

Digital Audio Broadcasting (DAB); Rules of Operation for the Multimedia Object Transfer Protocol

European Broadcasting Union



Union Européenne de Radio-Télévision

EBU-UER

DAB
Digital Audio Broadcasting



Reference

DTR/JTC-DAB-9

KeywordsDAB, digital, audio, broadcasting, multimedia,
protocol**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

Important notice

Individual copies of the present document can be downloaded from:

<http://www.etsi.org>

The present document may be made available in more than one electronic version or in print. In any case of existing or perceived difference in contents between such versions, the reference version is the Portable Document Format (PDF). In case of dispute, the reference shall be the printing on ETSI printers of the PDF version kept on a specific network drive within ETSI Secretariat.

Users of the present document should be aware that the document may be subject to revision or change of status. Information on the current status of this and other ETSI documents is available at

<http://portal.etsi.org/tb/status/status.asp>

If you find errors in the present document, send your comment to:

editor@etsi.fr

Copyright Notification

No part may be reproduced except as authorized by written permission.
The copyright and the foregoing restriction extend to reproduction in all media.

© European Telecommunications Standards Institute 2002.

© European Broadcasting Union 2002.

All rights reserved.

DECT™, **PLUGTESTS™** and **UMTS™** are Trade Marks of ETSI registered for the benefit of its Members.
TIPHON™ and the **TIPHON logo** are Trade Marks currently being registered by ETSI for the benefit of its Members.
3GPP™ is a Trade Mark of ETSI registered for the benefit of its Members and of the 3GPP Organizational Partners.

Contents

Intellectual Property Rights	5
Foreword.....	5
1 Scope	6
2 References	6
3 Definitions and abbreviations.....	6
3.1 Definitions	6
3.2 Abbreviations	6
4 General	6
4.1 Introduction	6
5 Structural description	7
5.1 Segmentation of objects	8
5.1.1 Segmentation of the MOT body	8
5.1.2 Segmentation of the MOT directory	8
5.1.3 Segmentation of the MOT header.....	8
5.2 Mapping of MOT segments into MSC data groups.....	9
5.3 Packetizing segments - Network level.....	9
5.3.1 Packet Mode	9
5.3.2 X-PAD	9
6 Fields of the MOT Header.....	11
6.1 Header core	11
6.2 Header extension	11
6.2.1 Naming files and directory structures	12
7 Model of a MOT data decoder and its interfaces	12
7.1 Network level	13
7.2 Data group level	14
7.3 Segmentation and object level.....	14
7.3.1 General description of the MOT data decoder.....	14
7.3.2 The reassembly unit	14
7.3.2.1 MOT directory mode.....	14
7.3.2.2 MOT header mode	15
7.3.2.3 Segmentation of MOT bodies	15
7.3.3 The object management unit.....	15
7.3.3.1 MOT directory mode.....	15
7.3.3.2 MOT header mode	17
7.3.4 Advanced caching.....	20
7.4 User application level.....	20
8 Transmission Mechanisms	20
8.1 General	20
8.2 Data transmission	21
8.2.1 Single object transmission	21
8.2.2 Multiple object Transmissions.....	21
8.3 Update, addition and deletion of objects	22
8.3.1 Directory mode	22
8.3.2 Header mode	23
Annex A: Coding	24
A.1 The MSC data group	24
A.1.1 Coding of the MSC data group.....	24
A.1.2 Examples of coding	25
A.1.2.1 Example 1: MOT object, no segmentation	25
A.1.2.2 Example 2: MOT object, segmentation	27

A.1.2.3 Example 3: MOT directory, no segmentation.....30

History32

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 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 Report (TR) has been produced by the 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 1: 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 Eureka Project 147 was established in 1987, with funding from the European Commission, to develop a system for the broadcasting of audio and data to fixed, portable or mobile receivers. Their work resulted in the publication of European Standard, EN 300 401 [1], for DAB (see note 2) which now has worldwide acceptance. The members of the Eureka Project 147 are drawn from broadcasting organizations and telecommunication providers together with companies from the professional and consumer electronics industry.

NOTE 2: DAB is a registered trademark owned by one of the Eureka Project 147 partners.

1 Scope

The present document provides useful information concerning the use of the MOT protocol.

2 References

For the purposes of this Technical Report (TR) the following references apply:

- [1] ETSI EN 300 401: "Radio broadcasting systems; Digital Audio Broadcasting (DAB) to mobile, portable and fixed receivers".
 - [2] ETSI EN 301 234: "Digital Audio Broadcasting (DAB); Multimedia Object Transfer (MOT) protocol".
-

3 Definitions and abbreviations

3.1 Definitions

For the purposes of the present document, the terms and definitions given in [1] and [2] apply.

3.2 Abbreviations

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

DAB	Digital Audio Broadcasting
HTML	Hyper Text Markup Language
MOT	Multimedia Object Transfer
MSC	Main Service Channel
PAD	Programme Associated Data
X-PAD	Extended Programme Associated Data

4 General

MOT is a transport protocol for the transmission of Multimedia content in DAB data channels to various receiver types with Multimedia capabilities.

4.1 Introduction

Various possibilities for transmitting information are incorporated into a common transport mechanism for different DAB data channels, so that the access to Multimedia content is unified within the DAB system.

MOT ensures interoperability between:

- different data services and application types;
- different receiver device types and targets;
- equipment from different manufacturers.

Each data service has an associated application specification, and that specification includes the transport mechanisms for the data content. If the application uses files of information then these are best transported using the MOT protocol layered onto the DAB transport mechanisms for packet mode and PAD.

The MOT protocol allows objects of a finite length from an information source, i.e. the Content/Service provider to be conveyed to a destination, i.e. the terminal, as shown in figure 1, where in terms of MOT:

the Content/Service provider is capable of processing various types of Multimedia content (e.g. picture and text files) in an appropriate way, so that this data is compliant with the MOT specification and can be fed into a DAB Ensemble multiplexer;

the Terminal is fed from a MOT data decoder capable of processing the multimedia content of a DAB Ensemble in an appropriate way, so that it is:

- decoded and presented to the user; or
- forwarded to a following entity, which then processes the content.

The definition of interfaces between the different entities is not within the scope of the MOT specification.

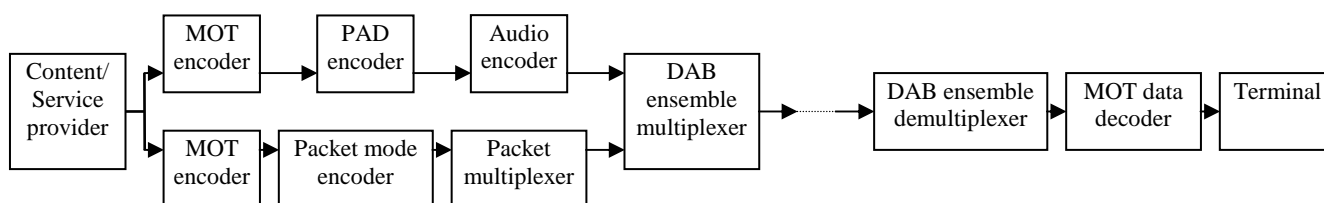


Figure 1: Overview of MOT entities

5 Structural description

This clause describes the different operations needed in order to transmit a file or a set of files in MOT format on DAB. MOT provides two modes of operation - directory mode and header mode. The directory mode should be used where a set of files need to be managed (file addition, deletion and modification), whereas header mode can be used for applications that have less stringent requirements. The data application specification describes which mode shall be used.

The directory mode provides a management mechanism for broadcasting MOT objects in a data carousel. A data carousel is a delivery system that allows an application server to present a set of distinct objects to an application decoder by cyclically repeating the contents of the carousel.

