SimpleType
dvbStringCodec()	-	see dvbStringCodec(); data taken from string buffer.
}		
}		
Element(**Synopsis**){	1	Optional element is encoded
Attributes()		No attributes
Content(){		SimpleType
dvbStringCodec()	-	see dvbStringCodec(); data taken from string buffer
}		

Syntax	Value	Notes
}		
Element(**Genre**)	0	Optional element not encoded
Element(**AVAttributes**)	0	Optional element not encoded
Element(**MemberOf**)	0	Optional element not encoded
}		
}		
Element(**PublishedStartTime**){	1	Optional element is encoded
Attributes()	-	No attributes
Content(){		SimpleType
dvbDateTimeCodec()	10 1 [11 bits]	a Published Time is encoded reuse previously encoded date time (minutes since 00:00:00)
}		
}		
Element(**PublishedEndTime**)	0	Optional element not encoded
Element(**PublishedDuration**){	1	Optional element is encoded
Attributes()	-	No attributes
Content(){		SimpleType
dvbDurationCodec()	1 [11 bits]	optimized encoding number of minutes
}		
}		
Element(**Live**)	0	Optional element not encoded
Element(**Repeat**)	0	Optional element not encoded
Element(**FirstShowing**)	0	Optional element not encoded
Element(**LastShowing**)	0	Optional element not encoded
Element(**Free**)	0	Optional element not encoded
}		
}		
}		
}		
}		
}		
}		

The contents of the string buffer used by the dvbStringCodec for the above example is provided in table B.2. The characters are encoded according to UTF-8 and strings are terminated by a null terminator.

Table B.2: Content of the string buffer used by the dvbStringCodec() (UTF-8 encoding assumed)

String buffer content	String offset	Notes
"dvb://1234..0001"	0	Value of the serviceIDRef
"crid://1a2b3c4d"	17	CRID of the Program
"Footie"	33	Title of the program
"Kicking around a pig"s bladder"	40	Synopsis of the program

Annex C (informative): Example TVA-init and DecoderInit messages

C.1 Example TVA-init message

Table C.1 is an example TVA-init message that conforms to the profile specified in the present document.

Table C.1: Example TVA-init message

Field	Number of bits	Value	Notes
TVA-init {			
EncodingVersion	8	'0xF0'	DVB profile of BiM
IndexingFlag	1	0	No indexing used in the current TVA stream
reserved	7	1111111	
DecoderInitptr	8	5	Position of the decoderInit data from the beginning of the TVA-init message
{			/* EncodingVersion == "0xF0"*/
BufferSizeFlag	1	0	default buffer size for the ZlibCodec is used
PositionCodeFlag	1	0	position codes are not used
reserved	6	111111	
CharacterEncoding	8	'0x01'	UTF-8 Character Encoding
}			
Reserved	0 or 8+		
DecoderInit( )	8+	[data]	Decoder Initialization message
}			

C.2 Example decoderInit message

Table C.2 is an example DecoderInit message that conforms to the profile specified in the present document.

Table C.2: Example decoderInit message

Field	Number of bits	Value	Semantic
DecoderInit {			
SystemsProfileLevelIndication	16	'0x80'	arbitrary value
UnitSizeCode	3	000	default unit size
ReservedBits	5	11111	
NumberOfSchemas	8	'0x01'	Only one schema used: TV-Anytime
{			
SchemaURI_Length[0]	8	'0x15'	21 characters in the URI string
SchemaURI [0]		"urn:tva:metadata:2002"	
LocationHint_Length[0]	8	'0x00'	no location hint is provided
NumberOfTypeCodecs[0]	8	'0x00'	Only default codecs are used
}			
InitialDescription_Length	8	'0x00'	The initial root description is conveyed in the TVAMain fragment
}			

Annex D (informative): Example extension of the TVA Schema

D.1 Example extended schema

The following is an example schema that extends the TVA schema defined in TS 102 822-3-1 [4].

