ETSI TS 102 832 V1.2.1 (2010-01)

Technical Specification

Digital Video Broadcasting (DVB); IP Datacast over DVB-H: Notification Framework

Reference
RTS/JTC-DVB-275

Keywords
digital, DVB, mobile, 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

The Digital Video Broadcasting Project (DVB) is an industry-led consortium of broadcasters, manufacturers, network operators, software developers, regulatory bodies, content owners and others committed to designing global standards for the delivery of digital television and data services. DVB fosters market driven solutions that meet the needs and economic circumstances of broadcast industry stakeholders and consumers. DVB standards cover all aspects of digital television from transmission through interfacing, conditional access and interactivity for digital video, audio and data. The consortium came together in 1993 to provide global standardisation, interoperability and future proof specifications

Introduction

IP Datacast over DVB-H is an end-to-end broadcast system for delivery of any types of digital content and services using IP-based mechanisms optimized for devices with limitations on computational resources and battery. An inherent part of the IPDC system is that it comprises of a unidirectional DVB broadcast path that may be combined with a bi-directional mobile/cellular interactivity path. IPDC is thus a platform that can be used for enabling the convergence of services from broadcast/media and telecommunications domains (e.g. mobile/cellular).

1 Scope

The present document defines mechanisms for the delivery of messages which are used by the IPDC over DVB-H network to provide information about forthcoming events; such messages are referred to as Notification messages.

The present document specifies the message format, transport and access mechanisms of Notification messages.

2 References

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.
- Non-specific reference may be made only to a complete document or a part thereof and only in the following cases:
  - if it is accepted that it will be possible to use all future changes of the referenced document for the purposes of the referring document;
  - for informative references.

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

NOTE: While any hyperlinks included in this clause were valid at the time of publication ETSI cannot guarantee their long term validity.

2.1 Normative references

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

[1] ETSI TS 102 472: "Digital Video Broadcasting (DVB); IP Datacast over DVB-H: Content Delivery Protocols".

[2] ETSI TS 102 471: "Digital Video Broadcasting (DVB); IP Datacast over DVB-H: Electronic Service Guide (ESG)".

[3] IETF RFC 2387: "The MIME Multipart/Related Content-type".

[4] IETF RFC 3550: "RTP: A Transport Protocol for Real-Time Applications".

[5] IETF RFC 1952: "GZIP file format specification version 4.3".

[6] IETF RFC 4574: "The Session Description Protocol (SDP) Label Attribute".

[7] IETF RFC 2616: (June 1999): "Hypertext Transfer Protocol -- HTTP/1.1".

[8] OMA Push V2.2: "Push Access Protocol".

NOTE: See http://www.openmobilealliance.org/Technical/release_program/push_v2_2.aspx, Open Mobile Alliance, October 2007.

[9] OMA OMNA Registered PUSH Application ID list.

NOTE: See http://www.openmobilealliance.org/Tech/omna/omna-push-app-id.aspx.

[10] HTML 4.01 Specification, W3C Recommendation, 24 December 1999.

NOTE: see http://www.w3.org/TR/html401.

[11] OMA Smartcard Web Server V1.0.

NOTE: See http://www.openmobilealliance.org/Technical/release_program/scws_v1_0.aspx, Open Mobile Alliance, April 2008.

[12] OMA Smartcard Web Server V1.1.

NOTE: See http://www.openmobilealliance.org/Technical/release_program/scws_v1_1.aspx, Open Mobile Alliance, May 2009.

2.2 Informative references

The following referenced documents are not essential to the use of the present document but they assist the user with regard to a particular subject area. For non-specific references, the latest version of the referenced document (including any amendments) applies.

[i.1] ETSI EN 302 304: "Digital Video Broadcasting (DVB); Transmission System for Handheld Terminals (DVB-H)".

[i.2] ETSI TS 102 468: "Digital Video Broadcasting (DVB); IP Datacast over DVB-H: Set of Specifications for Phase 1".

[i.3] ETSI TR 102 469: "Digital Video Broadcasting (DVB); IP Datacast over DVB-H: Architecture".

[i.4] IETF RFC 3926: "FLUTE - File Delivery over Unidirectional Transport".

3 Definitions and abbreviations

3.1 Definitions

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

channel: end-to-end path over which the data composing the Service Component is delivered

Default Notification Channel: channel from which a terminal can retrieve Notification Messages without prior user selection

NOTE: A Default Notification Channel may require registration to a server when accessed over an interactive network.

ESG Default Notification Channel: Default Notification Channel delivering to terminals Notification Messages related to Services signalled in the ESG

In-band Service Related Notification Channel: channel delivering Service Related Notification Messages that may need to consumed by the terminal concurrently with the other Service Components composing the Service

Network Default Notification (NDN) channel: Default Notification Channel delivering Notification Messages to terminals attached to the network

notification: act of transmitting a Notification Message

notification channel: end-to-end path over which the data (e.g. the notification messages) composing one or more Notification Service Components is delivered

notification Message: information about a forthcoming event that some entity wishes to promptly communicate to terminals or users

NOTE: A Notification message is composed of a generic message part, and a payload, itself composed of an application specific message part and possibly media objects.

Notification Service (NS): service composed exclusively of one or more Notification Service Components

NOTE: A Notification Service may be a component of another service.

notification service component: service component carrying notification messages

Out-of-band Service Related Notification Channel: channel delivering Service Related Notification Messages that can be consumed by the terminal independently from the other Service Components composing the Service

Platform Default Notification Channel: Default Notification Channel delivering Notification Messages to terminals attached to an IP platform

registration: procedure by which a terminal becomes listed to use a service

Service Component: element of a service characterized by the nature of data it carries

NOTE: A service is composed of one or more Service Components (e.g. a Video component, one or more Audio components, etc.).

Subscription: agreement between a user and a service provider on the access to a service

NOTE 1: The Service Definition assumes that such an agreement can be made, but does not prescribe the method by which it is made.

NOTE 2: A user making a subscription may have its terminal automatically registered or may proceed to a registration step. The successful completion of the registration step may not require any subscription.

User-selected Notification Service: Notification Service that the User discovers from the ESG and that the User may need to subscribe to in order to retrieve the Notification Messages

3.2 Abbreviations

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

CBMS	Convergence of Broadcast and Mobile Services
DVB	Digital Video Broadcasting
DVB-H	DVB-Handheld
EDN	ESG Default Notification
ESG	Electronic Service Guide
FDT	File Delivery Table
FLUTE	File Delivery over Unidirectional Transport
ID	IDentifier
IP	Internet Protocol
IPDC	IP Datacast
iSRN	In-band Service Related Notification
MIME	Multipurpose Internet Mail Extensions
MTU	Maximum Transmission Unit
NDN	Network Default Notification
NIC	Notification Init Container
NPF	Notification Payload Format
NS	Notification Service
OMA	Open Mobile Alliance
OMNA	OMA Naming Authority
oSRN	out-of-band Service Related Notification
OTA	Over The Air
PDN	Platform Default Notification
RFC	Request For Comments
RTCP	RTP Control Protocol

RTP	Real-time Transport Protocol
SCWS	Smart Card Web Server
SDP	Session Description Protocol
SI	Service Information
SRN	Service Related Notification
UDP	User Datagram Protocol
WSP	Wireless Session Protocol

4 Overview

Notification is a function by which the network provides messages about forthcoming and not predictable events of interest to the terminal or the user. The Notification may lead to subsequent interaction from the user/the terminal.

The information carried in the Notification messages can be related to the (DVB) network supporting the IPDC system, the IP platform, or the services described in a given ESG.

The Notification messages are delivered by broadcast networks to terminals. In addition, for terminals with Interaction capability, these messages can be delivered to or retrieved by the terminals using an Interaction network. Also, an Interaction network could deliver Notification messages to complement messages that are delivered via Broadcast network.

4.1 Categories of Notification

Based on use case analysis, several Notification delivery mechanisms have been identified, leading to different methods to access them.

4.1.1 Default Notification

A Default Notification can be automatically accessed by terminals using the Default Notification Channel in a Broadcast network and should be able to receive these messages without any particular subscription. Registration may be required when the default Notification messages are available over the Interaction network.

Three Default Notification categories have been identified, depending on the scope of the messages attached to the category:

- Network Default Notification.
- Platform Default Notification.
- ESG Default Notification.

4.1.1.1 Network Default Notification (NDN)

The NDN category includes messages that are relevant at the DVB network level. No user selection is required. All terminals attached to the network can access the messages. Notification messages related to the network are carried over the PDN channels.

Messages in NDN category are agnostic to the services described in the ESG. Examples of such messages could relate to emergency such as Amber alerts, interruption of broadcast, change in network connection parameters, etc.

4.1.1.2 Platform Default Notification (PDN)

The PDN category includes messages that are relevant in a particular IP Platform. All terminals attached to that IP Platform can receive the messages without user selection. Messages in PDN category are agnostic to the services described in the ESG. Examples of such messages could relate to change of platform configuration.

4.1.1.3 ESG Default Notification (EDN)

The EDN category includes messages that are related to the services described by a given ESG. All terminals using that ESG of a specific ESG provider can receive the EDN messages automatically without user selection.

Such Notification messages can be delivered in (one of) the ESG carousels or in a dedicated channel. EDN messages can relate to the ESG provider only (e.g. Notification of a new service available) and/or to services described in the ESG (e.g. special event occurring in a service).

4.1.2 User-selected Notification

User-selected Notifications are messages related to services discovered in the ESG. They are user selected in the sense that the user has to discover and select a service and related Notification components via the ESG.

4.1.2.1 Service Related Notification (SRN)

Service Related Notifications are messages related to a specific service described in an ESG (e.g. mobile TV channel). Such Notification messages, when delivered as part of the service session (and preferably in the same DVB-H burst) are referred to as in-band Service Related Notification messages (iSRN).

When in-band SRN messages need to be tightly synchronized to the service which it refers to, the RTP protocol is used to provide synchronization information as it allows for accurate synchronization of the different media components of the service session. Service Related Notification messages can also be carried in the EDN session, for terminals to be able to receive messages without the need to tune into a particular service.

SRN may require subscription, in addition to subscription to the related service.

4.1.2.2 Notification Service (NS)

A Notification Service is a service consisting in the delivery of Notification messages for consumption by the user. An example of such service is a News service. Such service is described in the ESG and discoverable by the user while searching the ESG. A NS service may require subscription and purchase.

4.2 Notification Framework

The Notification framework is the system configuration enabling the delivery of Notification messages according to the categories described in clause 4.1. Figure 1 depicts an instantiation of such framework.

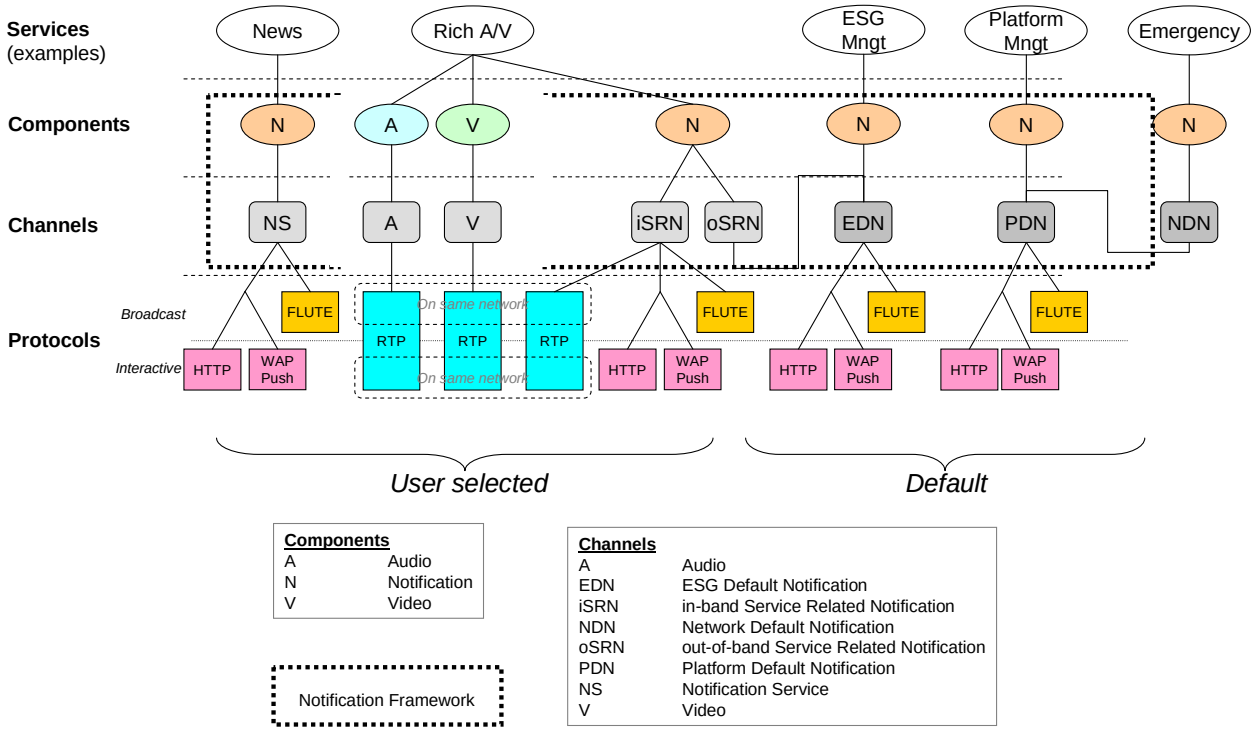

Figure 1: Example of Notification framework configuration

Services may have Notification components to deliver Notification messages. Depending on the scope of the Notification message, a Notification channel is selected to deliver the messages. The Notification channels map on protocols suited for the transport requirements of the Notification messages.

When messages are delivered over the Interaction Channel, the messages are either pushed to individual terminals or retrieved (Polling) by the terminal irrespective of the type of Notification message.

When messages are expected to have tight timing relationship with A/V components (hence using the RTP protocol), they are expected to be delivered over the same bearer, i.e. over the Broadcast network if the A/V components are over Broadcast, over the Interaction network if the A/V components are over Interaction.

The discovery method of a Notification channel is dependent on its scope.

Default Notification channels are discovered from the bootstrap channel used to discover the ESGs. Notification channels delivering messages related to services described in the ESG are discovered via the ESG, as any channel related to a service described in the ESG.

The only exception to this principle is for out-of-band Service-related Notification messages. Such messages may be delivered in the EDN channel when they may be consumed independently of the service they are related to.

Figure 2 depicts the discovery paths of the various Notification channels.

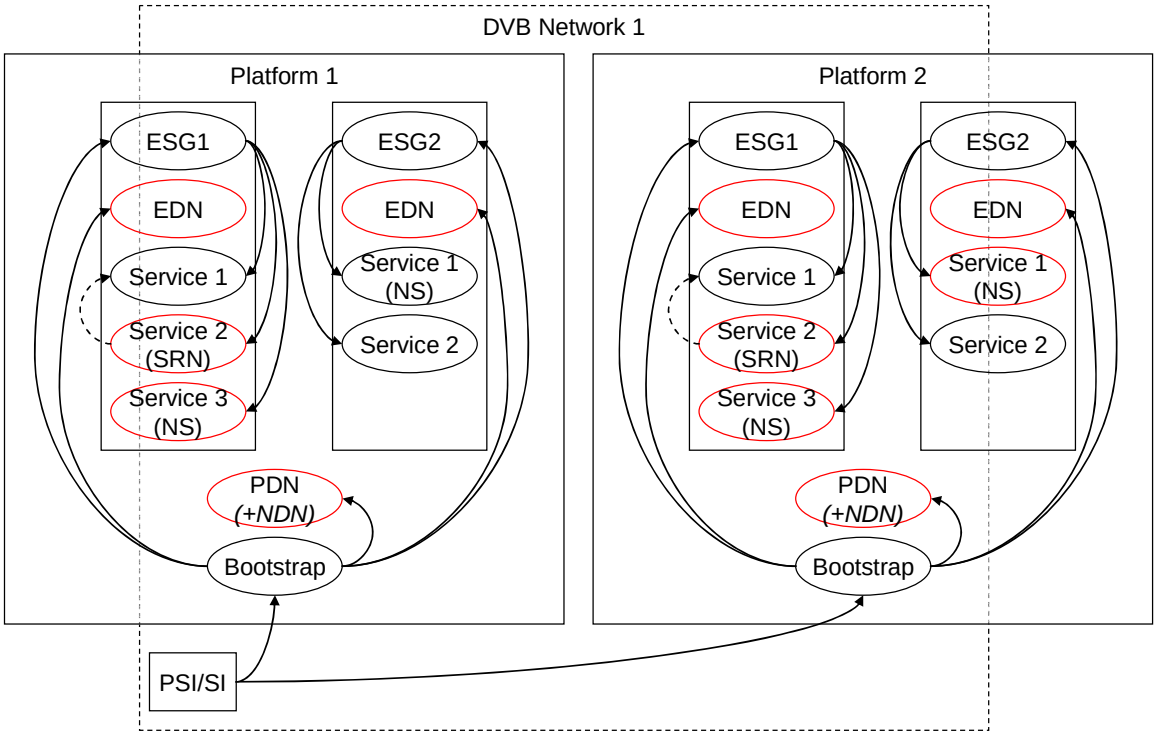

Figure 2: General Notification organization

When a terminal connects to a DVB network transporting an IP platform, it first acquires the DVB signalling (PSI/SI). PSI/SI enables the terminal to locate the bootstrap channel for a given IP platform.