Within a data carousel, a directory is used to provide a complete description of the contents of the carousel, together with sufficient information to find the data for each described object. Version control mechanisms applied both to the objects within the carousel and the directory itself provides the ability to correctly manage updates to the carousel with minimum effort and at all times ensure that the correct version of an object is used by the application.

If an application requests a particular object, the receiver can easily determine by looking in the directory whether or not the requested objects exists within the carousel and where to find the object data. If the object the application requests is not stored in the receiver it may simply wait for the next time that the object is broadcast. If desired, the receiver may optionally implement caching strategies to reduce the latency of accesses by the application decoder and improve the performance of the carousel.

The first step in the transmission process is to identify the file and to create the MOT header. The MOT header contains both pure file identification and additional information. The file is referred to as the MOT body. At this stage the MOT header and MOT body correspond to a MOT object, which is ready for segmentation.

The header of a MOT object can be transported on its own (in an MSC data group of type 3) or as a part of a MOT directory (in an MSC data group of type 6).

NOTE: All the steps of the MOT encoding described in this clause are summarized in figures 2 to 5.

5.1 Segmentation of objects

The lowest common structure for the two different transport mechanisms for which MOT is defined (Packet Mode and X-PAD) is MSC data groups. This data group structure is also mapped to PAD transport for compatibility reasons. It is therefore the goal of the segmentation to map the MOT object into MSC data groups.

The MOT header and body are transported in different types of MSC data group and therefore the segmentation will apply independently on header and body. The MOT header will be split up in header segments with equal size and a last header segment with the size of the remaining bytes of the MOT header. The MOT body will be split up in body segments with equal size and a last body segment with the size of the remaining bytes of the MOT body. The size of header segments and the size of body segments are independent as well as the size of the last header segment and the size of the last body segment.

To elaborate a proper segmentation strategy the following considerations should be taken into account:

- minimize the overhead;
- improve the robustness of the transmission;
- facilitate the segment management on MOT data decoder side.

The link between an MOT header (data group type 3) and its MOT body (data groups type 4 or 5) is established by the TransportId. The TransportId of the MOT header and its MOT body is the same. If scrambled MOT objects are used, the CA messages related to this object also have the same TransportId.

5.1.1 Segmentation of the MOT body

The segments of the MOT body shall be transported in MSC data group type 4 if they are not scrambled, and in MSC data group type 5 if they are scrambled.

Segmentation is applied considering the size of the MOT body and the segmentation strategy.

5.1.2 Segmentation of the MOT directory

The segments of the MOT directory shall be transported in MSC Data group type 6. The reason for carrying the MOT directory segments in its own MSC data group type is:

- to enable the MOT data decoder to focus on recovering the MOT directory. The MOT data decoder can, based on the knowledge gained from the MOT directory and its own resources, decide to decode or ignore the body segments of MOT objects.

Segmentation is applied considering the size of the MOT directory and the segmentation strategy.

The MOT directory is never scrambled.

The link between a directory entry inside an MOT directory (data group type 6) is established by the TransportId. The TransportId of a directory entry and its MOT body is the same. If scrambled MOT objects are used, the CA messages related to this object also have the same TransportId.

5.1.3 Segmentation of the MOT header

The segments of the MOT header shall be transported in MSC Data group type 3. The reasons for carrying the MOT header segments in its own MSC Data group type are:

- to enable the MOT data decoder to focus on recovering the MOT header. The MOT data decoder can, based on the knowledge gained from the MOT header and its own resources, decide to decode or ignore the body segments of the MOT object related to this MOT header;
- to enable MOT header insertion which is used to reduce the time to collect all segments of the related MOT body in case of for instance switching on the MOT data decoder.

In order to enable easier access to the header information and to reduce the memory demand in the reassembly unit of the MOT data decoder, it is recommended to send the MOT header in one MSC data group (which is equivalent to no segmentation).

The MOT header is never scrambled.

5.2 Mapping of MOT segments into MSC data groups

After the segments have been defined they are associated with a segmentation header and mapped directly into the MSC data group data field. The signalling for recovering all the segments of the MOT object is done in the MSC data group session header (Last flag, Segment number and TransportId), see [1].

5.3 Packetizing segments - Network level

This clause describes the process for applying the MSC data groups onto the different transport channels. Only the MOT specific description is provided.

5.3.1 Packet Mode

The MSC data groups are packetized as described in [1]. The Command flag on network level in Packet Mode is used to distinguish between command packets (carrying for instance CA commands) and data packets. This Command flag is set to 1 for packets transporting the MSC data groups type 1, and set to 0 for packets transporting the MSC data groups type 3, 4, 5 and 6.

5.3.2 X-PAD

The MSC data group is mapped 1 to 1 into an X-PAD data group. Considering this, the X-PAD data group will be split into X-PAD data subfields for transportation as described in [1]. Since the size of the X-PAD data group can not be implicitly determined on network level in PAD as the size of the MSC data group in Packet Mode, the X-PAD data group Length indicator (X-PAD Application type 1) has to be used. This X-PAD data group Length indicator signals the total size of the following X-PAD data group. It has to be transmitted immediately before the first X-PAD data subfield of the data group it applies to.

MSC data groups containing MOT data (MSC data groups type 3, 4, 5 and 6) are transported in X-PAD Application type 12 and 13 where 12 indicates start of X-PAD data group and 13 continuation of X-PAD data group.

MSC data groups containing CA messages (MSC data group type 1) are transported in X-PAD Application type 14 and 15 where 14 indicates start of CA message and 15 continuation of CA message. The differentiation between X-PAD Application types 12/13 and 14/15 is similar to the mechanism of the command flag in Packet Mode.

For further details on the transmission strategies concerning X-PAD refer to [1].