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema targetNamespace="urn:extended_schema:2003"
    xmlns:xs="http://www.w3.org/2001/XMLSchema"
    xmlns:mpeg7="urn:mpeg:mpeg7:schema:2001"
    xmlns:tva="urn:tva:metadata:2002"
    xmlns="urn:extended_schema:2003"
    elementFormDefault="qualified"
    attributeFormDefault="unqualified">
    <!-- ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ -->
    <!--        imports                                           -->
    <!-- ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ -->
    <xs:import  namespace="urn:mpeg:mpeg7:schema:2001"
                schemaLocation="mpeg7_tva.xsd"/>
    <xs:import  namespace="urn:tva:metadata:2002"
                schemaLocation="tva_metadata_v13.xsd"/>

    <!-- ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ -->
    <!--        TV-Anytime Extension                              -->
    <!-- ......................................................... -->
    <xs:complexType name="my_BasicContentDescriptionType">
        <xs:annotation>
            <xs:documentation>
                This is the extension of the tva:BasicContentDescriptionType
                which allows to provide the URL of a Logo for the program.
            </xs:documentation>
        </xs:annotation>
        <xs:complexContent>
            <xs:extension base="tva:BasicContentDescriptionType">
                <xs:sequence>
                    <element name="ProgramLogoURL" type="xs:anyURI" minOccurs="0"/>
                </xs:sequence>
            </xs:extension>
        </xs:complexContent>
    </xs:complexType>
</xs:schema>
```

D.2 Example decoderInit message for the extended schema

The main principle of forward compatibility is to use the namespace of the schemas, i.e. the schema URIs, as unique version identifiers. The identifiers are generated on the basis of URIs conveyed in DecoderInit (see clause 9.4.2.2).

Table D.1 provides an example of DecoderInit message for the extended TV-Anytime schema.

Table D.1: Example decoderInit message for an extended schema

Field	Number of bits	Value	Semantic
DecoderInit {			
SystemsProfileLevelIndication	16	'0x80'	arbitrary value
UnitSizeCode	3	000	default unit size
ReservedBits	5	11111	
NumberOfSchemas	8	'0x02'	Two schemas are used: TV-Anytime schema, and the extended schema
{			
/* schema 0: TVA */			
SchemaURI_Length[0]	8	'0x15'	21 characters in the URI string
SchemaURI [0]		"urn:tva:metadata:2002"	
LocationHint_Length[0]	8	'0x00'	no location hint is provided
NumberOfTypeCodecs[0]	8	'0x00'	Only default codecs are used
/* schema 1: extension */			
SchemaURI_Length[1]	8	'0x18'	24 characters in the URI string
SchemaURI [1]		"urn:extended_schema:2003"	
LocationHint_Length[1]	8	'0x00'	no location hint is provided
NumberOfTypeCodecs[1]	8	'0x00'	Only default codecs are used
}			
InitialDescription_Length	8	'0x00'	The initial root description is conveyed in the TVAMain fragment
}			

D.3 Example index XPaths for the extended schema