Once the descriptors in the bootstrap channel are acquired, the terminal can locate the available PDN channel as well as the EDN channel associated to each ESG present in the IP platform. The bootstrap descriptors may provide entry point for the PDN and EDN channels over an Interaction network.

NOTE: The PDN may also carry NDN messages.

From the ESG, the terminal can discover IPDC services that can be "regular" IPDC services (e.g. an A/V service), Notification Services (NS) and Service Related Notification (SRN) components described as IPDC services. Such SRN components can be related to a regular IPDC service; the relationship between the SRN component and the base regular service is discovered in the SRN service fragments or in the Notification message itself. Thus, IPDC phase 1 terminals should not be impacted as they would ignore such SRN services.

The discovery of access information for the delivery of the Notification messages via Interaction network depends on the nature of the Notification. For Default Notification messages, the access information is extracted from the access descriptor in the bootstrap channel for the given IP platform over the Broadcast network. For User Selected Notification, the access information is provided in the ESG along with "regular" IPDC services.

4.3 Mapping of Notification messages on transport layer

4.3.1 Structure of Notification messages

A Notification Message is an extensible structure composed of several parts needed to support Notification applications. The parts composing the Notification message are:

- Generic Notification Message part (clause 6.1.1).
- Notification payload.

The Notification Message payload is further broken down into:

- Application-specific Notification message part.

- Media objects.

The Notification message provides support for:

- Message identification: ID and version number of the message.
- ESG/Service/Content reference: ESG, service or content it relates to.
- Timing information: Timing information for message lifecycle management.
- Filter information: Message attributes for filtering at reception.
- Fragmentation and re-assembly.

4.3.2 Mapping of Notification messages on broadcast transport layer

Depending on the target Notification application and attributes (size, time and synchronization constraints, etc.), the Notification messages can be split and mapped on several transport protocols as described in the sequel.

If Notification messages can be delivered without time constraints or with a time constraint compatible with an advanced delivery, they should be mapped on FLUTE protocol as depicted in figure 3.

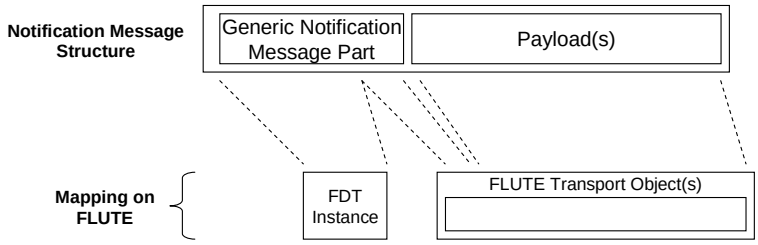

Figure 3: Mapping of Notification message on FLUTE (single message)

For efficiency reasons, more than one Notification message can be aggregated within a single transport object as depicted in figure 4. The Notification messages carried in the aggregate transport object may be individually described in the FDT instance, but it is also possible to restrict the description to the common parts of all messages.

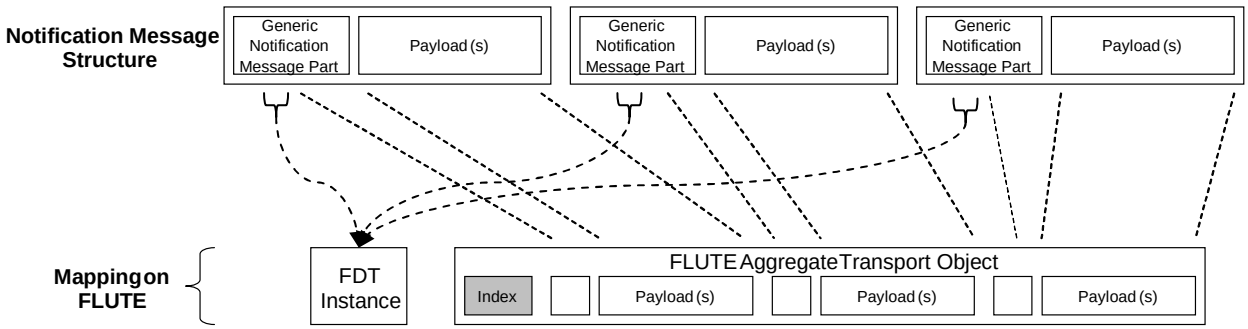

Figure 4: Mapping of Notification message on FLUTE (multiple messages)

If Notification messages need to be delivered with time constraints relative to an audio and/or video stream, they are mapped on the RTP protocol as depicted in figure 5. The lower part of the figure illustrates the case where only parts of the Notification message are sent in synchronization with an audiovisual stream, and others are delivered at different time. In this case it is possible to deliver only the former over RTP while carrying the bulk of the message (including the Notification payload) over FLUTE.

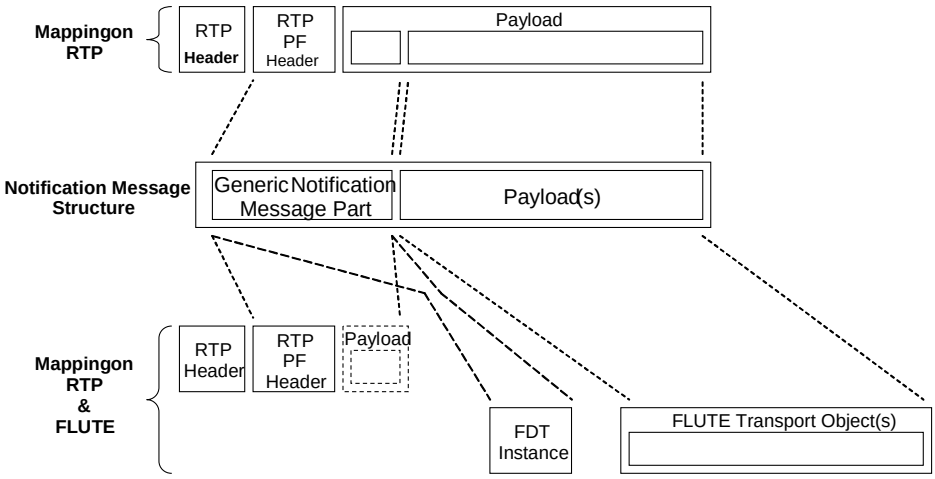

Figure 5: Mapping of Notification message on RTP and RTP+FLUTE

NOTE: Multiple messages can be carried in an aggregated container in case FLUTE transport is also involved.

For efficiency reasons, more than one Notification message can be aggregated within a single payload in RTP as depicted in figure 6. In this case, only the parts of the generic Notification message that are common to all messages are carried in the RTP Payload Format header, all other parts are carried in the respective sections or in the index of the aggregate payload.

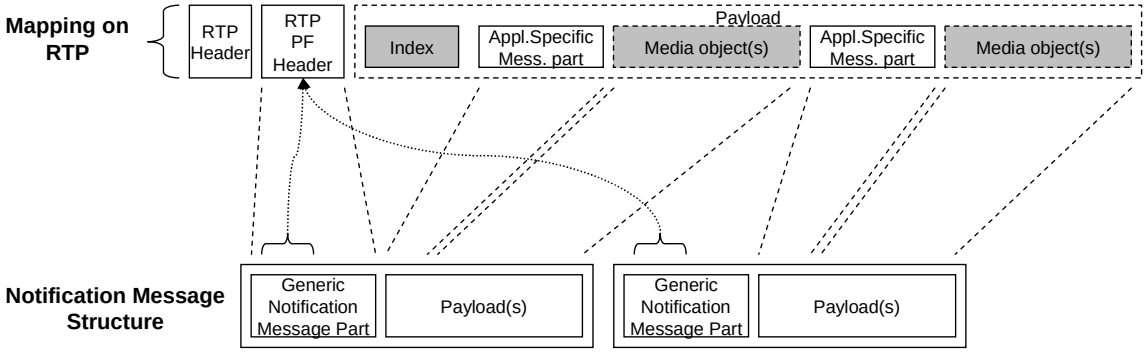

Figure 6: Mapping of Notification message on RTP (multiple messages)

4.3.3 Mapping of Notification messages on interactive transport layer

For services using Notification components not having real time requirements, or when the Broadcast network is not available, the Interaction network can be used to deliver Notification messages.

In this case, the Notification messages are carried in the same structure as for the broadcast transport.

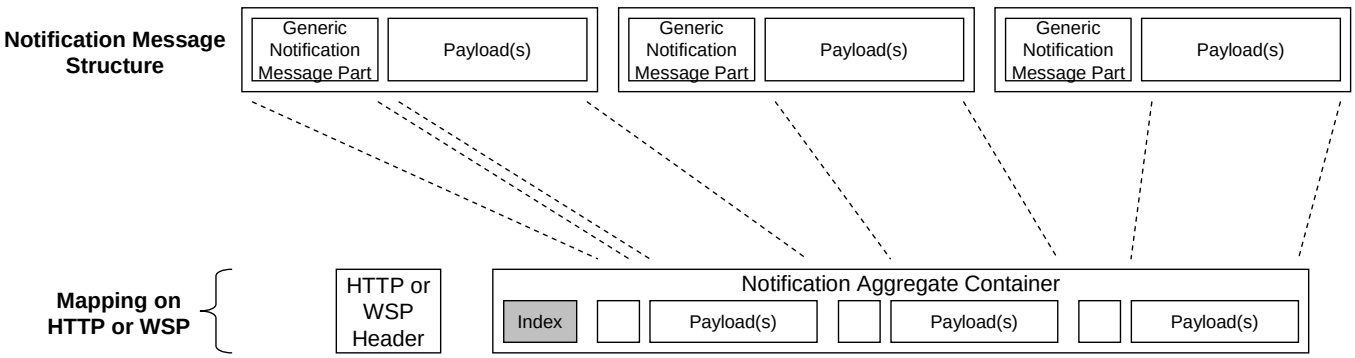

Figure 7: Mapping on HTTP or WSP

Retrieval of Notification messages is driven by terminals through a querying mechanism directed to the Notification server.

4.4 Dynamics of the Notification framework

In the Notification framework, a service component carrying Notification messages has a Component ID attached to it. The Component ID is used in the signalling of service components that may need to be consumed concurrently. In case a Notification component is shared among several services, Notification Types are used by the Notification client to retrieve the relevant Notification messages. A Notification Type is an identifier of the type of Notification message, similar to a port number in IP protocols. The Notification Type may be used to identify the handling application of the Notification messages. Some type values are registered, in order to map with "well-known" applications, such as for emergency; all others may be dynamically allocated. The Notification Type is a field defined in the Notification message header.

Messages are identified with a message ID. Such message ID maps to the handling context based on the Notification life cycle defined in clause 6.3. The Notification life cycle model is aimed at defining a processing model of the Notification messages.

When a terminal discovers the availability of the Notification service, it first acquires the initialization parameters from the ESG, more particularly the location of the Notification components and the location of an Initialization Container. Such Notification Initialization Container (NIC) may be carried in an ESG session or in-band with the Notification messages. The NIC may provide information such as the filtering criteria that may apply to the Notification messages.

5 Architecture

The IP Datacast Architecture document [i.3] defines the reference architecture for Notification services delivered by IP Datacast [i.2] over DVB-H [i.1]. The reference architecture specification is provided to illustrate the way the different components (e.g. audio, video and Notification) in IP Datacast over DVB-H work together.

Figure 8 identifies sub-entities of the main entities involved for the default Notification operation. It also highlights the reference points that are involved.

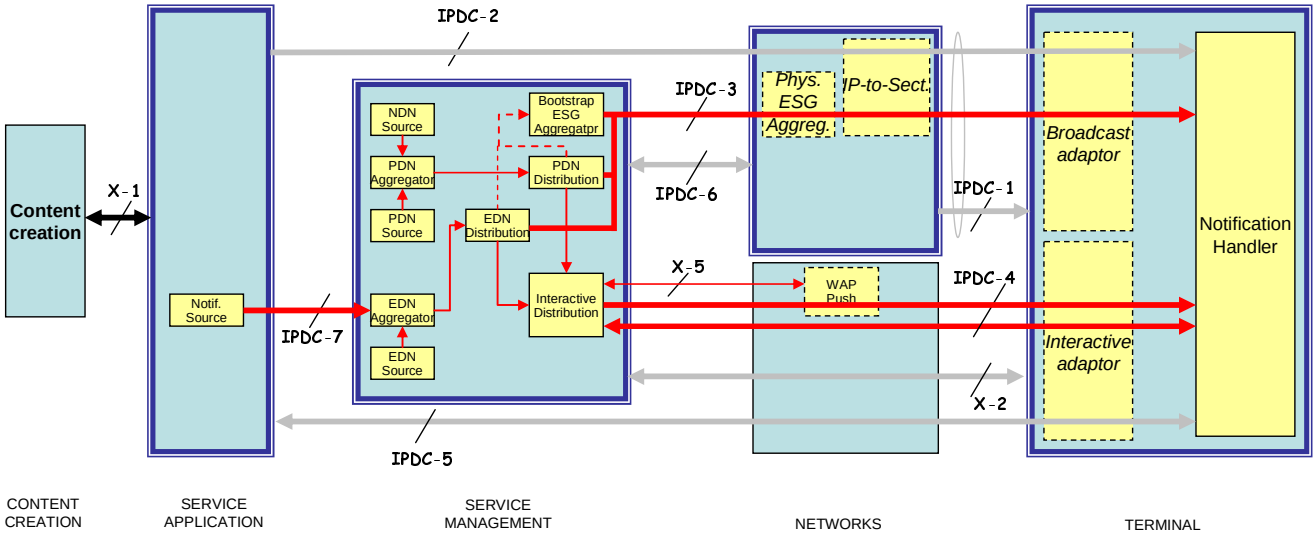

Figure 8: Sub-Entities and reference points activated for Default Notification delivery

Figure 9 identifies sub-entities of the main entities involved for the User selected Notification operation. It also highlights the reference points that are involved.