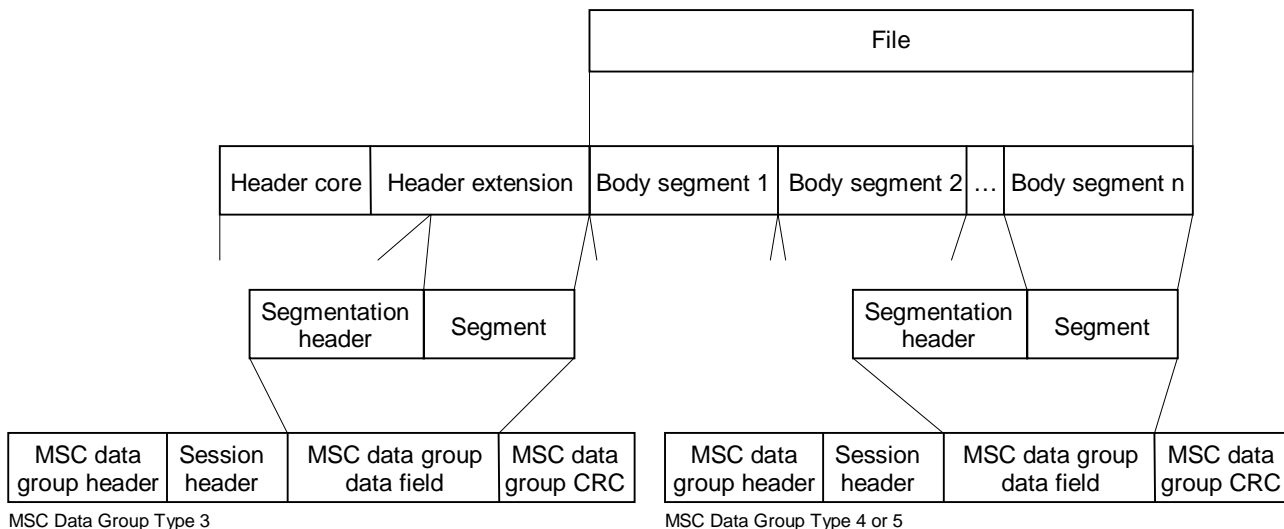


Figure 2: MOT segmentation process for an MOT object

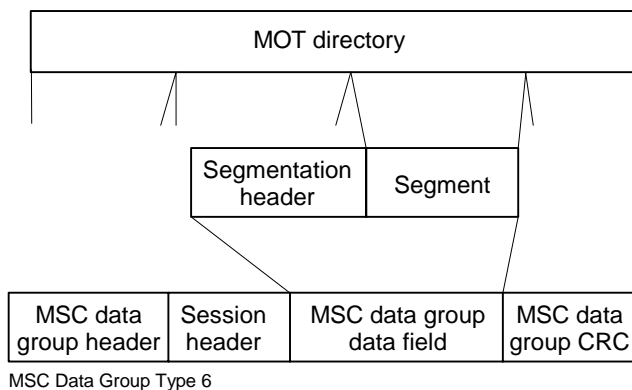


Figure 3: MOT segmentation process for an MOT directory

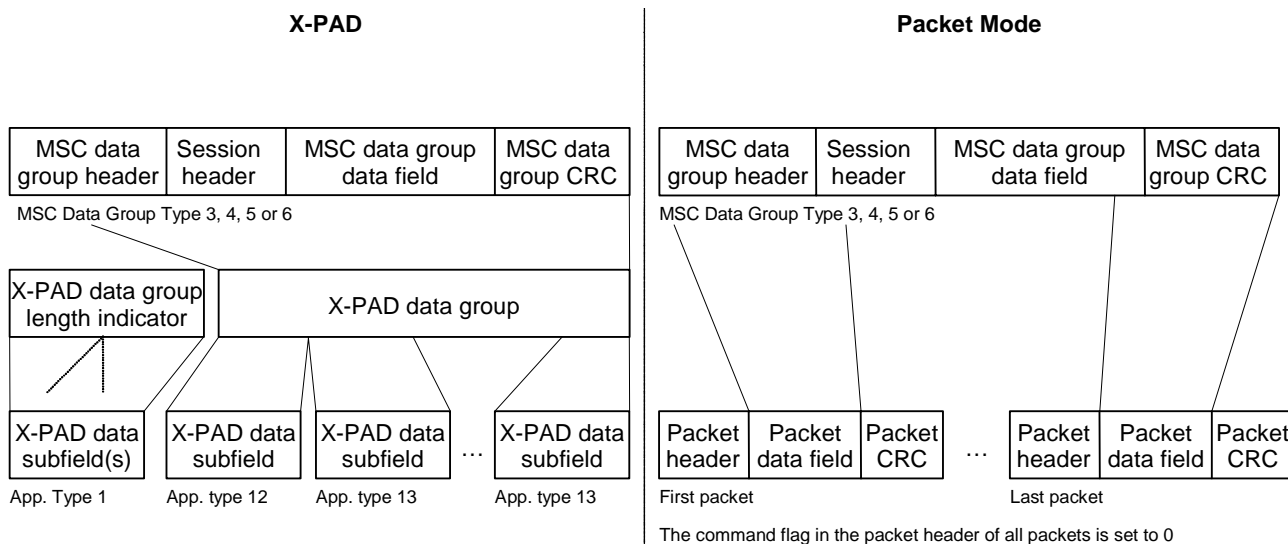


Figure 4: Splitting up MSC data groups type 3, 4, 5 and 6 into X-PAD data subfields and packets

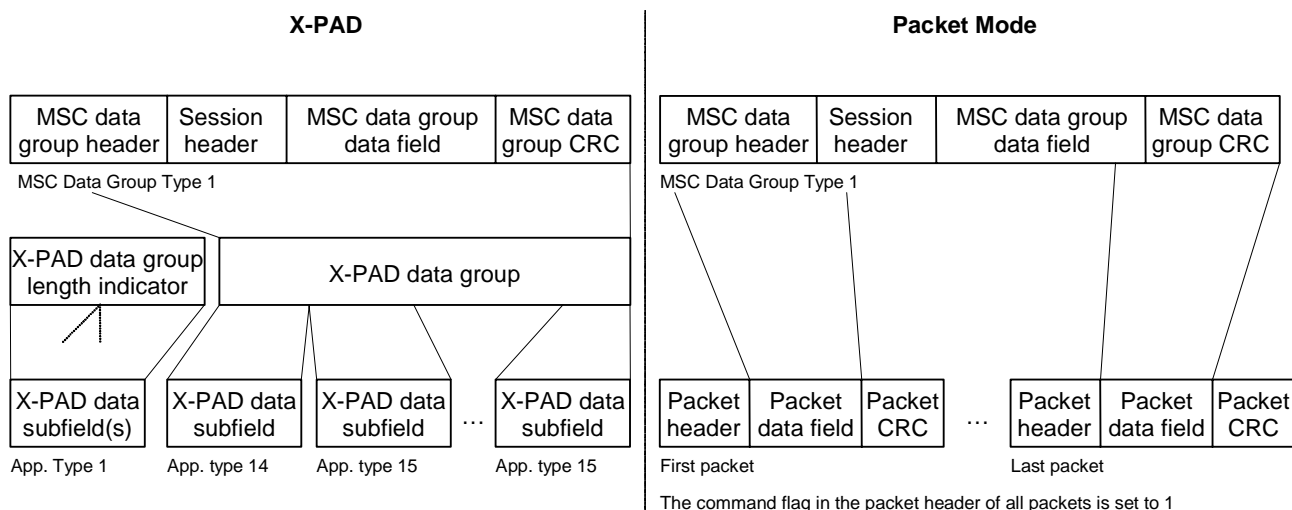


Figure 5: Splitting up MSC data groups type 1 into X-PAD data subfields and packets

6 Fields of the MOT Header

The MOT Header consists of two parts: The header core and the header extension. The header core is a set of four sub fields that shall be specified, while the header extension is a variable length field that may contain an arbitrary number of "parameters" to be associated with each file.

6.1 Header core

The header core fields are BodySize, HeaderSize, ContentType and ContentSubType. The first two sub fields simply indicate the length, in bytes, of the body and header respectively. The ContentType and ContentSubType pair of sub fields may be used to indicate that the associated file is of a particular file type, where that type identifier is taken from an enumerated list.

In many applications, this list may be sufficient to define all the possible types of file that may be used by the application. However, since the enumerated list is necessarily constrained to the set of types that have been registered, an application may choose to use an alternative mechanism to determine file type, if needed, such as using an application specific parameter (see below). Where an application does not need to use one of the enumerated types listed in the specification [1], these fields may be set to 0 to indicate general data/object transfer.

6.2 Header extension

Each parameter in the header extension begins with a length indicator followed by a parameter identifier. The parameter identifier then determines how the following data fields are to be interpreted. The parameter identifier field (ParamId) is divided into two ranges: 0 to 31 and 32 to 63.

The first set of ParamId values is used for parameters that may be of general applicability to a wide range of applications. Of particular note is the ContentName parameter which can be used to name each file within a carousel.

The use of the second set of ParamId values is entirely determined by the application specification. An example of an application specific parameter, taken from the Broadcast Web Site application specification, is the MimeType parameter which declares the type of each file in the form of a mime type identifier (e.g. "text/html", "image/jpeg", etc.).

It is important to note that the precise use of parameters to control access to a carousel is defined by the application specification itself, not the MOT specification.

6.2.1 Naming files and directory structures

As previously discussed, the ContentName parameter is typically used to provide a name for each file within a carousel. However, the MOT protocol implements a simple Data Carousel that does not support more complex carousel concepts such as directories. Accordingly, if an application needs to be aware of the notion of a directory structure, then the directory hierarchy shall be specified through the use of fully qualified paths for the value of the ContentName parameter.

For example, if we wish to represent two directories ("dir1" and "dir2"), each containing one file ("file1" and "file2") using an MOT carousel, then the carousel will contain two files with the ContentName parameter set to "dir1/file1" and "dir2/file2" respectively.