Clause 9.4.2.2 defines how the signalling of schemas shall be used to define namepace prefixes for use in index XPaths.

The following example XPath expressions describe an index of ProgramInformation fragments indexed by the element ProgramLogoURL of the extended type my_BasicContentDescriptionType:

Example fragment Xpath:

/tva:TVAMain/tva:ProgramDescription/tva:ProgramInformationTable/tva:ProgramInformation

Example field Xpath:

/tva:BasicDescription/d1:ProgramLogoURL.text()

Annex E (informative): Example Scenarios for encoding of TVA_id running_status as carried in EIT-present

E.1 Introduction

The following examples illustrate possible usage of the running_status field for TVA_ids carried in a TVA_id descriptor delivered in an EIT present/following subtable, as defined in clause 11.2.

E.2 Examples

E.2.1 Example 1

Figure E.1 illustrates a particular Item of Content (Y) has a TVA_id associated with it whilst running.

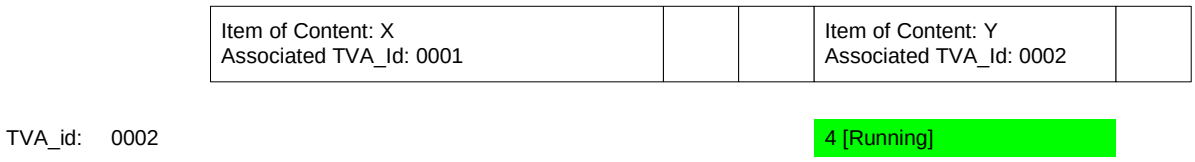

Figure E.1: Example 1

E.2.2 Example 2

Figure E.2 illustrates a number of Items of Content have a TVA_id associated with them whilst running.

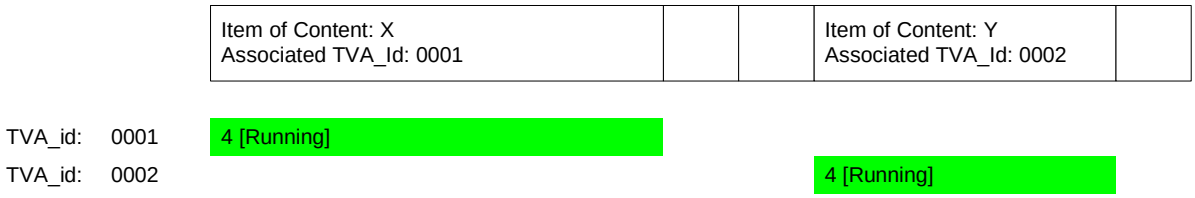

Figure E.2: Example 2

E.2.3 Example 3

Figure E.3 illustrates where start of Item of Content Y is proceeded by associated TVA_id with state "Starts shortly". This may or may not overlap with the previous Item of Content (X).

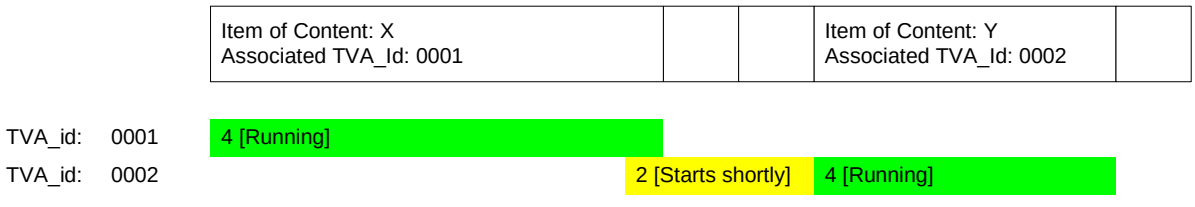

Figure E.3: Example 3

E.2.4 Example 4

In figure E.4, item of Content X overruns significantly compared to its scheduled start_time and duration. During the overrun the adjusted start_time for Item of Content Y may not yet be known and hence it may not be possible to update scheduled start_time. Transmitting the relevant TVA_id for Item of Content Y with the state "Not yet running" indicates that this content is still scheduled to be broadcast.

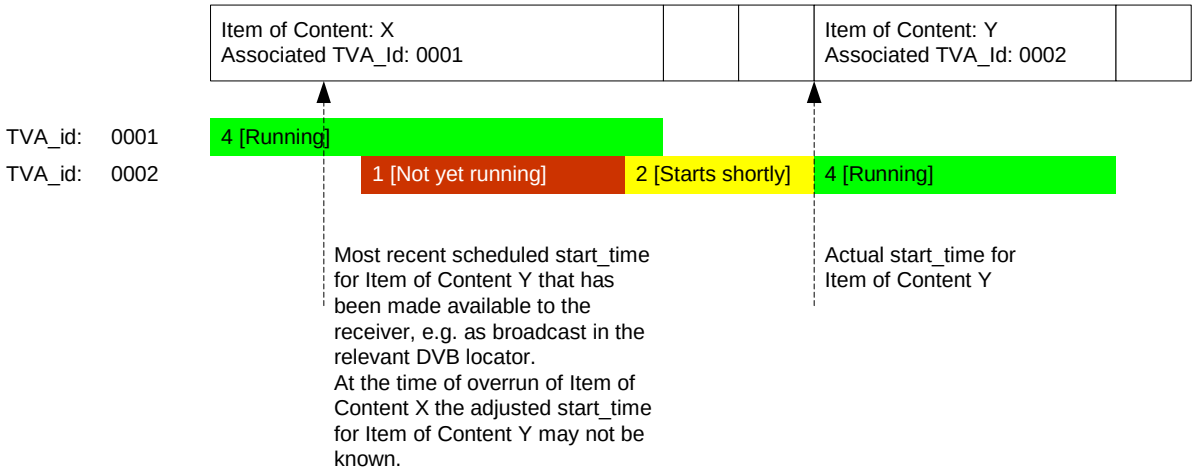

Figure E.4: Example 4

E.2.5 Example 5

In figure E.5, Item of Content X, e.g. a film, is split into two parts by Item of Content Y, e.g. a news flash.

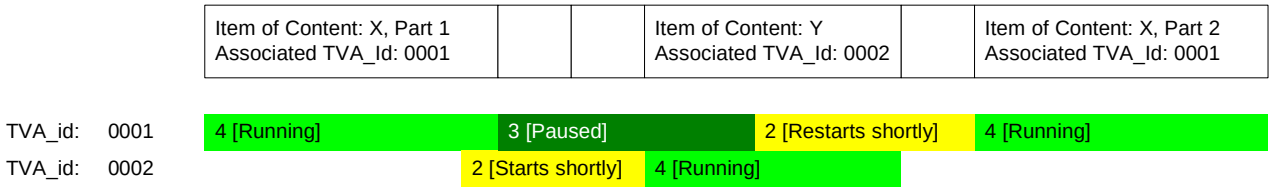

Figure E.5: Example 5

Annex F (normative): Classification schemes

F.1 Encoding

This annex describes the implementation of and the use of the dvbControlledTermCodec codec defined in the present document.

As described in clause 9.4.3.7, this codec encodes a reference to a term defined in a classification scheme by encoding the ID of the classification scheme and the rank of the term within that scheme.

In the case of the TVARoleCS classification scheme, one should note that the definition of the scheme begins by importing the terms of the RoleCS MPEG-7 classification scheme. This is equivalent to copying the definitions of all the terms in RoleCS into the beginning of the TVARoleCS.