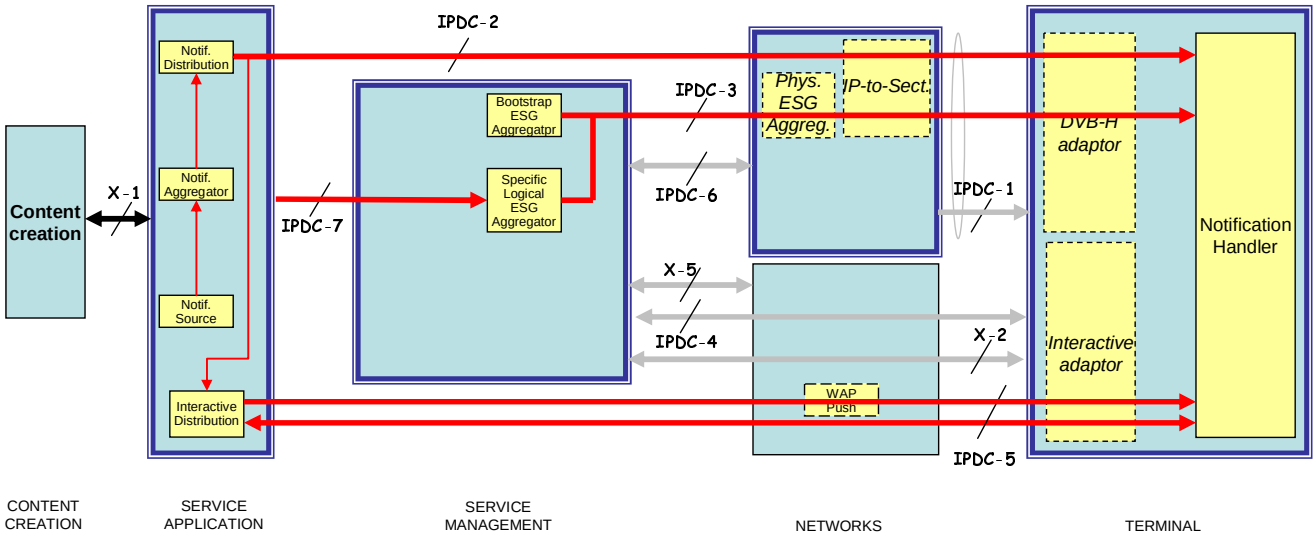

Figure 9: Sub-Entities and reference points activated for User selected Notification delivery

6 Notification message structure and transport

6.1 Notification message elements

In this clause the different Notification message parts as well as their fields are described.

6.1.1 Generic Notification Message Part

The generic Notification message part includes information that is to be processed by the Notification framework. This information may also be passed further to the Notification application.

The generic Notification message part consists of the following elements.

Table 1: Generic Notification message part elements

Field	Cardinality	Semantics
MessageID	1	ID of the current Notification message. May be avoided if signalled by the transport protocol.
Version	1	Version number of the current Notification message. May be avoided if signalled by the transport protocol.
Action	0..1	Describes the action to be performed on the Notification message.
NotificationType	1	Type of the Notification message. This information may be used to identify the target application for the Notification message. A list of static Notification message types is maintained by DVB. A range is reserved for dynamic assignment of Notification types. The scope of NotificationType for the dynamically assigned ones is the IP platform.
NotificationPayloadRef	0..1	Reference to the application specific message part of the Notification message. Contains Payload URI and optionally its container URI.
MediaObjectRef	0..N	Reference to the media parts of the Notification message. Contains Media URI and optionally its container URI.
ScheduleRef	0..N	Reference to the scheduled event to which the Notification message relates.
ServiceRef	0..N	Reference to the service to which the Notification message relates.
ESGRef	0..N	Reference to the ESG to which the Notification message relates.
IPPlatformRef	0..1	Reference to the IP Platform to which the Notification message relates.
TimingInformation	0..N	Indicates additional timing information to describe the handling of the Notification message.
FilterElementList	0..1	Indicates the filtering items that the current message satisfies.

"NotificationType" may be used to identify the target application for the Notification message. Values from 0 to 255 are reserved and their used described in annex A.

"Action" defines the action that is to be performed on the current Notification message. The possible actions are defined in table 2.

Table 2: Notification message actions

Value	Description
0	Launch
1	Cancel
2	Remove
3	Fetch ASAP
Other values	Reserved for future use
NOTE: When omitted, default value is "0" ("Launch").	

"TimingInformation" defines three sub-fields:

- **launch_time** (the time for the intended presentation time of the Notification message);
- **active_time** (the intended relative time from object activation until automatic cancellation);
- **life_time** (the intended life time until automatic removal of the object relative to object loading).

Table 3: Valid combinations of actions and timing information

Action	Timing information present			Description
	Launch_time	Active_time	Life_time	
Launch	**X**	**(X)**	**(X)**	Launch at time launch_time (see notes 1 and 2).
Cancel			**(X)**	Cancel immediately and update life_time if present.
Cancel		**X**	**(X)**	Update active_time and life_time if present.
Remove				Remove immediately.
Remove		**(X)**	**X**	Update life_time and active_time if present.
Fetch ASAP	**(X)**	**(X)**	**(X)**	Fetch immediately and update timing information if present.
NOTE 1: Launch_time is to be interpreted differently according to the transport protocol used. NOTE 2: If launch_time occurred in the past, then launch action is to be interpreted as launch ASAP if active_time is not elapsed.				

The generic message part may include a reference to the application-specific message part in the NotificationPayloadRef. The other media objects that are also part of the current Notification message may also be referred to in the generic message part by the MediaObjectRef. As the referred message parts may be carried as part of a Notification container, an optional reference to the container URI may be provided to facilitate access.

The generic Notification message part in XML format shall be compliant to the following schema.

```
<?xml version="1.0" encoding="utf-8" ?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
           xmlns:notif="urn:dvb:ipdc:notification:2008"
           elementFormDefault="qualified"
           targetNamespace="urn:dvb:ipdc:notification:2008">

    <xs:element name="NotificationDescription">
        <xs:complexType>
            <xs:sequence>
                <xs:element name="NotificationPayloadRef" type="notif:MessagePartRefType"
                            minOccurs="0" maxOccurs="1"/>
                <xs:element name="MediaObjectRef" type="notif:MessagePartRefType"
                            minOccurs="0" maxOccurs="unbounded"/>
                <xs:element name="TimingInformation" type="notif:TimingInformationType"
                            minOccurs="0" maxOccurs="unbounded"/>
                <xs:element name="FilterElementList" type="xs:base64Binary"
                            minOccurs="0" maxOccurs="1"/>
                <xs:element name="ScheduleRef"
                              type="xs:anyURI"
                              minOccurs="0" maxOccurs="unbounded"/>
                <xs:element name="ServiceRef"
                              type="xs:anyURI"
                              minOccurs="0" maxOccurs="unbounded"/>
                <xs:element name="ESGRef"
                              type="xs:anyURI"
                              minOccurs="0" maxOccurs="unbounded"/>
                <xs:element name="IPPlatformRef"
                              type="xs:anyURI"
                              minOccurs="0" maxOccurs="1"/>
                <xs:any minOccurs="0" maxOccurs="unbounded" namespace="##other"
                            processContents="lax"/>
            </xs:sequence>

            <xs:attribute name="MessageID"
                          type="xs:unsignedShort"
                          use="optional"/>
            <xs:attribute name="Version"
                          type="xs:unsignedByte"
                          use="optional"/>
            <xs:attribute name="Action"
                          type="xs:unsignedByte"
                          use="optional"/>
            <xs:attribute name="NotificationType"
                          type="xs:unsignedShort"
                          use="optional"/>
            <xs:anyAttribute processContents="skip"/>
        </xs:complexType>
    </xs:element>
```

```
    <xs:complexType name="MessagePartRefType">
        <xs:simpleContent>
            <xs:extension base="xs:anyURI">
            <xs:attribute name="ContainerRef" type="xs:anyURI" use="optional"/>
            </xs:extension>
        </xs:simpleContent>
    </xs:complexType>

    <xs:complexType name="TimingInformationType">
        <xs:attribute name="active_time" type="xs:unsignedInt" use="optional"/>
        <xs:attribute name="launch_time" type="xs:unsignedInt" use="optional"/>
        <xs:attribute name="remove_time" type="xs:unsignedInt" use="optional"/>
    </xs:complexType>

</xs:schema>
```

6.1.2 Notification Message payload

The Notification Message payload is composed of an Application-specific Notification message part and possibly Media objects. The Notification Message payload is intended to be consumed by the target application, resolved from the NotificationType value.

6.1.3 Encapsulation and Aggregation of Notification Messages

Notification messages may be composed of multiple parts. The transport of the Notification message parts may be done separately or as a single object. For the encapsulation of Notification message parts into a single transport object the Multipart/Related MIME format RFC 2387 [3] shall be used. Multiple Notification messages may also be aggregated into a single container using the same format.

The following parameters are indicated in the Content-Type field of the Multipart/Related MIME message:

- **boundary:** indicates a string value that is used as boundary between the different parts of the Multipart/Related MIME message.
- **start:** may optionally be used to indicate the ID of the root part of the Multipart/Related MIME message. It is expected that the root part is the first part of the message.
- **type:** shall be used to indicate the MIME type of the root part of the Multipart/Related MIME message. For a single Notification message in the Multipart/Related MIME container, the MIME type shall either be "application/vnd.dvb.notif-generic+xml" to indicate that the root part is the generic message part, or the MIME type of the application-specific Notification message part. If the Multipart/Related MIME container aggregated several Notification messages, the root part of the Notification message shall be an index list as defined below and the type shall be "application/vnd.dvb.notif-aggregate-root+xml".

In case of a single Notification message in the Multipart/Related MIME container, the root element shall be the generic message part, or, if the generic message part is carried separately, the application-specific message part.

In case of aggregation of multiple message parts from different Notification messages into a single Multipart/Related MIME container, the root part shall be an XML index list with the following format.

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
           xmlns:notif="urn:dvb:ipdc:notification:2008"
           elementFormDefault="qualified"
           targetNamespace="urn:dvb:ipdc:notification:2008">

  <xs:element name="MultipartIndex">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="MessagePart" type="notif:MessagePartType" minOccurs="0"
                  maxOccurs="unbounded" />
        <xs:element name="InitContainer" type="notif:PartType" minOccurs="0"
                  maxOccurs="unbounded"/>
      </xs:sequence>
      <xs:anyAttribute namespace="##any" processContents="lax"/>
    </xs:complexType>
  </xs:element>
```

```
  <xs:complexType name="MessagePartType">
    <xs:complexContent>
      <xs:extension base="notif:PartType">
        <xs:sequence>
          <xs:element name="FilterElementList" type="xs:base64Binary"
                  minOccurs="0" maxOccurs="1"/>
        </xs:sequence>
        <xs:attribute name="MessageID" type="xs:unsignedShort" use="optional"/>
      </xs:extension>
    </xs:complexContent>
  </xs:complexType>

  <xs:complexType name="PartType">
    <xs:attribute name="Version" type="xs:unsignedByte" use="optional"/>
    <xs:attribute name="NotificationType"  type="xs:unsignedShort"  use="optional"/>
    <xs:attribute name="Content-ID" type="xs:anyURI" use="required"/>
    <xs:attribute name="Content-Position" type="xs:nonNegativeInteger" use="optional"/>
    <xs:attribute name="Content-Type" type="xs:string" use="optional"/>
    <xs:attribute name="Content-Transfer-Encoding" type="xs:string" use="optional"/>
    <xs:attribute name="Content-Description" type="xs:string" use="optional"/>
  </xs:complexType>

</xs:schema>
```

Table 4 gives a description of the different fields of the index list.

Table 4: Fields of the index list for of Notification messages aggregate

Field	Semantics
MessageID	ID of the Notification message to which this Notification message part belongs to.
Version	Version number of the Notification message to which this Notification message part belongs to.
NotificationType	Type of the Notification message.
Content-ID	ID of the Notification message part.
Content-Position	Index of the Multipart/Related message part that contains the corresponding Notification message part. The index of the index list shall be 0.
Content-Type	Indicates the MIME type of the corresponding Notification message part.
Content-Transfer-Encoding	Indicates the type of content transfer encoding applied to the corresponding Notification message part.
Content-Description	Textual description of the message part.
FilterElementList	Filtering elements may be added to the index list to facilitate and accelerate filtering.
InitContainer	Indicates the initialization data for a specific Notification type. The structure is further specified in clause 7.3.
NOTE 1: When Notifications of the same type are aggregated, the NotificationType can be signalled in the FDT. NOTE 2: When Notifications of different types are aggregated and carried over a default Notification channel, the NotificationType field is mandatory. This enables to identify each corresponding NIC.	

6.2 Mapping on transport protocols

A Notification message may be transported in different formats, depending on its size, its components, and on the transport channel in use. Table 5 represents the different formats of the Notification payload as well as the corresponding mappings to the transport protocol.

Table 5: Notification Payload Format Semantics and values of NPF field

Format of the Notification payload	FDT content type	RTP NPF value
Reserved	N/A	0
Action - No payload	N/A	1
Generic message part only	application/vnd.dvb.notif-generic+xml	2
Generic message part + Application-specific message part with reference(s) to external Media objects	application/vnd.dvb.notif-container+xml	3
Generic message part + Application-specific message part without reference(s) to external Media objects	application/vnd.dvb.notif-container+xml	4
Aggregate of multiple Notification messages sharing some Notification information fields (e.g. same timestamp in RTP transport)	application/vnd.dvb.notif-container+xml	5
Initialization container for the Notification application	application/vnd.dvb.notif-init+xml	6
Reserved for future use	N/A	7 to 31

In case of RTP, "Generic message part + Application-specific message part with reference(s) to external Media objects" relates to Media objects carried over FLUTE, while for "Generic message part + Application-specific message part without reference to external Media objects", the complete message is carried over RTP.

In case of FLUTE, the reference(s) to Media objects is in the scope of the same FLUTE session where the Generic and Application-specific message parts are found.

6.2.1 Mapping on FLUTE

Notification messages or parts thereof may be transported over FLUTE as transport objects. A FLUTE transport object may also carry several Notification messages aggregated together as defined in clause 6.1. When used FLUTE [i.4] shall be based on TS 102 472 [1]

When the Notification message is transported over FLUTE, the generic message elements may be transported in the FLUTE object and/or in the FDT according to table 6.

Table 6: Location of generic Notification message elements for Notification Messages transported on FLUTE as individual Transport Object

Field	Carried in FDT	Carried in Object
MessageID	M	O
Version	M	O
Action	O	O
NotificationType	M	O
NotificationPayloadRef		O
MediaObjectRef		O
ScheduleRef		O
ServiceRef		O
ESGRef		O
IPPlatformRef		O
TimingInformation	O	O
FilterElementList	O	O

When an optional field is not present in FDT, it still may be present in the transport object. In case an optional field is present on both FDT and transport object, they shall be identical. If a terminal finds different values for a same field, it shall discard the complete message.

Table 7: Location of generic Notification message elements for Notification messages aggregated into single FLUTE Transport Object

Field	Carried in FDT	Carried in Object	Notes
MessageID	O	M	
Version	O	M	
Action	O	O	
NotificationType	M/O	O	For transport objects carrying Notification messages of same NotificationType, the NotificationType field is mandatory in the FDT.
NotificationPayloadRef		O	
MediaObjectRef		O	
ScheduleRef		O	
ServiceRef		O	
ESGRef		O	
IPPlatformRef		O	
TimingInformation	O	O	
FilterElementList	M/O	O	For transport objects carrying Notification messages of same NotificationType, the FilterListElement field is mandatory in the FDT.

6.2.1.1 Notification Message description

Upon reception of the FDT, the receiver is able to identify transport objects that carry Notification messages or parts thereof based on the indicated Content-Type as described in table 5. If compression, e.g. gzip, is applied to the transport object then it is indicated by the "Content-Encoding" field in the FDT, as specified in [1].

In order to enable fast identification and selection of the desired Notification messages, extensions to the FDT are defined to carry information about the Notification messages. The extension shall conform to the following XML schema.

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema targetNamespace="urn:dvb:ipdc:notif:FDText:2008"
           elementFormDefault="qualified"
           xmlns="urn:dvb:ipdc:notif:FDText:2008"
           xmlns:xs="http://www.w3.org/2001/XMLSchema"
           xmlns:notif="urn:dvb:ipdc:notification:2008">

  <xs:import namespace="urn:dvb:ipdc:notification:2008"/>

  <xs:element name="NotificationMessageDescription"
              type="NotificationMessageDescriptionType"/>

  <xs:element name="NotificationAggregateDescription">
    <xs:complexType>
        <xs:sequence>
            <xs:element name="FilterElementList" type="xs:base64Binary"
                  minOccurs="0" maxOccurs="1"/>
            <xs:element name="NotificationMessageDescription"
                  type="NotificationMessageDescriptionType"
                  minOccurs="0" maxOccurs="unbounded"/>
            <xs:element name="NICDescription"
                  type="NICDescriptionType"
                  minOccurs="0" maxOccurs="unbounded"/>
            <xs:any namespace="##other" processContents="skip" minOccurs="0" maxOccurs="unbounded"/>
        </xs:sequence>
        <xs:attribute name="NotificationType" type="xs:unsignedByte" use="optional"/>
        <xs:anyAttribute processContents="skip"/>
    </xs:complexType>
  </xs:element>

  <xs:complexType name="NotificationMessageDescriptionType">
    <xs:sequence>
      <xs:element name="TimingInformation" type="notif:TimingInformationType" minOccurs="0"
                  maxOccurs="unbounded"/>
      <xs:element name="FilterElementList" type="xs:base64Binary"
                  minOccurs="0" maxOccurs="1"/>
     <xs:any namespace="##other" processContents="skip" minOccurs="0" maxOccurs="unbounded"/>
    </xs:sequence>
```

```
    <xs:attribute name="MessageID" type="xs:unsignedShort" use="required"/>
    <xs:attribute name="Version" type="xs:unsignedByte" use="required"/>
    <xs:attribute name="Action" type="xs:unsignedByte" use="optional"/>
    <xs:attribute name="NotificationType" type="xs:unsignedByte" use="required"/>
    <xs:anyAttribute processContents="skip"/>
  </xs:complexType>

  <xs:complexType name="NICDescriptionType">
    <xs:sequence>
      <xs:any namespace="##any" processContents="skip" minOccurs="0" maxOccurs="unbounded"/>
    </xs:sequence>
    <xs:attribute name="NotificationType" type="xs:unsignedShort" use="required"/>
    <xs:anyAttribute processContents="skip"/>
  </xs:complexType>

</xs:schema>
```

NOTE: If the aggregate contains messages of the same Notification type, then the NotificationType in the NotificationAggregateDescription should be present, otherwise no NotificationType field in the NotificationAggregateDescription is allowed.