7 Model of a MOT data decoder and its interfaces

The model describes the functionality of the MOT data decoder on different levels including the interfaces to the DAB receiver and the terminal (see also figure 1). Real implementations may be quite different, optimized according application specific needs and receiver design constraints etc.

The relations between all the levels described in this clause are summarized in figure 6.

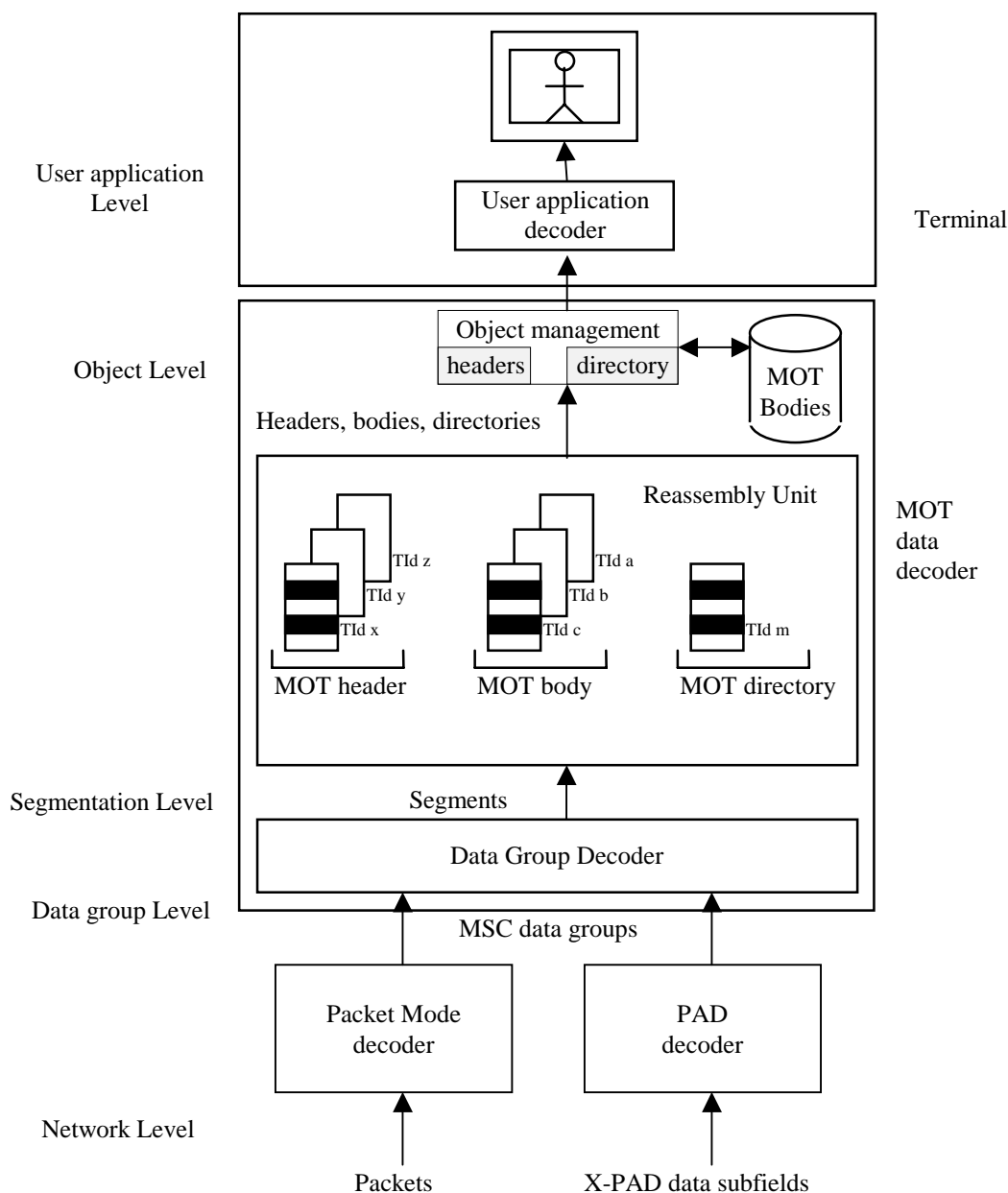


Figure 6: General description of a MOT data decoder and its interfaces

7.1 Network level

At the network level packets and/or X-PAD data subfields are processed as described in [1] and complete Data Groups are passed to the data group level.

In packet mode, the packet address has to be used to identify a particular service component within a subchannel. The packets are collected by taking into account the Continuity index. The length of the data group is derived by the First/Last flag and the Useful data length in the packet headers. The validity of each packet is verified by the evaluation of the packet CRC.

In X-PAD mode, the MSC data group length is derived from the data group Length indicator (Application type 1) immediately preceding the start of the MOT data group.

7.2 Data group level

The validity of each single MSC data group is verified by the evaluation of the MSC data group CRC. A MOT data decoder needs to decode only data group types 1, 3, 4, 5 and 6. Data groups with other Data group types are discarded. The MSC data group data field contains a complete segment (including the segmentation header with RepetitionCount and SegmentSize). The corresponding segment number and the TransportId are provided by the session header of the data group. A CRC checked segment, together with its corresponding TransportId and segment number will be passed to a re-assembly unit working at the segmentation level.

7.3 Segmentation and object level

The reassembly unit reassembles segments with the same TransportId. The reassembly unit processes data groups with Data group type 3 (MOT header), type 4 (MOT body), type 5 (MOT body and CA parameters), type 6 (MOT directory) and type 1 (CA messages).

7.3.1 General description of the MOT data decoder

The MOT data decoder consists of two parts:

- The reassembly unit reassembles headers, bodies and the MOT directory.
- The object management controls the reassembly unit, stores the received objects and handles requests by the user application.

In this general description two operation modes of a MOT data decoder are described:

- The MOT header mode: in this mode MOT headers and bodies are processed.
- The MOT directory mode: in this mode the MOT directory and bodies are processed.

Both reassembly unit and object management unit are in the same mode.

If a stream contains both headers and directory, the MOT data decoder may work in one or the other mode.

7.3.2 The reassembly unit

The functionality of the reassembly unit depends on its operation mode:

- MOT header mode: in this mode MOT headers and bodies are reassembled (Data group types 3 and 4; for CA also Data group types 1 and 5).
- MOT directory mode: in this mode the MOT directory and bodies are reassembled (Data group types 6 and 4; for CA also Data group types 1 and 5).

The reassembly unit continuously evaluates the incoming data groups. It shall be prepared that several objects are transmitted applying interleaving, so that they are to be decoded quasi in parallel. It is not required by the reassembly unit to evaluate the data group Repetition index or the RepetitionCount of the segmentation header.

7.3.2.1 MOT directory mode

The MOT directory is reassembled and forwarded to the object management. If the MOT directory is forwarded, its TransportId is stored inside the reassembly unit. From now on all MOT directory data groups transported with this TransportId are discarded. If a MOT directory data group is received with a different TransportId this means that the MOT directory is updated and therefore the new MOT directory is reassembled and forwarded and its TransportId stored.

The object management orders the reassembly unit to reassemble MOT bodies. It is up to the object management to assure that there is enough memory to store these bodies and still reassemble and forward a new MOT directory.

7.3.2.2 MOT header mode

When a MOT header is forwarded (to the object management), its TransportId is kept in a list inside the reassembly unit. From now on all MOT header data groups with one of the TransportIds in this list can be ignored, because they are already known by the Object management. So the list contains the TransportIds of all headers to be discarded. If the object management removes a MOT header from its memory, it informs the reassembly unit about the TransportId of the removed header. The reassembly unit will then remove this TransportId from its list and thus accept a MOT header with this TransportId again.