As a consequence, every term defined in the RoleCS, for example, is also defined in the TVARoleCS and can therefore be referenced by two different URIs (one referencing the RoleCS and the other the TVARoleCS). Care must be taken when identifying the rank of terms defined in TVARoleCS but not imported from RoleCS because the imported terms must be enumerated first.

Table F.1 illustrates this by providing the correct ranks for some of these terms.

Table F.1: Rank of terms defined in TVARoleCS and RoleCS classification schemes

controlled term URI	name	Encoded Classification Scheme ID	Encoded term rank
urn:mpeg:mpeg7:cs:RoleCS:2001:**AUTHOR**	Author	0x0D	0
urn:mpeg:mpeg7:cs:RoleCS:2001:**UNKNOWN**	Unknown	0x0D	54
urn:tva:metadata:cs:TVARoleCS:2002:**AUTHOR** (see note 1)	Author	0x0E	0
urn:tva:metadata:cs:TVARoleCS:2002:**UNKNOWN** (see note 1)	Unknown	0x0E	54
urn:tva:metadata:cs:TVARoleCS:2002:**V708** (see note 2)	Dubber	0x0E	55
urn:tva:metadata:cs:TVARoleCS:2002:**V494** (see note 2)	Production Secretary	0x0E	143
NOTE 1: These terms are imported from the RoleCS. NOTE 2: These terms are defined in TVARoleCS but are ranked after imported terms.			

F.2 Extension

F.2.1 Introduction

The definition of classification schemes extensions with new controlled terms shall be done by defining a new classification with the new controlled term.

The URI of the new classification scheme is user defined.

The termID assigned to the new controlled terms should be given the same value as if the terms had be appended to the actual classification scheme being extended.

Also this is perfectly valid since the extension has its own URI, no termIDs from the original classification scheme should be re-used for the newly defined controlled terms.

F.2.2 Example extension

The following classification scheme is extending the classification whose URI is urn:tva:metadata:cs:ContentCS:2002 with two new controlled terms.

```
<ClassificationScheme uri="urn:broadcaster:ContentCSExtension">
    <!-- ########################################################## -->
    <!-- Extension to urn:tva:metadata:cs:ContentCS:2002          -->
    <!-- Definition: This classification scheme defines two types -->
    <!-- of programmes for the Leisure/Hobby categories (termID 3.3)-->
    <!-- ########################################################## -->
    <Term termID="3.3.35">
        <Name xml:lang="en">Internet/web</Name>
        <Definition xml:lang="en">Programme about subject on the
            world-wide-web and related technologies</Definition>
    </Term>
    <Term termID="3.3.36">
        <Name xml:lang="en">Cyberculture</Name>
        <Definition xml:lang="en">Programme about cyberculture (art,
        trends, people,...)</Definition>
    </Term>
</ClassificationScheme>
```

Annex G (informative): Bibliography

- ETSI ETR 162: "Digital Video Broadcasting (DVB); Allocation of Service Information (SI) codes for DVB systems".
- W3C Recommendation: "XML Schema Part 1: Structures". http://www.w3.org/TR/xmlschema-1/.

History

Document history		
V1.1.1	September 2004	Publication