The presence of an information field in the FDT depends on whether the message is carried as an aggregate or not.

In the case of a single Notification message in the transport object, the NotificationMessageDescription element may be used to provide information about that Notification message. A NotificationMessageDescription element may also be used to describe a Notification container that encapsulates message parts of a single Notification message. It is recommended to provide all of the available information in this element, in order to accelerate filtering of the Notification messages upon reception of the FDT.

In case a Notification container aggregates message parts from multiple Notification messages, a NotificationAggregateDescription element may be present in the FDT to indicate further information about the aggregate. The NotificationAggregateDescription element provides filtering information applicable to the whole container. This implies that all used filter definitions are the same for the NotificationTypes of all contained messages. The filtering information may be provided as described in clause 6.4.2. More specific information for each single message of the aggregate (e.g. messageID, additional filter elements etc.) may be delivered in form of NotificationMessageDescription elements nested in the NotificationAggregateDescription element. The presence of the NICDescription element signals the existence of a Notification Initizalization Container in the aggregate itself for the NotificationType signalled in the element.

The following is an example of an FDT that describes a transport object carrying an aggregate of multiple Notification messages and providing the FDT extensions.

```
<?xml version="1.0" encoding="utf-8" ?>
<FDT-Instance xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema"
        xmlns:notif="urn:dvb:ipdc:notif:FDText:2008"
        Expires="3377633598"
        FEC-OTI-Maximum-Source-Block-Length="128000"
        FEC-OTI-Encoding-Symbol-Length="512"
        xmlns="urn:dvb:ipdc:cdp:flute:fdt:2005">
    <File Content-Location="NotificationContainer"
        TOI="1"
        Content-Length="1608"
        Content-Type="application/vnd.dvb.notif-container+xml">

        <notif:NotificationAggregateDescription>
            <notif:FilterElementList>A0BE21F4ABC231654=</notif:FilterElementList>
            <notif:NotificationMessageDescription MessageID="24"
                Version="1"
                NotificationType="56"/>
            <notif:NotificationMessageDescription MessageID="25"
                Version="3"
                NotificationType="56"/>
            <notif:NotificationMessageDescription MessageID="23"
                Version="1"
                NotificationType="56"/>
            <notif:NICDescription NotificationType="56"/>
        </notif:NotificationAggregateDescription>
    </File>
</FDT-Instance>
```

6.2.1.2 Selection and Filtering of Notification Messages

In order to increase efficiency of Notification delivery, the FDT may include information related to a Notification message (or several Notification messages) carried in a transport object. The information in the FDT that can be used to filter and select desired Notification messages are presented:

- **Service Reference (ServiceRef):** terminals interested in Notification messages of a given (Notification) service use the ServiceRef element to identify and receive those messages.
- **Notification Type (NotificationType):** terminals interested in specific Notification messages (e.g. only emergency messages) use the Notification type field to identify and receive those messages.
- **Notification Message ID (MessageID)** and **Notification Message Version (Version):** allow to the terminal to identify and discard duplicate messages.

NOTE: The Notification Message ID is unique in the scope of the NotificationType.

- **Subscription Information (SubscriptionInformation):** if present subscription information is used to identify whether the terminal has subscribed to the service providing this Notification message.
- **Filter Elements (FilterElementList):** filtering criteria are used to decide whether a transport object is considered or not. The terminal should receive the transport object if at least one of the filter items provided by the Notification application is signalled.

In case of an aggregate message, the FDT shall either include a description of all the messages in the aggregate or none. In the former case, terminals can decide whether the transport object is to be received or not based on the information about the contained Notification messages. In the latter case, the FDT may still provide filtering information that applies to all the messages in the aggregate. The terminal should be able to discard the aggregate transport object based on this information. Otherwise, the terminal should retrieve the transport object and check the aggregate index structure for further filtering information and more detailed and complete description of each Notification message in the aggregate.

6.2.1.3 Timing Information

The Timing information extension shall conform to the definition given in clause 6.1.1.

Table 8: Semantics of timing information when mapping over FLUTE

Field	Semantics
launch_time	A value that specifies the NTP value of the intended presentation time of the Notification message. This time may be in the past when a message (or trigger) is repeated to cope with packet loss or channel switching (see note).
active_time	A value indicating the intended relative time from object activation until automatic cancellation, in milliseconds.
life_time	A value indicating the intended life time until automatic removal of the object relative to object loading, in milliseconds.
NOTE: Launch_time is to be interpreted differently when used with RTP.	

6.2.2 Mapping on RTP

Notification messages that are tightly synchronized with media streams of a service shall be transported using RTP.

When the Notification message is transported over RTP, the generic message elements may be transported in the payload format header, extension header or payload according to table 9.

Table 9: Location of generic Notification message elements if transported over RTP

Field	Carried in payload format header	Carried in extension header	Carried in payload
MessageID	M		
Version	M		
Action	M		
NotificationType	M		
NotificationPayloadRef			O
MediaObjectRef			O
ScheduleRef			O
ServiceRef			O
ESGRef			O
IPPlatformRef			O
TimingInformation		O	O
FilterElementList		O	O
SubscriptionInformation		O	O

In case an optional field is present on both the extension header and the payload, they shall be identical. If a terminal finds different values for a same field, it shall discard the complete message.

6.2.2.1 RTP Header

Figure 10 describes the RTP packet header as defined by RFC 3550 [4]. The fields of the RTP packet header are used as described in RFC 3550 [4].

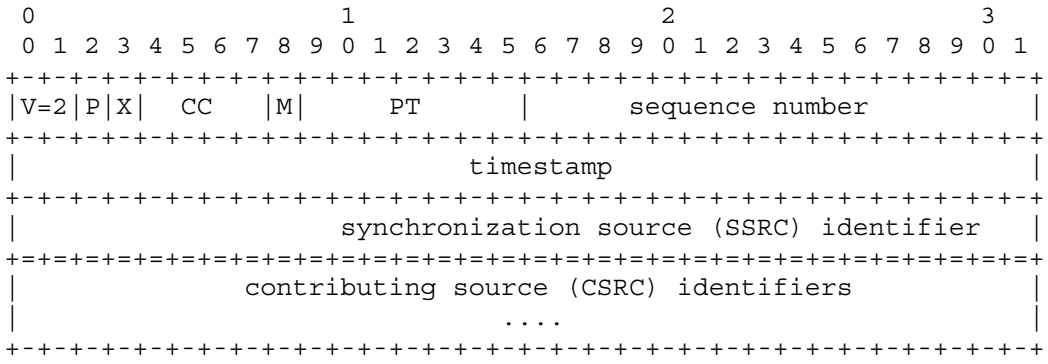

Figure 10: RTP Header

The Notification framework makes use of the RTP synchronization mechanism to achieve tight synchronization between the Notification message stream and other media streams of the same session.

The RTP timestamp along with the RTCP Sender Reports shall be used to determine the accurate time at which the indicated action is to be performed on the Notification message. If further timing information are present, then that information overwrites the information indicated in the RTP header. However, the additional timing information shall use the re-constructed sender timeline using RTP/RTCP.

6.2.2.2 RTP Payload Format Header

The RTP payload format header has the following format.

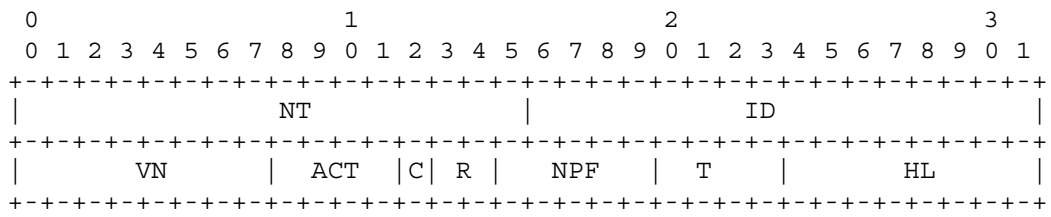

Figure 11: RTP payload format header

The syntax and semantics of the payload format header fields are defined as follows:

- ***NT (Notification Type): 16 bits***
  - This field is used to indicate the Notification type.
- ***ID (Message ID): 16 bits***
  - Identifier of the Notification message.
- ***VN (Version Number): 8 bits***
  - Notification message version. Can be used to check redundancy of Notification messages but also for fragmentation and reassembly.
- ***ACT (Action): 4 bits***
  - Defines the action that is to be performed on the current Notification message. ACT field takes the values between 0 and 15 as defined in table 2 in clause 6.1.1.
- ***NPF (Notification Payload Format): 5 bits***
  - Defines the format of the Notification payload of the RTP packet. An RTP packet may carry a complete Notification message (compound message), parts of a Notification message, the generic part of the Notification message, or no payload. If the payload consists of multiple Notification message parts then those are encapsulated to build a single container.

    The NPF field should be set according to table 5.
- ***R (Reserved): 2 bits***
- ***C (Compression): 1 bit***
  - Indicates whether compression has been applied to the RTP payload (before fragmentation). In case compression is applied, the default compression algorithm is supposed to be "gzip". The field is defined according to table 10.

Table 10: Compression of RTP Payload

Compression (C)	Indicates whether compression is applied to the RTP payload.
0	No compression is applied.
1	Compression is applied. The compression algorithm used should be signalled out-of-band. The algorithm defaults to "Gzip" as defined by RFC 1952 [5].

- ***T (Packet Type): 4 bits***

  - The packet type as defined in table 11; packets with reserved values of the type field shall be discarded.

Table 11: RTP packet type

Type	Description
0	Single packet
1	Fragmentation start Packet
2	Fragmentation continuing Packet
3	Fragmentation end Packet
4 to 15	Reserved for future use

- ***HL (Header Length): 8 bits***

  - defines the length of this payload format header (including extension headers) in 32 bits units.

Special restrictions apply when the payload does not carry a single notification message but an aggregate or an initialization container:

When NPF=5, ID shall be set to 0, VN shall be set to 0, ACT shall be set to 0; if the aggregate messages have different notification types, then NT shall be set to 0, if they have the same notification type, NT should reflect this Notification Type.

When NPF=5, terminal shall process NT. In case NT is different from 0, it means that all messages in the aggregate belong to the same Notification Type.

When NPF=6, ID shall be set to 0, ACT shall be set to 0.

6.2.2.3 Extension headers

The RTP payload format header is extensible and allows for additional information to be included in form of extension headers. The generic format of an extension header is as follows.

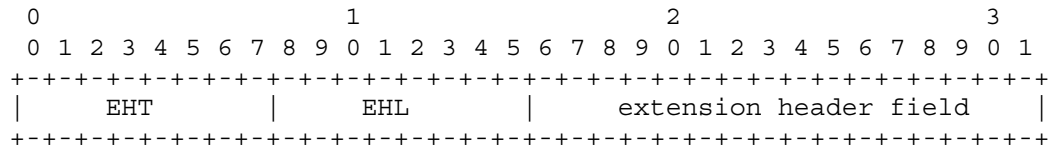

Figure 12: Generic extension header format

- ***EHT (extension header type): 8 bits***

  - Type of the extension header. A list of registered extension headers is defined and maintained by DVB.

- ***EHL (extension header length): 8 bits***

  - Length of the extension header field in bytes.

- ***Extension header: EHL bytes***

  - Content of the extension header.

The following extensions headers are defined:

Table 12: Extension headers and their types

Extension header	EHT value
Filter Element List	1
NotificationPayloadID	2
launch_time	3
active_time	4
life_time	5

- **Filter Element List** is defined in clause 6.4.2.
- **NotificationPayloadID:** a 16 bit uimsbf used to identify the file carrying the payload part of the Notification message.
  If "NotificationPayloadRef" is not explicitly given, the payload is carried in the related FLUTE session and the Content-Location field is set to a URI formatted as follows:
  "*dvb–ipdc_Notification_payload_<NotificationPayloadID>*",
  NotificationPayloadID being interpreted as a single positive decimal number coded into an ASCII string without leading zero.
  If "NotificationPayloadRef" is given, the "NotificationPayloadID" shall be ignored by the terminal.
- **launch_time:** a 32-bit uimsbf value that specifies the value of the timestamp of the Notification RTP stream at the intended presentation time of the Notification message. This time may be in the past when a message (or trigger) is repeated to cope with packet loss or channel switching.

NOTE: Launch_time is to be interpreted differently when used with FLUTE.

- **active_time:**a 32-bit uimsbf indicating the intended relative time from object activation until automatic cancellation in milliseconds.
- **life_time:** a 32-bit uimsbf indicating the intended life time until automatic removal of the object relative to object loading, in milliseconds.

6.2.2.4 Fragmentation Packets

Objects that exceed the networks maximum transmission unit (MTU) need to be fragmented before transmission. By fragmenting at the RTP level one need not rely on lower layer fragmentation, e.g. IP.

The payload format defines fragmentation of objects into two or more RTP packets.

NOTE: Fragmentation on the RTP level should however be seen as a solution only when fragmentation at the application level is not possible or available. Application level fragmentation allows creation of packets that are smaller than MTUs and can be processed individually, which results in better error resilience when packets are lost.

The common header values are as follows:

- **T (Type)**: 1, 2 or 3.
- **ID (Message ID)** and **VN (Version Number)**: must be identical in all the packets of a fragmented Notification message.

The first fragment shall be marked as type 1 and the last fragment shall be marked as type 3. Other fragments shall be marked as type 2. Type 0 indicates that no fragmentation has been performed.

Using the Notification message ID, the RTP sequence numbers, and the Type values (T), the receiver can reconstruct the original payload out of its fragments. Each fragment carries the same RTP payload format header with the difference being in the RTP sequence number and the packet type.

Extension headers are not expected to be present in each fragment of an RTP packet, but rather only in the first fragment of the packet.

6.2.2.5 SDP Parameters

The Notification component transported over RTP shall be described in the session description (SDP) of the session by a dedicated media line.

The Session Description specifies the destination port, media type, clock rate, and other initialization information.