The object management orders the reassembly unit to reassemble MOT bodies. It is up to the object management to assure that there is enough memory to store these bodies and still reassemble and forward new MOT headers.

7.3.2.3 Segmentation of MOT bodies

Reassembly of MOT bodies is independent from the mode the reassembly unit is in. If the object management requests a body, the reassembly unit gets a request indicating which bodies are to be reassembled. This request will include the TransportId and size of the bodies and maybe also the SegmentSize (if given in the MOT directory). The reassembly unit can thus allocate memory for the requested bodies.

The TransportId is not only used to reassemble all segments of a header or a body of a MOT object, but also to establish the link between header and body and to link them to the related CA messages, if applicable. If the reassembly unit is ordered to reassemble a body it shall collect data groups of type 4 (unscrambled body) or data groups of type 5 and 1 (scrambled body/CA messages).

7.3.3 The object management unit

The object management stores objects and permits the user application to request objects, e.g. by their ContentName or the object with a MOT label. The object management tries to reduce the object access time and thus includes some caching strategy.

Although the terminal will allocate the greatest part of its memory to the storage of the objects, it shall be prepared for situations, where the size and number of objects exceeds the memory resources. In such cases, the object management can make use of additional information on the relevance and availability of the object, i.e. by the parameters RetransmissionDistance and Priority in the header/directory of the object for optimal use of memory.

The object management unit evaluates the MOT parameters and makes these parameters available to the user application. If an object expires due to an ExpireTime parameter, this object is removed from memory and it is signalled to the user application that the object is no longer valid and should be removed from the presentation. No object is forwarded to the application layer before its StartValidity.

According to the model described in this clause, it is the object management which orders the reassembly unit to decode the multiple bodies. If memory capacity of the object management allows, the object management will order to decode all body segments as they are incoming, either sequentially one after the other or interleaved, quasi in parallel. (Since it already knows the size of the bodies from the header/directory, it can assure that there is enough memory for the reassembly unit to hold all bodies reassembled in parallel and the headers/directory).

The functionality of the object management depends on its operation mode.

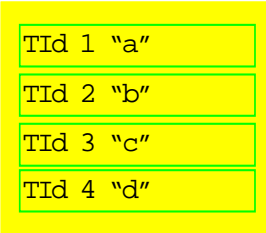
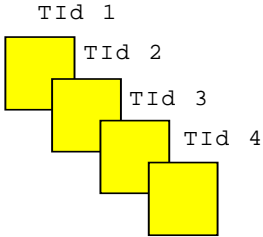
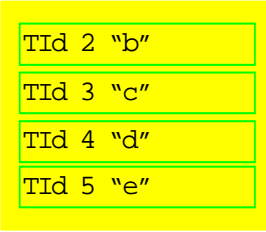
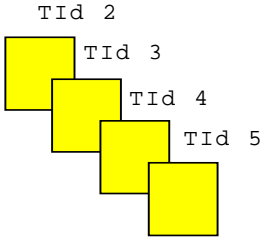
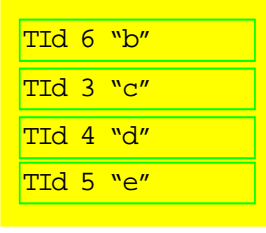
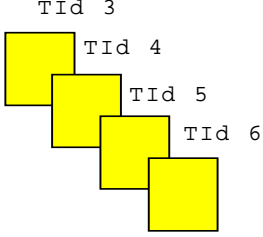
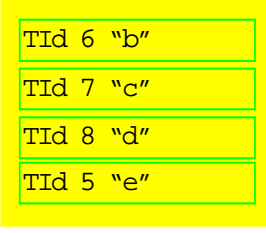
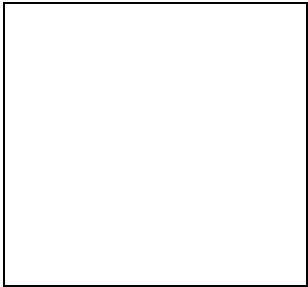
7.3.3.1 MOT directory mode

If a MOT directory is transmitted, the object management will be informed by the reassembly unit after the MOT directory is completely reassembled. The object management can then order the reassembly unit either to reassemble all incoming bodies or- in case of memory shortage - to select the specific objects that are to be reassembled, e.g. by starting to request first the object with a MOT label or objects with a high Priority parameter until all objects have been received. If there is not enough memory to hold all objects within the carousel, the object management uses a caching strategy that determines which objects should be stored. A very simple caching strategy would require the reassembly unit to reassemble only the body of the object that is currently requested by the user application and to cache these bodies. A more advanced strategy will try to load bodies before they are requested by the user application and thus reduce access time.

To improve the startup time when a new service is selected by the user, the object management orders the reassembly unit to forward all completely reassembled bodies regardless of their TransportId even before the MOT directory is received. Note however that this request will not include the size of the bodies. Therefore the reassembly unit cannot allocate memory for all bodies. The object management should store these bodies so that they can be accessed by the user application immediately after the directory has been received.

If a new directory is forwarded by the reassembly unit, the object management compares the old directory and the new one and removes all objects from the cache that are no longer signalled within the directory. If an object is transmitted with a different TransportId but still has the same VersionNumber parameter, this means, that there might have been a change in the header of the object or in the segmentation of the object, but the body of the object is still the same. Therefore the object management can keep the body of an object if its VersionNumber and the VersionNumber signalled within the directory are the same.

EXAMPLE: TId is used as an abbreviation for TransportId. If an object appears more than once in the example, it is assumed that it has the same content, i.e. the same header and body.

MOT directory	Actions to do after reception of the MOT directory	Objects in the cache
<p>TId 100</p> 	<ul style="list-style-type: none"> ▪ store objects with TId 1, 2, 3, 4 	
<p>TId 101</p> 	<ul style="list-style-type: none"> ▪ delete object with TId 1 ▪ keep objects with TId 2, 3, 4 ▪ store object with TId 5 	
<p>TId 102</p> 	<ul style="list-style-type: none"> ▪ delete object with TId 2 ▪ keep objects with TId 3, 4, 5 ▪ store object with TId 6 	
<p>TId 103</p> 	<ul style="list-style-type: none"> ▪ delete objects with TId 3, 4 ▪ keep objects with TId 6, 5 ▪ store objects with TId 7, 8 	

7.3.3.2 MOT header mode

The object management can uniquely identify and access an object - at least within one MOT stream - by its ContentName.

In MOT header mode the reassembly unit continuously checks for headers and forwards these to the object management.

All received MOT headers forwarded from the reassembly unit are collected to form a list of currently transmitted objects; this list will be called LH (list of headers) from now on.

If a header is forwarded from the reassembly unit, the object management checks if a header for the same object is already in the LH; this is done by comparing the ContentNames. If a header with the same ContentName is found, the object management compares the VersionNumbers of the two headers. If the VersionNumbers are different, the body of this object (if present) shall be removed from memory. Note that the reassembly unit will not forward headers with the TransportId of an already stored header.

If a header of an object is received again but with a different TransportId, the new header replaces the old one. The TransportId of the old (removed) header is signalled to the reassembly unit so that the reassembly unit is prepared that from then on the same TransportId may be used by the provider for a new object later on. From then on the reassembly unit accepts a header with this TransportId. If the body of a header is not already stored and especially if the body is already requested by the user application, the object management will request the reassembly of the body. A request to the reassembly unit includes the TransportId and the size of all bodies that should be reassembled.

The content of the LH can be queried by the user application. The user application can e.g. request the object with the MOT label.

The terminal can never be sure that its LH contains the headers of all objects that are currently broadcast. Therefore if an object with a certain ContentName is requested and its header is in the LH, the object management knows that this object was once broadcast, but not if it is still broadcast. If the body is stored inside the MOT data decoder, it can be forwarded to the user application. If it is not stored, the object management shall order the reassembly of this body. But since the MOT standard does not require that the TransportId of an object is kept the same as long as the object is not changed, the object management shall not assume that the TransportId is the same as in the header already stored in the LH. The object management can order the reassembly unit to reassemble the body with the given TransportId, but if the header with the requested ContentName is received again but with a different TransportId, the object management unit shall forward the new TransportId of the body to the reassembly unit.

If no header from a requested object is stored in the LH, the object management shall wait until the header with the requested ContentName is received and immediately after receiving the header it shall order reassembly of the body.

Since the MOT data decoder cannot assume that the requested object will ever be received, a timeout has to be used and after its expiration an error can be signalled to the user application.

If an object including its associated attributes, i.e. its parameters, has been successfully and completely assembled in the reassembly unit, the object is passed over to the object management in the Terminal. The object management will then evaluate the ContentName and the VersionNumber and, depending on the caching strategy and the user application, it will decide whether to pass the object directly to the user application decoder or to store or to discard the object. If a previous version of the object has been forwarded to the user application decoder and is currently presented and now a new version has been received, it shall be passed to the user application decoder.

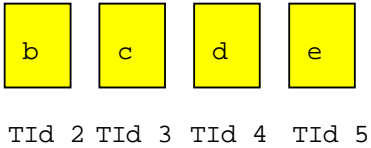
The object management can uniquely identify and access an object - at least within one MOT stream- by its ContentName together with its VersionNumber[^]. So these parameters may be used as key parameters for storing and retrieving header and body of an object in a data base, whereas the TransportId is valid only during the transport time of the object (see also clause 3.1 in [2]). It is not intended to be used as a key parameter for accessing the object.

Because the terminal will always keep only one version of an object, i.e. the latest received version, the ContentName (without VersionNumber) is already sufficient for the object management to identify the object. But in any case it has to keep track of the VersionNumber of that object: If a new version of an object is received, i.e. its VersionNumber is different from that of the stored object, the terminal will replace the old version, i.e. the memory of the old version will be released.

If the header of an object has changed and if this is signalled by a Header Update object (ContentType MOT Transport, second method for updating headers), the terminal will replace the header of the object with ContentName and VersionNumber by the extension parameters of the Header Update object. Note that a Header Update may not contain a VersionNumber, therefore the original VersionNumber shall be kept!

If the object management removes a header from its memory, this shall be signalled to the reassembly unit because the reassembly unit will forward every header with a given TransportId only once and after forwarding it to the object management it will ignore header segments with this TransportId.

EXAMPLE: TId is used as an abbreviation for TransportId. If an object appears more than once in the example, it is assumed that it has the same content, i.e. the same header and body. It is assumed that after reception of a header the object management orders the reassembly unit to reassemble its body.

Arriving objects	Actions to do after reception of an object	Objects in the cache	List of Headers (LH) after reception of the last object
	a -> store object with TId 1 b -> store object with TId 2 c -> store object with TId 3 d -> store object with TId 4		1 "a" 2 "b" 3 "c" 4 "d"
	b -> keep object with TId 2 c -> keep object with TId 3 d -> keep object with TId 4 e -> store object with TId 5		1 "a" 2 "b" 3 "c" 4 "d" 5 "e"
	b -> store object with TId 6 and delete object with TId 2 (the ContentName is the same but the TId has changed) c -> keep object with TId 3 d -> keep object with TId 4 e -> keep object with TId 5		1 "a" 6 "b" 3 "c" 4 "d" 5 "e"
	b -> keep object with TId 6 c -> store object with TId 7 and delete object with TId 3 d -> store object with TId 8, and delete object with TId 4 e -> keep object with TId 5		1 "a" 6 "b" 7 "c" 8 "d" 5 "e"

The object "a" is deleted by the service provider with the MOT parameter ExpireTime or by the terminal when the cache is full.

7.3.4 Advanced caching

The MOT directory or a MOT header contains object descriptions of broadcast objects and their TransportIds, but it does not signal when which body is broadcast. Therefore an object management strategy that reassembles objects with the highest priority first might have a bad startup time since all other objects broadcast before these objects are ignored. The reassembly unit might signal to the object management which body TransportIds are currently broadcast. E.g. if the first segment of any body is received, this could be signalled to the object management unit. The object management could then order the reassembly of this body.

Using the time when the first body segment is received and the RetransmissionDistance parameter of the object can also be used by the object management unit to predict the time of the next retransmission of the body. This permits advanced caching strategies.

It is also possible to evaluate the data group Repetition index (see clause 5.3.3.1 in [1]) or the RepetitionCount of the segmentation header (see clause 6.1.1 in [2]).

However, such additional processing by the reassembly unit and object management unit is just optional.

7.4 User application level

The user application level requests objects from the MOT data decoder and presents them. The specification of the user application level is not a part of MOT.

8 Transmission Mechanisms

8.1 General

When transmitting data in a radio broadcast system, the service provider always has to take in consideration that the terminal may miss data due to:

- The receiver being switched off, out of signal or tuned to another service or ensemble.
- Bad reception due to the conditions of the radio channel (bit errors).

Without an interaction channel the receiver does not have the possibility to demand retransmission of lost data. Therefore MOT uses some mechanisms which permit data to be repeated and so allow the receiver several opportunities to receive objects and to ensure the correct reception:

- Repetition on data group level.
- Repetition on object level.
- Retransmission of objects.
- Insertion of MOT headers.

Transmission errors may cause the receiver to fail in decoding the data, therefore it is strongly recommended to use one or more of the four mentioned mechanisms to ensure the reception, although the use of them decreases the useful bit rate.

MOT also provides another mechanism in order to permit the transmission/reception of objects in parallel: Interleaving of objects in an MOT stream.

Interleaving can be used to insert high priority objects into the MOT stream during transmission of big objects with a long transmission time.

8.2 Data transmission

This clause details the main differences and characteristics of the single object transmission and multiple object transmission mechanisms.

8.2.1 Single object transmission

The non-cyclic transmission scheme (see figure 7) is useful when only one object is needed at a time, for example in a slide show.

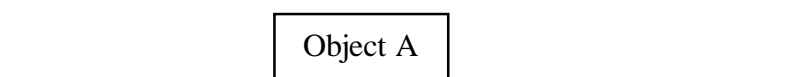


Figure 7: Single object transmission

Repetition and insertion of MOT headers (see figure 8) can be used to ensure reliable reception.

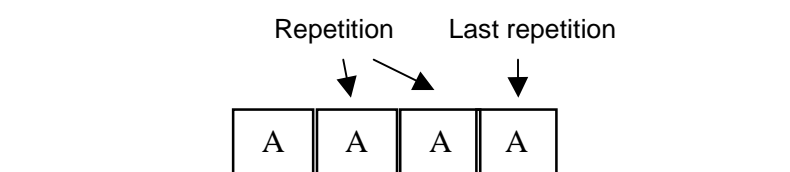


Figure 8: Single object transmission with repetitions

8.2.2 Multiple object Transmissions

Multiple object transmission is intended for user applications that need to have several objects available on the terminal at the same time. If the terminal requests a MOT object that is not already cached, it has to wait for the next cycle. An example for such a user application is a broadcast web site.

Each object is transmitted several times in a cyclic manner with a cycle time between each transmission (see figure 9). The cycle time between subsequent transmissions of an object may vary.

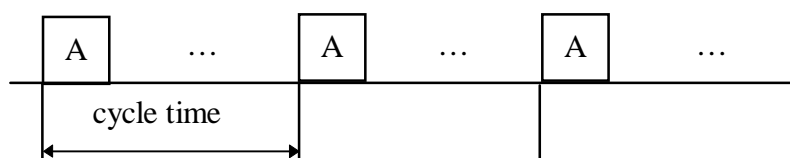


Figure 9: Multiple object transmission

Within a cycle, repetition and insertion of MOT header can be used to ensure the reception (see figure 10). In a new cycle the object is retransmitted. The content of the object can be the same or can be updated.