The fields in the Session Description Protocol (SDP) are defined as follows:

- The media name in the "m=" line shall be "application".
- The encoding name in the "a=rtpmap" line shall be "NOTIF".

The SDP label attribute "a=label" RFC 4574 [6] shall be present and shall include an id value that uniquely identifies the Notification component among the media components of the same service.

Additional parameters to specify rate and version of the notification framework specification may be provided as part of the codec specific information in the "a=fmtp" line. These parameters are expressed in the form of a semicolon separated list of parameter=value pairs.

In the following, an example of the description of a Notification component in the SDP is given:

```
m=application 12345 RTP/AVP 100
a=rtpmap:100 NOTIF/1000
a=label:5
a=fmtp:100 Version =1;
```

6.2.3 Mapping on Interaction Network protocols

Delivery of Notification messages over the Interaction Network is performed according to the following steps:

1) Discovery: the terminal discovers the access to the desired Notification type . The terminal gets all the necessary information to register and get access to Notification messages of the desired type.

2) Registration: the terminal registers with the Notification service provider for the desired Notification service. The delivery modes are push and poll. In the push mode, the terminal may select between receiving message lists or the actual Notification messages or both, as long as those modes are supported by the server.

3) Delivery of the Notification Message List: In this step, the Notification message list is delivered to the terminal. In the Poll mode, the terminal connects to the interactive server and requests an actual Notification message list. In the push mode, the Notification message list is pushed to the terminal by the interactive server.

4) Retrieval of Notification Messages: the terminal requests one or more Notification messages from the interactive server. Alternatively, the terminal may retrieve the message from the related broadcast channel. The server can push the Notification messages to the terminal without a specific request.

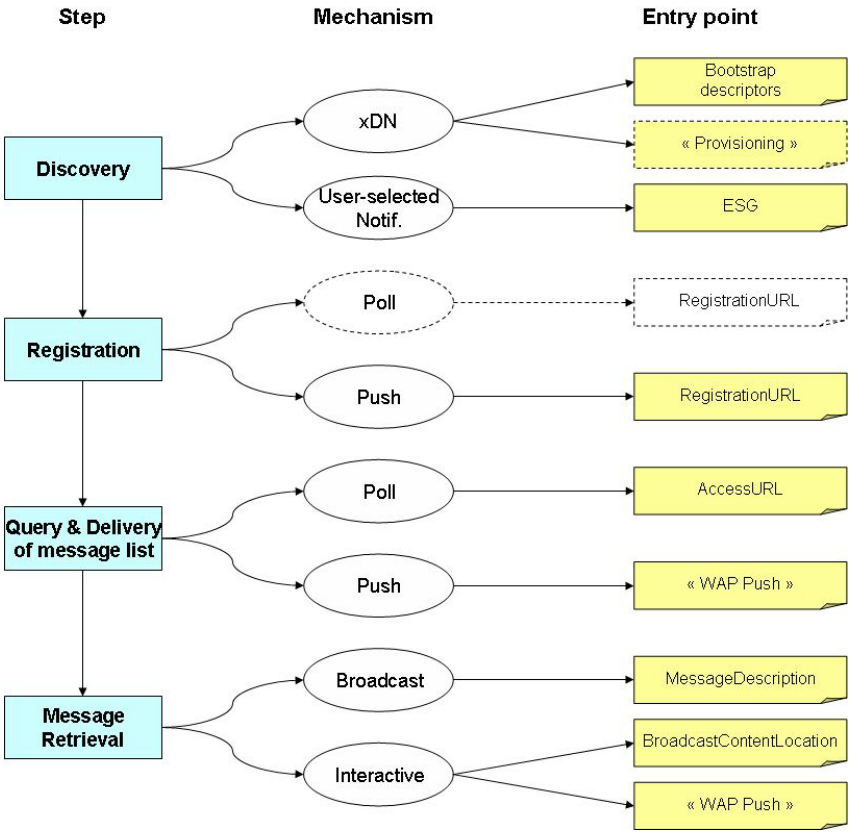

Figure 13: Overview of Notification over Interaction Network

6.2.3.1 Discovery of Notification access over the Interaction Network

Discovery of Notification access over the Interaction Network follows different mechanisms depending on whether the Notification channels are default ones or user-selected ones.

- In case of Default Notification channels, the Notification access over the Interaction Network is signalled in the DefaultNotificationAccessDescriptor, as specified in clause 7.1.1: for PDN and each EDN, the Notification bootstrap descriptor signals a URL and whether it is to register to an OMA PUSH delivery of Notifications, or to retrieve Notifications.

- In case of User-selected Notifications, the Notification access over the Interaction Network is signalled in the ESG and more specifically in the Acquisition Fragment where access to a Notification channel is signalled in the session description of the Notification component (see [2]).

An Acquisition Fragment may carry a description of a Notification component delivered over the Broadcast Network, Interaction Network, or both. In first two cases, it is intended that a terminal may select the one that best fits actual terminal context while third one implies that both Broadcast and Interaction Networks are simultaneously available to access the Notification channels. The type of Notification delivery is signalled in the ESG as well in order to help the terminal select the appropriate Acquisition Fragment. Acquisition Fragments for multiple delivery types may be simultaneously present.

6.2.3.2 Registration

When required, the terminal shall perform a registration procedure to enable delivery over the Interaction Network. To cancel the registration the terminal shall perform a De-registration procedure. The registration expires automatically at the indicated expiry time.

The registration procedure is performed using HTTP 1.1 POST [7] request/response messages.

6.2.3.2.1 Registration and Deregistration Request

The request is directed to the access point indicated by the RegistrationURL discovered from the discovery process.

The MIME type of the request shall be "application/vnd.dvb.notif-ia-registration-request+xml". The body of the request contains the following information:

a) the registration operation required (e.g. Register or Deregister);

b) the delivery mode, i.e. push (message list or messages or both) or poll;

c) the address of the device;

d) an identification of the device;

e) the notification type;

f) an optional reference to the ESG service.

The XML schema of the request body is described in the following table.

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
           xmlns:notif="urn:dvb:ipdc:notification:2008"
           elementFormDefault="qualified"
           targetNamespace="urn:dvb:ipdc:notification:2008">

    <xs:element name="RegistrationRequest" type="notif:RegistrationRequestType"/>

    <xs:element name="DeregistrationRequest" type="notif:DeregistrationRequestType"/>

    <xs:complexType name="RegistrationRequestType">
        <xs:sequence>
            <xs:element name="DeviceAddress" minOccurs="1">
                <xs:complexType>
                    <xs:simpleContent>
                      <xs:extension base="xs:string">
                        <xs:attribute name="Type" type="notif:DeviceAddressType" use="required"/>
                      </xs:extension>
                    </xs:simpleContent>
                </xs:complexType>
            </xs:element>
            <xs:element name="DeviceID" minOccurs="0">
                <xs:complexType>
                    <xs:simpleContent>
                        <xs:extension base="xs:string">
                            <xs:attribute name="Type" type="notif:DeviceIDType" use="required"/>
                        </xs:extension>
                    </xs:simpleContent>
                </xs:complexType>
            </xs:element>
            <xs:any namespace="##other" processContents="lax" minOccurs="0" maxOccurs="unbounded"/>
        </xs:sequence>
        <xs:attribute name="DeliveryMode" type="notif:DeliveryModeType" use="required"/>
        <xs:attribute name="NotificationType" type="xs:unsignedShort" use="optional"/>
        <xs:attribute name="ESGService" type="xs:anyURI" use="optional"/>
        <xs:anyAttribute namespace="##any" processContents="lax"/>
    </xs:complexType>

    <xs:complexType name="DeregistrationRequestType">
        <xs:sequence>
            <xs:any namespace="##any" processContents="lax" minOccurs="0" maxOccurs="unbounded"/>
        </xs:sequence>
        <xs:attribute name="RegistrationID" type="xs:unsignedInt" use="required"/>
    </xs:complexType>

    <xs:simpleType name="DeviceAddressType">
        <xs:restriction base="xs:string">
            <xs:enumeration value="MSISDN"/>
            <xs:enumeration value="IMSI"/>
            <xs:enumeration value="URI"/>
            <xs:enumeration value="IMPI"/>
            <xs:enumeration value="MIN"/>
            <xs:enumeration value="username"/> <!-- as defined in RFC2865 -->
        </xs:restriction>
```

```
    </xs:simpleType>

    <xs:simpleType name="DeviceIDType">
        <xs:restriction base="xs:string">
            <xs:enumeration value="DVB Device ID"/>
            <xs:enumeration value="IMEI"/> <!-- as defined in 3GPP TS 23.003 -->
            <xs:enumeration value="MEID"/> <!-- as defined in 3GPP2 C.S0072 -->
        </xs:restriction>
    </xs:simpleType>

    <xs:simpleType name="DeliveryModeType">
        <xs:restriction base="xs:string">
            <xs:enumeration value="Push Message List"/>
            <xs:enumeration value="Mixed"/>
            <xs:enumeration value="Push Messages"/>
            <xs:enumeration value="Poll"/>
        </xs:restriction>
    </xs:simpleType>

</xs:schema>
```

6.2.3.2.2 Registration and Deregistration Response

The response to a successful registration or deregistration request shall be an HTTP 200 OK message. In case of a registration request, the response shall include a registration identifier and the expiry time of the registration. The content type of the body shall be "application/vnd.dvb.notif-ia-registration-response+xml".

The XML schema of the registration response is given in the following table.

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
           xmlns:notif="urn:dvb:ipdc:notification:2008"
           elementFormDefault="qualified"
           targetNamespace="urn:dvb:ipdc:notification:2008">

    <xs:element name="RegistrationResponse" type="notif:RegistrationResponseType"/>
    <xs:element name="DeregistrationResponse" type="notif:DeregistrationResponseType"/>

    <xs:complexType name="RegistrationResponseType">
        <xs:sequence>
            <xs:any namespace="##any" processContents="lax" minOccurs="0" maxOccurs="unbounded"/>
        </xs:sequence>
        <xs:attribute name="RegistrationID" type="xs:unsignedInt" use="required"/>
        <xs:attribute name="ExpiryTime" type="xs:unsignedInt" use="optional"/>
        <xs:attribute name="AccessURL" type="xs:anyURI" use="required"/>
        <xs:anyAttribute namespace="##any" processContents="lax"/>
    </xs:complexType>

    <xs:complexType name="DeregistrationResponseType">
        <xs:sequence>
            <xs:any namespace="##any" processContents="lax" minOccurs="0" maxOccurs="unbounded"/>
        </xs:sequence>
        <xs:attribute name="RegistrationID" type="xs:unsignedInt" use="required"/>
    </xs:complexType>

</xs:schema>
```

The RegistrationID is used for subsequent operations on the registration, e.g. the de-registration request.

In case of unsuccessful registration, the HTTP response shall indicate an error code as described by table 13.

Table 13: HTTP Status code after Registration request

HTTP Status Code	Description
200 OK	The request has succeeded. The information returned with the response is dependent on the method used in the request.
400 Bad Request	The registration or de-registration request is not recognized or contains incorrect information.
401 Unauthorized	The request is rejected due to un-authorized access.
406 Not Acceptable	The request contains parameters or terminal selections that cannot be supported by the server.

6.2.3.3 Delivery of the Notification Message List

In case of Interaction Network delivery, the terminal receives a message that contains either a list of one or more Notification messages that are available for retrieval or a pointer to such a list. The latter configuration can be used in situations where the message is constrained to a maximum size by the transport layer (e.g. message list delivery over OMA push).

6.2.3.3.1 Format of Notification Message List

The information provided in the message list is given by table 14.

Table 14: Message list fields

Field	Level	Semantics
FilterInformation	Message list and Message	Base64 encoded filter list as described by clause 6.4.
StartTime	Message list	NTP timestamp. Indicates that the delivery list contains Notifications messages that have been available after the indicated StartTime.
EndTime	Message list	NTP timestamp. Indicates that the delivery list contains Notifications messages that have been available up to the indicated EndTime.
MessageID	Message	Identifier of the current Notification message.
MessageVersion	Message	Version number of the current Notification message.
NotificationType	Message	Indicates the Notification type for the current message.
ServiceID	Message List	Identifier of the Notification service fragment to which this message list applies.
RegistrationID	Message List	Identifier of the registration.
Message size	Message	Indicates the size of the Notification message.
BroadcastContentLocation	Message	Indicates the URI that is used as Content-Location for the delivery of the notification message over FLUTE in the corresponding broadcast channel.

NOTE: BroadcastContentLocation is provided optionally in the message list when the Notification message is available over broadcast bearer as well and the server wishes to indicate it is the preferred retrieval mechanism for that message.

The message list shall use "application/vnd.dvb.notif-ia-msglist+xml" as the MIME type and shall conform to the following XML structure.

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
           xmlns:notif="urn:dvb:ipdc:notification:2008"
           xmlns:fdt="urn:dvb:ipdc:notif:FDText:2008"
              elementFormDefault="qualified"
              targetNamespace="urn:dvb:ipdc:notification:2008">

    <xs:import namespace="urn:dvb:ipdc:notif:FDText:2008"/>

    <xs:element name="NotificationMessageList">
        <xs:complexType>
            <xs:choice>
```

```
                <xs:element name="NotificationMessageListPointer" type="xs:anyURI"/>
                <xs:element name="ActualMessageList">
                    <xs:complexType>
                        <xs:sequence>
                            <xs:element name="NotificationMessage"
                                type="notif:NotificationMessageType" minOccurs="1"
                                maxOccurs="unbounded"/>
                            <xs:any namespace="##any" processContents="lax" minOccurs="0"
                                maxOccurs="unbounded"/>
                        </xs:sequence>
                    </xs:complexType>
                </xs:element>
            </xs:choice>
            <xs:attribute name="StartTime" type="xs:unsignedInt" use="optional"/>
            <xs:attribute name="EndTime" type="xs:unsignedInt" use="required"/>
            <xs:attribute name="FilterInformation" type="xs:base64Binary" use="optional"/>
            <xs:attribute name="RegistrationID" type="xs:unsignedInt" use="optional"/>
            <xs:attribute name="ServiceID" type="xs:unsignedInt" use="optional"/>
            <xs:anyAttribute namespace="##any" processContents="lax"/>
        </xs:complexType>
    </xs:element>

    <xs:complexType name="NotificationMessageType">
        <xs:sequence>
            <xs:element name="NotificationMessageDescription"
                        type="fdt:NotificationMessageDescriptionType" minOccurs="1"/>
            <xs:element name="BroadcastContentLocation"
                        type="xs:anyURI" minOccurs="0"/>
        </xs:sequence>
        <xs:attribute name="size" type="xs:unsignedInt" use="optional"/>
    </xs:complexType>

</xs:schema>
```

The terminal first checks whether the message list contains new Notification messages by comparing its modification timestamp to the timestamp of the last received message list. If it is more recent, the message list is checked to find out any Notification messages of interest. If a Notification message is found to be of interest, the terminal checks how to retrieve the message and performs the retrieval.

6.2.3.3.2 Query format

The parameters associated with the HTTP POST request shall be communicated as key-value pairs following the conventions defined in section 17.13 of HTML 4.01 [10] for submitting HTML form data by the "POST" method using the "application/x-www-form-urlencoded" encoding type. More specifically, once encoded as "application/x-www-form-urlencoded", the parameters to be passed from terminal to system shall be communicated in the "message-body" of HTTP/1.1 "Request" message as defined in section 5 of RFC 2616 [7].

Within a single request, the terminal may include multiple key-value pairs. As defined by HTML4.01 [7] these key-value pairs SHALL be delimited by an "&".

The terminal may assign several values for a certain key. In this case the different values are separated by comma (",").

Table 15: Notification Query list and parameters

Key	Applicability	Value
Type	All	1: Notification message. 2: Notification Delivery List. 3: Notification Initialization Container.
ServiceID	All	Identifier of the Notification service fragment to which this query applies.
MessageID	1	Identifier of the requested Notification message.
MessageVersion	1	Version number of the requested Notification message.
NotificationType	All	Indicates the Notification type for the requested message or message list.
FilterInformation	1 and 2	Base64 encoded filter list as described by clause 6.4.
RegistrationID	All	Identifier of the registration.
StartTime	2	NTP timestamp. Requests that the delivery list contains Notifications messages that have been available after the indicated StartTime.
EndTime	2	NTP timestamp. Requests that the delivery list contains Notifications messages that have been available up to the indicated EndTime.

6.2.3.3.3 Push delivery

When registering to a Notification service, the terminal indicates the type of delivery it wants to have. If the delivery type is supported by the service provider, the registration is performed successfully.

For the push delivery, the terminal registers as a receiver and provides its device address. The service provider adds the terminal to its distribution list. The push delivery is performed on need basis, e.g. when a certain amount of new Notification messages becomes available, or it may be done periodically.

The push delivery shall use OMA PUSH OTA [8]. The OMA PUSH application ID assigned for the Notification framework agent by OMNA [9] is 0x9050.

If interactive delivery in push mode is supported, OTA-WSP [8] connectionless push over SMS (consisting in WSP Push units over SMS) shall be supported.

NOTE: [8] recommends that the overall size of the payload is restricted to no more than four SMS messages, leading to an available payload (after optional content encoding, e.g. gzip) of around 450 bytes for the Notification framework.

A PUSH message shall either contain a message list or a Notification message. The message list describes a set of Notification messages that are available for retrieval. The message list is defined in clause 6.2.3.2. Upon reception of the message list, the terminal selects the messages of interest and retrieves them as described in clause 6.2.3.4.

The Notification message shall be encapsulated as described in clause 6.1.2.

6.2.3.3.4 Poll delivery

In the case the Notification service is available over Poll delivery channel, HTTP POST shall be used for requesting the message list as described in clause 6.2.3.3.

The time interval between two consecutive polls shall not be less than the indicated poll period.

The Notification service provider may overwrite the polling period to improve its performance and to optimally use the network bandwidth.

6.2.3.4 Retrieval of Notification Messages

After processing the message list, the Notification messages of interest can be retrieved by using the AccessURL available in the acquisition fragment or signalled in a successful registration.

In the case the Broadcast content location is signalled in the message list, the terminal may tune into the related Broadcast Network and retrieve the message based on its identifiers (i.e. Notification type, message id, and version number or the URL to the transport object that carries the message).

If the delivery network is the Interaction Network, the message of interest is retrieved by sending an HTTP POST request as specified in clause 6.2.3.3.2 to the indicated retrieval URL. The following cases can be distinguished:

- Case 1: NotificationMessageList delivered over OMAPush:
  - Acquisition fragment provides RegistrationURL for OMA Push registration exclusively, not for Notifications retrieval.
  - The Registration response message provides the accessURL for Notification message retrieval.
- Case 2: NotificationMessageList is retrieved via the AccessURL using polling mechanism:
  - Acquisition fragment provides AccessURL for polling messages and message lists and an optional RegistrationURL for registration (if required).
  - If registration is required, the Registration response message provides the accessURL for message retrieval.
  - If registration is not required, assumption is that messages are retrieved via the accessURL provided in the acquisition fragment for polling.

In case a single message is requested, the successful response shall be formatted according to the Notification container as described in clause 6.1.2.

In case multiple Notification messages are requested, the successful response shall be formatted according to the aggregate message container as described in clause 6.1.2.

6.3 Notification object lifecycle

Notifications can be used for various purposes, from simple display of messages (e.g. emergency messages) over software updates to the transport, launch and control of entire interactive applications (e.g. voting buttons). The detailed behaviour of a given Notification application inside the terminal is out of scope of the present document, it is determined by the Notification object(s), carried in or referenced by the Notification message payload and delivered by the Notification framework to the application.

In order to guarantee predictable reaction of the system in a dynamic environment (unreliable transmission channel, interference with user action, e.g. channel change) it is useful to describe the intended behaviour in function of the state of the Notification object (e.g. absent, activated etc.) and the transitions between these states in function of the time and the received messages.

A major design goal is the achievement of a stable behaviour even when messages are lost (e.g. since the user hops back and forth between services), while still allowing resource optimization (e.g. when remaining lifetime of an object is not enough to allow satisfactory activation). Annex B gives an example of the most essential steps in the object lifecycle.

It shall be noted that the state diagram given below serves as a reference model and shall not impose any specific implementation, neither of the application, nor of the Notification framework. Many applications will have a higher number of internal states, and not all states may be observable (e.g. if an object has effectively been cleared from internal memory or not). Any implementation that exhibits the described behaviour is compliant to the present document.

6.3.1 States

From the point of view of the present document the Notification object may be in three stable or one transient state. These states together with their transitions as sketched in figure 14. Especially transitions from the active and loaded states to (eventually) the idle state are triggered not only through explicit actions, but also happen automatically so that the system retrieves its initial state even under bad reception conditions, or when the user decides to switch to different channels. Transitions in the diagram are understood to be instantaneous and to happen at the end of the implied action (e.g. an object is considered to be loaded while removal is in progress).

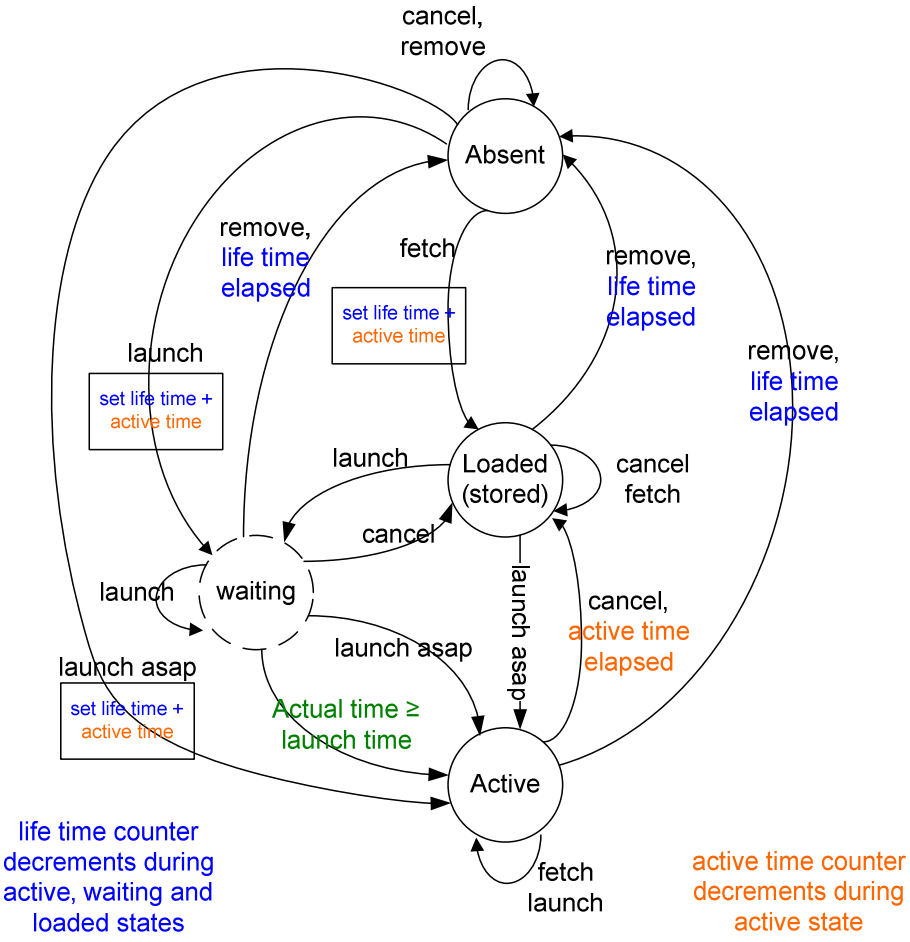

Figure 14: Notification object life cycle

6.3.1.1 Absent

This is the initial state of the object, and also the final state once the object has been (completely) removed from the system. This is the only state in which an object will stay forever. No timers are associated to this state. Transition from this state to any other state implies loading the object.

6.3.1.2 Loaded

This is the state, in which an object has been loaded (pre-fetched) into the system, but it has been neither activated nor has activation been programmed for some future time. (Due to our convention, the object will stay also in this state if an immediate activation action has been received but the activation has not yet been completely performed, e.g. waiting for the application to start).

The life time counter continuously decrements during this state; the object is removed when the life time elapses.

6.3.1.3 Waiting

When the object has been loaded, and an action has been received for activation at some future time, the object is in the waiting state (and stays in this state until the activation is completed, i.e. the application is launched). In this waiting state the launch time is continuously compared to some external time reference (e.g. the RTP presentation time stamps of an associated video stream). Typically the object transitions to the active state when the intended launch time has arrived or exceeded; this may be the case immediately, e.g. if the launch action was delayed during transmission. Also transition to other states may be triggered by appropriate actions.

The life time counter continuously decrements during this state; the object is removed when the life time elapses.

6.3.1.4 Active

When the object has been loaded and become active, the object is in the active state.

During this state, both the active time counter and the life time counter decrement continuously. Elapsing of the active time triggers an automatic transition back to the loaded state (but the object stays present). Elapsing of the life time completely removes the object from the system (triggers a transition to the absent state).

6.3.2 Timers

To manage the automatic transition between the life cycle states, it is made reference to timers as explained below. These timers are of conceptual nature and do not preclude any differing implementation.

6.3.2.1 Active time

The remaining active time is the intended time until automatic cancellation. It is initialized as a relative time from intended object activation to cancellation.

6.3.2.2 Life time

The remaining life time is the intended time until automatic removal of the object. It is initialized as a relative time at the time of object loading, with a resolution of milliseconds.

6.3.3 Actions

Transitions between the object life cycle states are initiated by actions as specified in subsequent clauses. These actions may be initiated by reception of Notification messages (both explicit and implicit), or automatically triggered after a certain time. In the following the different actions are discussed together with the proposed parameters passed to the object by these actions.

6.3.3.1 Fetch

The fetch action may be used by the Notification service provider to indicate that the receiver shall download the Notification object. The terminal should then tune-in to the corresponding, already discovered, delivery channel (e.g. a FLUTE session), retrieve the object and store it for future use.

As the object is fetched, its intended lifetime (until removal) is determined (possibly a default value). Accuracy is not critical, the provider should provide for enough margins.

The intended active time may also be determined as soon as the object is fetched regardless if the fetch is explicit or implicit (i.e. triggered when a launch action for a not yet loaded object is received).

Passing this parameter with the launch action would in principle be possible, but this would waste bandwidth since the launch action needs to be repeated regularly during the active time.

6.3.3.2 Launch

The launch action is used to indicate to receivers that the corresponding Notification object needs to be activated and processed by the target Notification application. The launch time is either indicated separately as a parameter or as part of the transport protocol (e.g. the RTP timestamp of the carrying RTP packet).

If the maximum active time was not defined during object fetch, it needs to be communicated with the launch action. Since launch messages (triggers) should be repeated in order to cope with non perfect reception or late channel switch, it is preferable to have active time known from the fetch to save bandwidth.

The launch action may take effect immediately (when the launch time indicated in the action occurred in the past), or when the launch time indicated in the action has arrived; this needs comparison of the launch time to some time reference (depending on the transport mechanism, e.g. when the presentation time of the RTP time stamps exceeds the indicated launch time).

6.3.3.3 Cancel

The cancel action may be used to annulate an active Notification object. The Notification object may be re-activated again by re-submitting a new Notification message version with a launch action. Upon receiving a cancel action, the target Notification application stops any processing and actions triggered by the Notification object launch message.

While the cancel action may be triggered through a specific Notification message (or trigger), in numerous cases the deactivation is triggered by the expiration of a timer. For this reason the cancel action in general does not carry further parameters (this means that the life time will not be modified by a cancel).

6.3.3.4 Remove

The remove action may be used to instruct receivers to discard the corresponding Notification object completely (freeing up any used resources, e.g. by discarding Notification message parts).

While the remove action may be triggered through a specific Notification message (or trigger), in most cases the object will be removed after a given time. This ends the object life, so no parameters are transmitted.

6.4 Message filtering

This clause specifies the generic filtering of Notification messages at the terminal side. The generic filtering mechanism consists of two different parts.

The first part is used for out-of-band announcement of filter definitions. This is achieved through the Notification Init Container, which is transported out of band (see clause 7.3). This container consists of a Notification Filter Table which describes the filter elements that can be carried in a particular Notification session.

The second part consists of the embedding of filter elements in the Notification messages in order to associate metadata with individual Notification messages. The filter elements are used by the terminal to determine whether a particular Notification message is of relevance to an application or user. This is achieved by matching the filter elements to the filter criteria.

6.4.1 Filter Definitions

The NotificationFilterTable element contains the exhaustive list of filter definitions associated with a Notification service. The table consists of a sequence of optional FilterEnum elements and at least one FilterDefinition element.

The FilterDefinition element defines the syntax and semantics of a filter element type that can be carried in Notification messages of a particular session. The FilterEnum element specifies a list of all possible items within an enumerated filter type (e.g. news categories, country names and sport events).

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
           xmlns:notif="urn:dvb:ipdc:notification:2008"
           xmlns:mpeg7="urn:mpeg:mpeg7:schema:2001"
           elementFormDefault="qualified"
           targetNamespace="urn:dvb:ipdc:notification:2008">

    <xs:import namespace="urn:mpeg:mpeg7:schema:2001"/>
```

```
    <xs:element name="NotificationFilterTable" type="notif:NotificationFilterTableType"/>

    <xs:complexType name="NotificationFilterTableType">
        <xs:complexContent>
            <xs:extension base="notif:InitFragmentType">
                <xs:sequence>
                    <xs:element name="FilterEnum" type="notif:FilterEnumType"
                            minOccurs="0" maxOccurs="unbounded"/>
                    <xs:element name="FilterDefinition" type="notif:FilterDefinitionType"
                            maxOccurs="unbounded"/>
                </xs:sequence>
            </xs:extension>
        </xs:complexContent>
    </xs:complexType>

    <xs:complexType name="FilterEnumType">
        <xs:sequence>
            <xs:element name="FilterEnumItem" maxOccurs="unbounded">
                <xs:complexType>
                    <xs:simpleContent>
                        <xs:extension base="xs:string">
                            <xs:attribute name="itemID" type="xs:unsignedShort" use="required"/>
                        </xs:extension>
                    </xs:simpleContent>
                </xs:complexType>
            </xs:element>
        </xs:sequence>
        <xs:attribute name="enumID" type="xs:anyURI" use="required"/>
    </xs:complexType>

    <xs:complexType name="FilterDefinitionType">
        <xs:sequence>
            <xs:element name="FilterName" type="mpeg7:TextualType" maxOccurs="unbounded"/>
            <xs:element name="FilterDescription"
                        type="mpeg7:TextualType" maxOccurs="unbounded"/>
        </xs:sequence>
        <xs:attribute name="EnumIDRef" type="xs:anyURI" use="optional"/>
        <xs:attribute name="filterID" type="xs:unsignedByte" use="required"/>
    </xs:complexType>

</xs:schema>
```

Table 16: NotificationFilterTable Fields

Field	Semantics
itemID	Specifies a unique identifier of a single enumerated item within the FilterEnum. This identifier is unique within the FilterEnum.
enumID	Specifies a unique identifier of the FilterEnum. This identifier is unique within the NotificationType.
EnumIDRef	Specifies the enumID of the FilterEnum element which specifies the list of enumeration items of the filter definition type.
FilterName	The name of the filter definition in text form. The name may be specified in different languages.
FilterDescription	Specifies the textual description of the filter definition in a specified language.
filterID	Specifies a unique identifier of the instantiated filter definition (see note).
NOTE: Used to match the filterID of the filter definition from the filter table to a Notification filter element.	

6.4.2 Filter Elements

This clause specifies a binary representation for the filter elements carried in individual Notification messages. This binary representation can be embedded in the FilterElementList element of the generic header part of the Notification message using base64 encoding (see clause 6.1.1).

In case of FLUTE, the binary filter elements can also be embedded in the FDT extension as specified in clause 6.2.1.

In case of RTP, the binary filter elements can also be embedded as an optional extension header of the binary RTP Payload format header as specified in clause 6.2.2.3.