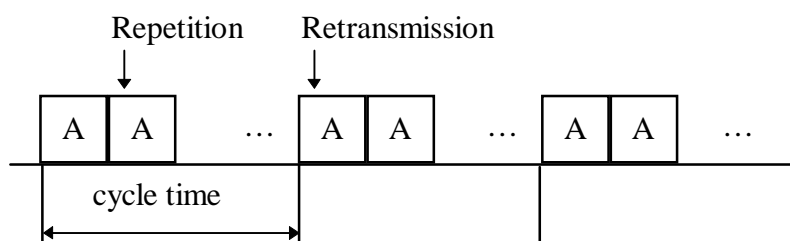


Figure 10: Multiple object transmission with repetitions

The different objects in the cyclic MOT stream are identified by their ContentNames. If in a retransmission the Header-, Body- or SegmentSize of an object have been changed, the TransportId shall be changed. If the body has been updated, the VersionNumber shall be increased by 1 modulo 256. If nothing is changed, the TransportId should remain the same.

It is recommended to transmit the most important objects (e.g. the HTML pages on the top of the hierarchy or the most visited pages of a broadcast web site) more frequently than the others to improve the access to the service.

8.3 Update, addition and deletion of objects

When objects are updated, added or deleted at the broadcast server, the corresponding change is signalled to the receiver.

An object is updated if its content (body), header or segmentation are changed.

A new object always uses a new TransportId.

When directory mode is used, the update, deletion and addition of objects is easily detected by the receive terminal comparing the new MOT directory (which has a new TransportId) with the old one.

When an object is modified, the VersionNumber shall be incremented.

A new object may be transmitted at any time.

When an object is deleted, the signalling method depends on whether directory mode or header mode is being used.

8.3.1 Directory mode

The directory object describes the exact state of the data carousel. Therefore, when the content of the carousel is changed, the directory object is also changed. The same procedure therefore applies whether the change of content is due to the update, addition or deletion of an object in the carousel.

The MOT directory shall be updated if there is any change to the information carried in the MOT directory. This situation occurs when:

- one or more of the fields in the MOT directory describing the data carousel is changed;
- one or more of the directory entries is changed.

The addition or deletion of an object from the data carousel causes an update of the MOT directory since this affects the content of the directory entries.

An update of an object carried in the data carousel also causes an update of the MOT directory since the TransportId and VersionNumber carried in the directory entry for these object have to be changed.

The update of the MOT directory is signalled by a change of TransportId. When the MOT data decoder receives a Data group type 6, it shall compare its TransportId with the TransportId for the current MOT directory. If they not are equal, the MOT directory has been updated and the MOT data decoder shall try to reassemble the new MOT directory in order to replace the old MOT directory. The new MOT directory becomes valid as soon as it is completely reassembled. When the new directory becomes valid, the old directory becomes invalid and should be deleted.

When a new MOT directory becomes valid, the object management unit shall:

- update parameters describing the carousel;
- delete objects not longer listed in the directory;
- update objects that have been subject to changes:
 - if the TransportId has been changed but the VersionNumber for an object remains the same, only the parameters describing the object have been updated, but not the content of the body;
 - if both the TransportId and the VersionNumber have been changed, both the parameters and the body of the object have been changed. (It is not possible to update only the body and not the parameters of an object since at least the TransportId and VersionNumber parameters shall be changed when the body is updated);
- store objects listed in the directory. If there is not enough free memory to store all objects, the object management can apply a storing strategy based on information received from the directory, e.g. the priority of objects.

When an object body in the data carousel is updated, it shall be preceded by an update of the MOT directory.

The MOT directory may be updated at any time regardless whether the data carousel has completed a full turn or not.

To ensure that as many receivers as possible receive the update of the MOT directory as quickly as possible, the data inserter may increase the repetition rate of the directory.

8.3.2 Header mode

If the information carried in the body of an object is updated, the complete new object has to be transmitted. While ContentName will be the same, the VersionNumber shall be incremented and the BodySize may also change. Therefore a new body and a new header have to be transmitted. The segments of the updated object will be transmitted with a new TransportId.

A new object can be added at any time. It will have a new ContentName and will have a new TransportId.

When an object is deleted, a header update with the corresponding ContentName and no VersionNumber (thus it refers to all versions) and the ExpireTime set to "Now" shall be transmitted as soon as possible. The body does not need to be transmitted any longer. The updated header should be transmitted a few times to allow all receiver terminals to delete the object.

MOT provides two methods to signal a change in the header information of an object:

- changing the TransportId;
- sending a Header Update.

The first method shall be used, when the header information is to be changed and the transmission of the body is to be continued. Then the transmission of the new object with new header (but with the same body) is started, while segmentation (for both, the body and the updated header) is done with a new TransportId.

The second method shall be used, when only the header is changed, but the body is not sent anymore (for example when an object is deleted). With the HeaderUpdate the TriggerTime(s) can be overwritten, before the (last original) TriggerTime becomes valid.

Regardless of which of the two methods have been applied by the transmission side, the MOT data decoder on reception side shall pass the object (at least the updated header information) to the user application decoder.

Annex A: Coding

A.1 The MSC data group

A.1.1 Coding of the MSC data group

The MSC data group is the lowest common level for the transport of MOT in Packet Mode and PAD. This clause will try to explain the coding of the MSC data group relevant for MOT and make a short example of how to code one.

The following parameters/flags occur in the MSC data group. For the structure of the MSC data group and the individual coding of the parameters please refer to [1].

MSC data group header	Description
<i>Extension flag</i>	Indicates whether the Extension field is present or not. In case of MOT, the extension field is currently only present for the data group type 5 since the field and this type are specified for the transport of CA parameters.
<i>CRC flag</i>	Indicates whether there is a MSC data group CRC or not. In the case of transport of MOT objects in PAD it is more than strongly recommended that there is a MSC data group CRC (CRC flag set to 1) since there is no other error detection. In Packet Mode there is further error detection on network level but the MSC data group CRC is in this case useful for the detection of packet loss.
<i>Session flag</i>	The session flag indicates the presence of the last flag and segment number. It might always be set to 1 but it shall be set to 1 in the following cases: <ul style="list-style-type: none"> ▪ for the data groups containing MOT header, if the MOT header needs to be split into more than one segment; ▪ for the data groups containing MOT body, if the MOT body needs to be split into more than one segment; ▪ for the data groups containing MOT directory, if the MOT directory needs to be split into more than one segment. <p>The reason is that if there are several MOT segments the last flag and segment number are needed in order to identify individual segments.</p> <p>If there is only one segment in a MOT header, body or directory, it is recommended to set the Session flag to 0 for this part of the object and thereby omit the last flag and segment number.</p>
<i>User access flag</i>	The User access flag indicates if the User access field is present or not. In the case of MOT this flag shall be set to 1 since the User access field contains the TransportId which is mandatory for MOT objects.
<i>Data group type</i>	The Data group type indicates the type of data conveyed in the MSC data field. For MOT (DSCTy == 60, resp. AppTy == 12...15) the types 3, 4, 5 and 6 are used. <p>3: indicates that the data group contains a MOT header segment.</p> <p>4: indicates that the data group contains a MOT body segment.</p> <p>5: indicates that the data group contains an encrypted MOT body segment and CA parameters. If CA is used, Data group type 1 is used to carry CA parameters.</p> <p>6: indicates that the data group contains a MOT directory segment.</p>
<i>Continuity index</i>	Shall be used as defined in [1]. Continuity index is incremented for each data group of a certain type with a content different from the immediately preceding data group. Useful in order to detect whether complete data groups are lost during reception.
<i>Repetition index</i>	Shall be used as defined in [1]. The Repetition index is used to indicate repetitions on MSC data group level.
<i>Extension field</i>	For MOT only used for Data group type 5 and contains parameters related to CA.
<i>Session header</i>	Only present if the Session flag is set.
<i>Last</i>	Indicates whether this MSC data group is the last segment of a group of segments (corresponding to one MOT header or one body). If there is only one segment it is recommended to omit the last flag and segment number by setting the Session flag to 0. Although this is equal to set the Session flag, set the last flag to 1 and the segment number to 0 indicating last and only segment.