The filter element list contains an arbitrary number of filter elements. The number of filter elements in the list is deduced from the length of the list.

Semantics for the filter element:

filter_ID (8 bits): the filter_ID identifier refers to the filter definition ID (filterID) of the filter element value.

value (16 bits): the actual value of the filter element. This can either be an itemID or an unsigned integer value.

6.4.3 Filtering of aggregates

When aggregating multiple messages of the same Notification type into the same container for carriage in a single transport object over FLUTE, the Notification service shall whenever possible include filtering information at the aggregate level.

The aggregate level filtering information is provided by the FilterElementList element in the NotificationAggregateDescription element.

The aggregate level filtering information shall be constructed from the filtering information of all the Notification messages in the aggregate as follows:

1) Extract the set of filtering elements that are common to all messages. A filtering element is said to be common if each Notification message of the aggregate has indicated at least one filter value for that filtering element.

2) For each of the common filtering elements, include all the corresponding filter values that apply to at least one message of the aggregate.

At the receiver side, the aggregate level filtering is performed as follows:

1) For each filtering element at the aggregate level, check that at least a corresponding filter value matches the user/terminal preferences. Discard the message if that is not the case.

2) If the user/terminal preferences uses other filtering elements that do not appear at the aggregate level then filtering continues at the message level.

The above behaviour applies assuming an "AND" relationship between the different filtering elements. In case of an "OR" relationship, the aggregate can only be discarded if all of the filter elements that appear in the user/terminal preferences are common filtering elements and are not satisfied. Otherwise, filtering at the message level will be needed.

7 Bootstrap and initialization of Notification services

7.1 Discovery of default Notification services

In this clause the bootstrap process for default Notification service is specified. As introduced in clause 5, one PDN service and several EDN services can be transported in parallel in an IP Platform. To indicate to the receiving terminal the availability of PDN and EDN services, additions to the bootstrap procedure are defined as follows:

- The "DefaultNotificationAccessDescriptor" and its transport are specified in clause 7.1.1. The "DefaultNotificationAccessDescriptor" provides the Acquisition of available Default Notification Services.

- The Notification bootstrap procedure is specified in clause 7.1.2.

7.1.1 Bootstrap descriptor

7.1.1.1 Syntax of DefaultNotificationAccessDescriptor

The "DefaultNotificationAccessDescriptor" is a binary representation of acquisition of default Notification service. The "DefaultNotificationAccessDescriptor" specifies the Acquisition information related to current IP platform or a particular ESGProviderID signalled in the "ESGProviderDiscovery" descriptor.

Table 17: DefaultNotificationAccessDescriptors Syntax

Syntax	No. of bits	Mnemonic
DefaultNotificationAccessDescriptor {		
n_o_PDNEntries	8	uimsbf
n_o_EDNEntries	8	uimsbf
For(i=0;i<n_o_PDNEntries;i++){		
PDNEntry()		
}		
For(i=0;i< n_o_ EDNEntries;i++){		
EDNEntry[i]()		
}		
}		

Syntax	No. of bits	Mnemonic
PDNEntry{		
PDNEntryVersion	8	uimsbf
ChannelType	8	uimsbf
EntryLength	8+	vluimsbf8
If (ChannelType == 1) {		
IPVersion6	1	bslbf
Reserved	7	bslbf
If(IPVersion6){		
SourceIPAddress	128	bslbf
DestinationIPAddress	128	bslbf
}else{		
SourceIPAddress	32	bslbf
DestinationIPAddress	32	bslbf
}		
Port	16	uimsbf
TSI	16	uimsbf
} else if (ChannelType == 2 || ChannelType == 3) {		
AccessURLLength	16	uimsbf
for (j=0;j<AccessURLLengt;j++) {		
AccessURL_char	8	uimsbf
}		
}		
if (ChannelType == 3) {		
PollInterval	32	uimsbf
}		
}		

Syntax	No. of bits	Mnemonic
EDNEntry{		
EDNEntryVersion	8	uimsbf
DeliveryMethod	8	uimsbf
EntryLength	8+	vluimsbf8
ProviderID	16	uimsbf
If (DeliveryMethod == 1) {		
IPVersion6	1	bslbf
Reserved	7	bslbf
If(IPVersion6){		
SourceIPAddress	128	bslbf
DestinationIPAddress	128	bslbf
}else{		
SourceIPAddress	32	bslbf
DestinationIPAddress	32	bslbf
}		
Port	16	uimsbf
TSI	16	uimsbf
} else if (DeliveryMethod == 2 || ChannelType == 3) {		
AccessURLLength	16	uimsbf
for (j=0;j<AccessURLLengt;j++) {		
AccessURL_char	8	uimsbf
}		
}		
if (DeliveryMethod == 3) {		
PollInterval	32	uimsbf
}		
}		

Table 18: DefaultNotificationAccessDescriptors Semantics

Field	Semantics
n_o_PDNEntries	Indicates the number of PDN entries in the current descriptor. At most one indicator per channel type is allowed.
n_o_EDNEntries	Specifies the number of EDN Entries in which access information of EDN service is signalled.
PDNEntryVersion	Specifies the version of PDN Entry Specification. The value shall be set to "1" (see notes 1 and 2).
EDNEntryVersion	Specifies the version of EDN Entry Specification. The value shall be set to "1" (see notes 3 and 4).
DeliveryMethod	Indicates the method of the delivery. The following values are currently defined: 1: Broadcast delivery 2: Push delivery as defined in clause 6.2.3.3.3. 3: Poll delivery as defined in clause 6.2.3.3.4.
EntryLength	Specifies the length of the PDN/EDN Entry in Bytes excluding the PDNEntryVersion/EDNEntryVersion and EntryLength fields (see note 5).
IPVersion6	If set to "1" specifies that the SourceIPAddress and the DestinationIPAddress are signalled according to IP version 6. If set to "0" specifies that the SourceIPAddress and the DestinationIPAddress are signalled according to IP version 4.
ProviderID	This ID is used to uniquely identify the ESG provider in the ESGProviderDiscoveryDescriptor. The ESG provider must register the ProviderID at the authority that manages the bootstrapping channel to guarantee uniqueness.
SourceIPAddress	Specifies the source IP address of the FLUTE session transporting the PDN/EDN messages. The IP Version is signalled by the IPVersion6 field.
DestinationIPAddress	Specifies the destination IP address of the FLUTE session transporting the PDN/EDN messages. The IP Version is signalled by the IPVersion6 field.
Port	Specifies the port number of the IP Stream of the FLUTE session in which the PDN /EDN messages is transported.
TSI	Specifies the Transport Session Identifier (TSI) of the FLUTE session in which the PDN /EDN messages is transported.
AccessURLLength	Indicates the length of the access URL.
AccessURL_char	Gives the URL to access the delivery service over interactive channel. The URL is formatted as a UTF-8 string.
PollInterval	Indicates the minimal interval in seconds between two consecutive poll requests.
NOTE 1: This version is incremented if the specification of PDN Entry is changed in a not forward compatible way. NOTE 2: A receiver should only decode PDN Entries which it complies to. NOTE 3: This version is incremented if the specification of EDN Entry is changed in a not forward compatible way. NOTE 4: A receiver should only decode EDN Entries which it complies to. NOTE 5: This allows forward compatible implementations even if fields are added in the future to DefaultNotificationAccessDescriptor.	

According to the file delivery specification in TS 102 472 [1], the following information is required to launch a FLUTE agent in the terminal. The listed fields are specified in the DefaultNotificationAccessDescriptor except the ones printed italic. For those printed italic default values are assumed as listed:

1) The Source IP address.

2) The *number of channels* in the session is fixed to 1.

3) The Destination IP address of the only channel of the session.

4) The Port number of the only channel of the session.

5) The Transport Session Identifier.

6) The *start and end time* for the session is fixed to 0-0.

7) The *protocol* is fixed to FLUTE/UDP.

8) The *media type* is assumed to be "application" and the format list contains only one item "0".

7.1.1.2 Transport of DefaultNotificationAccessDescriptor

The "DefaultNotificationAccessDescriptor" is transported in ESG Bootstrap FLUTE session in a well-known IP address and port, as defined in TS 102 471 [2].

Additionally the following restrictions apply:

The "DefaultNotificationAccessDescriptor" is transported in a dedicated transport object in the bootstrap FLUTE session. It should be signalled in the FDT by setting the attribute:

Content-Type="application/vnd.dvb.ipdcdftnotifaccess".

7.1.2 Bootstrap procedure

The foreseeable way to retrieve information to access the PDN service and a specific EDN service within an IP platform would be as follows:

1) Tune-in to the ESG Bootstrap session.

2) Retrieve "ESGProviderDiscovery" descriptor, "ESGAccessDescriptor" descriptor and "DefaultNotificationAccessDescriptor".

3) Look up "DefaultNotificationAccessDescriptor", get access information for PDN service.

4) Upon selecting a specific ESG described in "ESGProviderDiscovery" descriptor, get Provider ID of the selected ESG.

5) Look up "DefaultNotificationAccessDescriptor", filter EDN entries by Provider ID, and get access information for EDN service of the selected ESG.

7.2 Discovery of user selected Notification services

User selected Notification services can be categorized as plain Notification services and service related Notification components. Both are signalled in ESG datamodel based on TS 102 471 [2].

- A standalone Notification service is signalled as a regular service in the ESG.
- A service related Notification component is signalled as a component within a regular service in the ESG.
- A service-related Notification component is signalled as a regular service in the ESG linked to another regular service via RelatedMaterial. Figure 15 depicts the relationship between ESG fragments.

NOTE 1: The second configuration is NOT compatible with legacy terminals.

NOTE 2: Only the third configuration allows for separate purchase of the notification component from the base service it relates to.

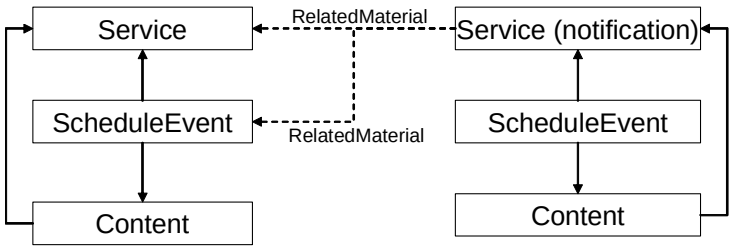

Figure 15: Relationship between fragments of Notification and regular services

7.3 Notification Initialization Container

7.3.1 NIC Format

The Notification Initialization Container (NIC) contains the necessary information to initialize the consuming Notification application. The NIC information applies to a specific Notification application, which is identified by the Notification type. The NIC container is in XML format and conforms to the following schema.

```
<?xml version="1.0" encoding="utf-8" ?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
           xmlns:notif="urn:dvb:ipdc:notification:2008"
           elementFormDefault="qualified"
           targetNamespace="urn:dvb:ipdc:notification:2008">

    <xs:element name="NotificationInitContainer">
        <xs:complexType>
            <xs:sequence>
                <xs:element name="InitFragment" type="notif:InitFragmentType"
                        minOccurs="1" maxOccurs="unbounded"/>
            </xs:sequence>
            <xs:attribute name="NotificationType" type="xs:unsignedShort" use="required"/>
            <xs:anyAttribute processContents="skip"/>
        </xs:complexType>
    </xs:element>

    <xs:complexType name="InitFragmentType" abstract="true">
        <xs:sequence>
            <xs:any namespace="##other" processContents="skip"
                    minOccurs="0" maxOccurs="unbounded"/>
        </xs:sequence>
        <xs:attribute name="InitFragmentID" type="xs:anyURI" use="required"/>
        <xs:anyAttribute processContents="skip"/>
    </xs:complexType>

</xs:schema>
```

The NIC is extensible and allows for new initialization data to be defined. The NIC consists of one or more initialization fragments, which provide initialization information for a specific functionality of the Notification service. Each initialization fragment is identified by a URI, which indicates the type of information that it contains. All initialization fragments have to be defined in a schema definition as extensions to the "NotificationInitFragmentType" element.

The current specification defines the following Notification initialization fragments are defined:

- Filtering Initialization Information as defined in clause 6.4.1. The corresponding InitFragmentID is defined to be "urn:dvb:ipdc:2008:Notification:NIC:filtering".
- Default Timer Initialization Information as defined in clause 7.3.4.
- The corresponding InitFragmentID is defined to be "urn:dvb:ipdc:2008:Notification:NIC:timers".
- Compression Algorithm Initialization Information as defined in clause 7.3.3. The corresponding InitFragmentID is defined to be "urn:dvb:ipdc:2008:Notification:NIC:compression".
- Notification type information as defined in clause 7.3.5. The corresponding InitFragmentID is defined to be "urn:dvb:ipdc:notification:2008:NIC:nt_information".

A terminal shall ignore an initialization data fragment that it does not recognize the InitFragmentID value.

7.3.2 Transport of the NIC

The Notification init container may be delivered in-band, along with the corresponding Notification messages, or out-of-band e.g. in the ESG. The NIC and all its external components may be encapsulated into a Notification container or delivered as separate objects.

At the receiver, the NIC of a specific Notification service is identified as follows:

- At ESG level: using the URI of NIC, if given by the ESG in the Acquisition Fragment.
- At RTP level: using the NPF field of the RTP payload format header.
- At FLUTE level:
  - in case of delivery as individual transport object: using the Content-Type of the transport object in the FLUTE FDT;
  - in case of aggregation with Notification messages or other objects: using the NICDescription element in the extended FDT, and the InitContainer element in the index list.

7.3.3 Signalling Compression Algorithms

The default compression algorithm used for compressing Notification messages is Gzip [5].

In case a different compression algorithm is needed by a Notification service, information about that algorithm is provided by the NotificationInitContainer. This is done using a new Init Fragment definition which conforms to the following schema definition.

```
<xs:complexType name="CompressionInit">
    <xs:complexContent>
      <xs:extension base="notif:InitFragmentType">
        <xs:attribute name="Name" type="xs:string" use="required"/>
        <xs:attribute name="Pointer" type="xs:anyURI" use="required"/>
      </xs:extension>
    </xs:complexContent>
</xs:complexType>
```

7.3.4 Default Timer Information

The default active time and life time values may be indicated in the Notification Initialization Container. If not present, the default active time is assumed to be 3 600 000 milliseconds and the default life time is assumed to be 86 400 000 milliseconds.

The initialization fragment of the NIC for the default timer information shall conform to the following schema definition.

```
<xs:complexType name="TimerInit">
    <xs:complexContent>
      <xs:extension base="notif:InitFragmentType">
        <xs:attribute name="active_time" type="xs:unsignedInt" use="optional"/>
        <xs:attribute name="life_time" type="xs:unsignedInt" use="optional"/>
      </xs:extension>
    </xs:complexContent>
</xs:complexType>
```

7.3.5 Notification type information

In order to enable the handling of unknown Notification types, terminals may find in the Notification Init Container a pointer (URL) to a source that provides further information about the Notification type.

The initialization fragment of the NIC for the Notification type information shall conform to the following schema definition.

```
<xs:complexType name="NotificationTypeInformation">
    <xs:complexContent>
      <xs:extension base="notif:InitFragmentType">
        <xs:attribute name="Pointer" type="xs:anyURI" use="required"/>
      </xs:extension>
    </xs:complexContent>
</xs:complexType>
```

7.3.6 Example of the NIC

In this clause an example of a NIC object is given.

```
<?xml version="1.0" encoding="utf-8"?>
<NotificationInitContainer xmlns="urn:dvb:ipdc:notification:2008"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      NotificationType="156">


  <InitFragment xsi:type="CompressionInit"
                InitFragmentID="urn:dvb:ipdc:notification:2008:compression" Name="ZLIB"
                Pointer="http://www.ietf.org/rfc/rfc1950.txt"/>

  <InitFragment xsi:type="TimerInit" InitFragmentID="urn:dvb:ipdc:notification:2008:timers"
                active_time="30000" life_time="3600000"/>

</NotificationInitContainer>
```

7.4 Processing of Notification Messages

The processing of a Notification message depends on the type of its application-specific message part. The Notification type of the Notification service is mapped uniquely to the MIME type of the application-specific message part. This mapping is static for well-defined Notification applications and dynamic for other Notification applications. A range for static Notification type values as well as the corresponding MIME type of the application-specific Notification message parts is defined in annex A At the receiver, a consuming application may consume one or several Notification services, by registering for the corresponding MIME types.

The following processing steps are performed at the receiver:

- the MIME type of the application-specific message part is identified;
- the consuming application is initialized with the Notification init container (if present) and the processing of the Notification message starts.

The first step is executed differently depending on the payload format of the Notification message and on the carrying Notification channel.

If the Notification message has a static Notification type then the MIME type is identified based on the registry table in annex A.

If the Notification type is dynamic and the carrying channel is a default Notification channel then the MIME type of the application-specific message part is extracted from one of the following fields:

- Content-Type field in the FDT, if the application-specific part is delivered as a standalone transport object.
- Content-Type field that is either available in the Index list of the carrying aggregate Notification container or in the corresponding MIME part of the multipart MIME/related Notification message payload.

If the Notification type is dynamic and the carrying channel is not a default Notification channel, i.e. the corresponding Notification service is declared in the ESG, the MIME type is identified from the ExtAcquisitionRef element of the Schedule Event Fragment. Alternatively, the MIME type can be discovered upon discovering the message, i.e. from the FDT or from the Notification container as described above.

An additional pointer (URL) in the Notification Init Container to a source that provides further information about the related Notification type may be provided for terminals, in order to enable the handling of unknown Notification types. Terminals may discard Notification messages of unknown Notification types.

Annex A (informative): Static Notification Types

Well-defined Notification applications are registered with DVB to guarantee compatibility and wide deployment. The Notification type value range 0 to 255 is reserved for static Notification types.

The following table is an example of how registry of static Notification types and their corresponding MIME type of the application-specific message part, could look like.

Notification Type	MIME type of the application-specific message part	Application description
0		Reserved
1	text/xml	ESG application (ESG update)
2	application/octet-stream	Notification application in the smartcard
3	text/xml	Emergency messages rendering application

For a Notification type to be registered, it is expected that the MIME type of the application specific message part, the message format and the expected handling application behaviour be document at registration, in order to help implementers developing the application.

Up to date list of registered static Notification types is available under http://www.dvbservices.com/identifiers/index.php.

Annex B (informative): Example Of The Object Lifecycle

A major design goal is the achievement of a stable behaviour of the Notification system even when messages are lost.

The following example highlights the most essential steps in the object lifecycle; it does not intend to give all possible cases. It uses the Notification in one of the most challenging applications, the activation and execution of interactive objects synchronized to a (video) service.

As in every (especially wireless) transmission system, Notification messages may be lost in the transmission channel before they reach the terminal. A second reason for message loss can be found if the user repeatedly switches between several services: the system is not only expected to be stable, but also to allow to the terminal to deliver a satisfactory user experience even in this case (at discretion of the terminal manufacturer).

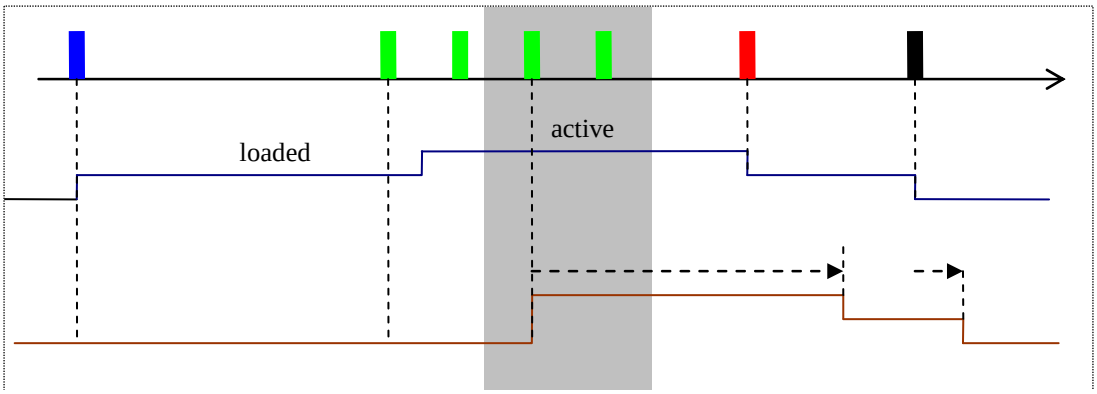

Figure B.1: Two example life cycles

Figure B.1 sketches (implicit or explicit) actions, and the resulting lifecycle of the object in two cases: the upper (dark blue) one for a terminal which is in perfect reception conditions, the lower (brown) one for a terminal that receives Notifications only during a limited time (grey).

The server first initiates a fetch action (blue), and later a launch message (green). In order to guarantee reliable delivery of Notification messages, and also to allow channel change by the user during the time that the (service related) Notification object is active, the server repeatedly transmits launch messages during all the time that the object is active. Repetition of messages with different actions may not be needed and can be avoided for higher efficiency: the loss of a "fetch" action will only delay the activation of the object (since its fetching will be triggered by the succeeding launch), Loss of "cancel" and "remove" actions can be compensated through timers as sketched above.

In the first (blue) case the object is loaded as soon as possible (Blue fetch action, e.g. implicit if object is carrouseled). On reception of the first "launch" Notification it is activated. The moment of activation may either be the reception of the Notification, or the Notification may indicate the moment of activation, related to an accompanying audiovisual flow. It is deactivated and unloaded through explicit actions.

In the second (brown) case, the terminal may switch to the channel only when the object could already be activated. It receives an activation message and loads (e.g. from a carrousel, or through the interactive link) and activates the object immediately. The terminal may do some resource optimization, e.g. it may skip activation when remaining lifetime of an object is not enough to allow satisfactory activation. Once the object is loaded and launched, the terminal has sufficient information to get rid of the object even when communication is disrupted: deactivation of the object is triggered by a timer after it has been active for a predetermined time. Finally the object is unloaded.

Annex C (normative): Notification Framework Usage

C.1 Example of Notification Initialization

C.1.1 Discovery of Default Notifications

The ESG bootstrapping is the mechanism through which the terminal discovers not only the ESG itself but also the Default Notifications, in the scope of a given IP platform.

In the ESG bootstrap FLUTE session of each IP platform, the terminal retrieves the ESGProviderDiscoveryDescriptor, ESGAccessDescriptor and DefaultNotificationAccessDescriptor transported in a certain transport objects specified in FDT. Here each descriptor file is signalled in the FDT by setting the attribute Content-Type as follows:

- ESGProviderDiscoveryDescriptor : Content-Type="text/xml".
- ESGAccessDescriptor : Content-Type="application/vnd.dvb.ipdcesgaccess".
- DefaultNotificationAccessDescriptor : Content-Type="application/vnd.dvb.ipdcdftnotifaccess".

At power on, the terminal joins the ESG bootstrap FLUTE session by tuning to the well-known destination IP address (224.0.23.14 defined for IP version 4 or FF0X:0:0:0:0:0:0:12D for IP version 6 format) and the port for "ipdcesgbs" as defined in [2].

Firstly, the terminal can extract the PDN entry information associated with the IP platform directly by parsing the DefaultNotificationAccessDescriptor.

Upon selecting a specific ESG described in ESGProviderDiscoveryDescriptor, terminal can get ProviderID of the chosen ESG. Then, by filtering EDN entries listed in the DefaultNotificationAccessDescriptor with acquired ProviderID, it also can get access information for EDN of selected ESG.

The following example illustrates the structure of the DefaultNotificationAccessDescriptor. Here, it provides access information both for PDN and EDN with various delivery types.

In PDN message case, it can be received by two means at the same time, Broadcast and Push:

Broadcast Type:

- Source address: 10.89.27.213 (IP version 4)
- Destination address: 225.0.0.59 (IP Version 4)
- Port No: 6512
- TSI: 0001

Push Type:

- Registration URL: http://example.notif.com/ipdc_PDN_PUSH

In the EDN case of this example, the IP platform is serving 2 ESG providers with:

1) ESG Provider 1 - ProviderName: IPDC_PR01 - ProviderID : 0x0001
2) ESG Provider 2 - ProviderName: IPDC_PR02 - ProviderID : 0x0002

The first ESG provider is delivering the EDN messages via the Push method only, while the 2nd ESG provider is delivering the EDN messages via the Poll method only, with a minimum interval of 15 s.

ESG Provider 1 - Push Type:

- Registration URL: http://example.notif.com/ipdc_EDN_PUSH

ESG Provider 2 - Poll Type:

- Access URL: http://example.notif.com/ipdc_EDN_POLL
- Poll Interval: 15 s

Then, the DefaultNotificationAccessDescriptor can be structured as shown in table C.1:

Table C.1: Example of a DefaultNotificationAccessDescriptor

Field	Value	No. of Bits	Description
n_o_PDNEntries	0x02	8	One PDN has two entries: 1) Broadcast channel access point. 2) Push channel access point.
n_o_EDNEntries	0x02	8	Two ESG providers in an IP Platform. And each ESG provider has one Delivery Method respectively.
PDNEntryVersion	0x01	8	Version of the PDN Entry spec. - version 1.
DeliveryMethod	0x01	8	PDN is delivered in Broadcast Type. (DeliveryMethod "1")
EntryLength	0x2E	8	The length of the PDNEntry in Bytes. - "46".
IPVersion6	0x00	1	Set to "0", Indicating the IP version 4.
Reserved	0x3F	7	
SourceIPAddress	0x0A591BD5	32	Source IP address "10.89.27.213" of the transport flow transporting the PDN.
DestinationIPAddress	0xE100003B	32	Destination IP address "225.0.0.59" of the transport flow transporting the PDN.
Port	0x1970	16	Port number "6512" of the transport flow transporting the PDN.
TSI	0x0001	16	transport Session Identifier (TSI) "1" of the transport flow transporting the PDN.
PDNEntryVersion	0x01	8	Version of the PDN Entry spec. - "version 1".
DeliveryMethod	0x02	8	PDN is delivered in Push Type. (DeliveryMethod "2")
EntryLength	0x29	8	The length of the PDNEntry in Bytes. - "41".
AccessURLLength	0X0026	16	Length of the URL address - "38".
AccessURL_char	See right	UTF-8 coded	URL access address http://example.notif.com/ipdc_PDN_PUSH.
EDNEntryVersion	0x01	8	Version of the EDN Entry spec. - "version 1".
DeliveryMethod	0x02	8	EDN is delivered in Push Type. - "Method 2".
EntryLength	0x2B	8	The length of the PDNEntry in Bytes. - "43".
ProviderID	0x0001	16	ESG Provider ID - "0001", corresponding to "IPDC_PR01".
AccessURLLength	0X0026	16	Length of the URL address - "38".
AccessURL_char	See right	UTF-8 coded	URL access address http://example.notif.com/ipdc_EDN_PUSH.
EDNEntryVersion	0x01	8	Version of the EDN Entry spec. - "version 1".
DeliveryMethod	0x03	8	EDN is delivered in Poll Type. - "Method 3".
EntryLength	0x2F	8	The length of the PDNEntry in Bytes. - "47".
ProviderID	0x0001	16	ESG Provider ID - "0002", corresponding to "IPDC_PR02".
AccessURLLength	0X0026	16	Length of the URL address - "38".
AccessURL_char	See right	UTF-8 coded	URL access address http://example.notif.com/ipdc_EDN_POLL.
Pollinterval	0x000F	32	Poll Interval for poll access - "15" s.

Notification Type

1) Timing.

2) Action.

3) Filter element.

The following is an example of a default notification message. An emergency message (Type - 0003) targeting to terminals is broadcast directing for immediate LAUNCH with a limited lifespan of 10 min (600000 milliseconds). The Init Container contains the definition of the filter elements. When the receiver receives the message using the chosen access method, the application to handle emergency message is launched and the filtering element in the message is interpreted according to the definition and value.

- Notification Type - "Emergency" (0003)
- TimingInformation - life_time = "600000"
- Action - "Launch" (0)
- FilterElementList - Value = "AAEBBA=="

The generic Notification message part XML is as follows:

```
<?xml version="1.0" encoding="utf-8"?>
<NotificationDescription xmlns="urn:dvb:ipdc:notification:2008"
    xmlns:xs="http://www.w3.org/2001/XMLSchema"
    xsi:schemaLocation="urn:dvb:ipdc:notification:2008 notification-description.xsd"
    MessageID="1048"
    Version="0001"
    Action="0"
    NotificationType="3">

        <NotificationPayloadRef>http://example.notif.com/noti</NotificationPayloadRef>
        <TimingInformation life_time="600000"/>
        <FilterElementList>AAEBBA==</FilterElementList>

</NotificationDescription>
```

C.2 Use of Notification message for ESG update

In this clause, an ESG update message is a Notification message allowing to notify updates of an ESG on broadcast and/or interactive channel.

C.2.1 Message format of ESG update message

ESG update message is a sub-type message of EDN, as defined in clause 4.1.1.3.

The generic message part of ESG update message shall be in the XML format defined in clause 6.1.1. Additionally, it shall conform to table C.2:

Table C.2: Generic message part of ESG update message

Field	Cardinality	Semantics
MessageID	1	As defined in the present document.
Version	1	As defined in the present document.
Action	0..1	Should not occur and terminal may ignore it.
NotificationType	1	Shall be 0x0001.
NotificationPayloadRef	0..1	As defined in the present document.
MediaObjectRef	0..N	As defined in the present document.
ScheduleRef	0..N	Should not occur and terminal may ignore it.
ServiceRef	0..N	Should not occur and terminal may ignore it.
ESGRef	0..N	Shall be the ID of the ESG which the message is relevant to. Notes that it may occur multiple times when the updated ESG part is shared by several ESG.
IPPlatformRef	0..1	As defined in the present document.
TimingInformation	0..N	Should not occur and terminal may ignore it.
FilterElementList	0..1	Should not occur and terminal may ignore it.

The application-specific message part shall be a DeliveryList in XML format, as defined in [2].

C.2.2 Transport

When present, ESG update message as a type of EDN shall be transported via broadcast network or interactive network, as defined in clause 6.2.

C.3 Use of Notification message for Addressing smartcard

Notification messages with NotificationType=0x0002 are targeting a Notification application inside the smartcard, invoked by the OMA Smart Card Web Server (SCWS) ([11] or [12]).

Such Notification messages shall consist of Generic message part + Application-specific message part without reference(s) to external Media objects ("application/vnd.dvb.notif-container+xml" MIME type). MIME type of application-specific message part shall be "application/octet-stream". When the terminal receives a Notification message with NotificationType=0x0002,

- if the terminal does not support SCWS, the terminal shall discard the message.
- if the terminal supports SCWS, but smartcard does not support SCWS, the terminal shall discard the message.
- if the terminal supports SCWS, and smartcard supports SCWS, the terminal shall not apply any filtering and shall transfer the message to Smart Card Web Server (SCWS).

Terminal shall choose to send message either to localhost (loopback address 127.0.0.1) or localuicc (TCP/IP), depending on the underlying transport protocol that is implemented in the terminal. Refer to [11] or [12].

HTTP POST method shall be used to transfer notification message to SCWS.

Content-Type shall be "application/vnd.dvb.notif-container+xml".

URL of the application shall be: /dvb-notification.

Message body shall contain the Notification Message.

EXAMPLE:

POST /dvb-notification HTTP/1.1 CRLF

Content-Type: application/vnd.dvb.notif-container+xml CRLF

Content-Length: *xx* CRLF

Host: anything

CRLF

[data : Generic Notification Message part (XML), Notification Message Payload (MIME, MediaObject)]

Annex D (informative): Bibliography

- IETF RFC 3551: "RTP Profile for Audio and Video Conferences with Minimal Control".
- ISO/IEC 15938-5: "Information technology -- Multimedia content description interface - Part 5: Multimedia description schemes".

History

Document history		
V1.1.1	November 2008	Publication
V1.2.1	January 2010	Publication