MSC data group header	Description
<i>Segment number</i>	Indicates the number of the segment related to that the first segment is numbered 0 and incremented for each new segment.
<i>User access field</i>	The User access field is present if the User access flag is set. In the case of MOT it shall present since the TransportId is located in the User access field and mandatory for MOT objects.
<i>Rfa</i>	Shall be used as defined in [1].
<i>TransportId flag</i>	Indicates the presence of the TransportId. In the case of MOT it shall always be set to 1 since the TransportId is mandatory. The TransportId flag also has influence of the parameter length in the User access field as follows: 0: no TransportId End user address field, max 15 bytes 1: TransportId, 2 bytes End user address field, max 13 bytes
<i>Length indicator</i>	Shall be used as defined in [1].
<i>TransportId</i>	The TransportId is mandatory for MOT, shall be coded as defined in [1] and used as described in other clauses of the present document.
<i>End User address field</i>	Shall be used as defined in [1].
MSC data group data field	The MSC data group data field contains the MOT segment which consist of the segmentation header and the MOT data (header or body) as specified in [2]. The maximum size of the MSC data group data field is 8 191 bytes as specified in [1].
MSC data group CRC	Present if the CRC flag is set. In the case of PAD it is strongly recommended to use the MSC data group CRC since there is no further error detection in PAD.

A.1.2 Examples of coding

We will now make three examples of the coding of the MSC data group.

A.1.2.1 Example 1: MOT object, no segmentation

We will transmit a file with the attributes:

File size	30 bytes
File type	text file, ISO Latin 1
File name	Testfile.txt

Using the following transport attributes for MOT:

Segmentation of MOT header	None
Segmentation of MOT body	None
Repetition	None on Transport or Object level
Transport	Packet mode
Continuity index	Is in this example set to zero indicating that each data group is the first of its type
TransportId	In this example set to 1010101010101010. Shall be set according to the rules described in the present document

Coding of first MSC data group (MOT header)

Parameter	Binary coding	Explanation
<i>Extension flag</i>	0	No extension field
<i>CRC flag</i>	1	CRC present
<i>Session flag</i>	0	No last flag and segment number
<i>User access flag</i>	1	User access field present
<i>Data group type</i>	0011	MOT header (DSCTy == 60)
<i>Continuity index</i>	0000	Continuity index from the previous data group of same type + 1, modulo 16
<i>Repetition index</i>	0	No data group repetition
<i>Extension field</i>	None	Omitted since no CA
<i>Last</i>	None	Omitted since only one segment. The complete MOT header is in the MSC data group
<i>Segment number</i>	None	Omitted since only one segment. The complete MOT header is in the MSC data group
<i>Rfa</i>	000	
<i>Id flag</i>	1	TransportId present
<i>Length indicator</i>	0010	Two bytes of TransportId, no End user address
<i>TransportId</i>	10101010 10101010	TransportId number according to the rules described in the present document
<i>End User address field</i>	None	No End user address
<i>MSC data group data field</i>	000 000000010110 00000000 00000000 00000001 1110 00000000 10110 000001 00000001 11001100 00001101 00000000 "Testfile.txt"	RepetitionCount, 3 bits no object repetition SegmentSize, 13 bits 22 bytes BodySize, 28 bits 30 bytes HeaderSize, 13 bits 22 bytes ContentType, 6 bits text ContentSubType, 9 bits ISO Latin 1 ContentName, 2 + 13 bytes Testfile.txt
<i>MSC data group CRC</i>	xxxxxxx xxxxxxx	2 bytes CRC check sum dependent on the content of the data group. Calculated as specified in [1]

Coding of second MSC data group (MOT Body)

Parameter	Binary coding	Explanation
<i>Extension flag</i>	0	No extension field
<i>CRC flag</i>	1	CRC present
<i>Session flag</i>	0	No last flag and segment number
<i>User access flag</i>	1	User access field present
<i>Data group type</i>	0100	MOT body (DSCTy == 60)
<i>Continuity index</i>	0000	Continuity index from the previous data group of same type + 1, modulo 16
<i>Repetition index</i>	0	No data group repetition
<i>Extension field</i>	None	Omitted since no CA
<i>Last</i>	None	Omitted since only one segment. The complete MOT body is in the MSC data group
<i>Segment number</i>	None	Omitted since only one segment. The complete MOT body is in the MSC data group
<i>Rfa</i>	000	
<i>TransportId flag</i>	1	TransportId present
<i>Length indicator</i>	0010	Two bytes of TransportId, no End user address
<i>TransportId</i>	10101010 10101010	TransportId number according to the rules described in the present document
<i>End User address field</i>	None	No End user address
<i>MSC data group data field</i>	000 00000000 11110 -----	RepetitionCount 3 bits SegmentSize 13 bits MOT body (file) 30 bytes
<i>MSC data group CRC</i>	xxxxxxx xxxxxxx	2 bytes CRC check sum dependent on the content of the data group. Calculated as specified in [1]

A.1.2.2 Example 2: MOT object, segmentation

We will transmit a file with the attributes:

File size	1 000 bytes
File type	HTML file
File name	Test_html.htm

Using the following transport attributes for MOT:

Segmentation of MOT header	None
Segmentation of MOT body	500 byte segments
Repetition	None on Transport or Object level
Transport	Packet mode
Continuity index	Is in this example set to zero indicating that each data group is the first of its type
TransportId	In this example set to 1111000011110000. Shall be set according to the rules described in the present document

Coding of first MSC data group (MOT header)

Parameter	Binary coding	Explanation
<i>Extension flag</i>	0	No extension field
<i>CRC flag</i>	1	CRC present
<i>Session flag</i>	0	No last flag and segment number
<i>User access flag</i>	1	User access field present
<i>Data group type</i>	0011	MOT header (DSCTy == 60)
<i>Continuity index</i>	0000	Continuity index from the previous data group of same type + 1, modulo 16
<i>Repetition index</i>	0	No data group repetition
<i>Extension field</i>	None	Omitted since no CA
<i>Last</i>	None	Omitted since only one segment. The complete MOT header is in the MSC data group
<i>Segment number</i>	None	Omitted since only one segment. The complete MOT header is in the MSC data group
<i>Rfa</i>	000	
<i>TransportId flag</i>	1	TransportId present
<i>Length indicator</i>	0010	Two bytes of TransportId, no End user address
<i>TransportId</i>	11110000 11110000	TransportId number according to the rules described in the present document
<i>End User address field</i>	None	No End user address
<i>MSC data group data field</i>	000 00000000 10111 00000000 00000000 00111101 000 00000000 10111 000001 000000010 11001100 00001110 00000000 "Test_html. htm"	RepetitionCount, 3 bits no object repetition SegmentSize, 13 bits 23 bytes BodySize, 28 bits 1 000 bytes HeaderSize, 13 bits 23 bytes ContentType, 6 bits text ContentSubType, 9 bits html ContentName, 2 + 14 bytes Test_html.htm
<i>MSC data group CRC</i>	xxxxxxx xxxxxxx	2 bytes CRC check sum dependent on the content of the data group. Calculated as specified in [1]

Coding of second MSC data group (MOT body segment 1)

Parameter	Binary coding	Explanation
<i>Extension flag</i>	0	No extension field
<i>CRC flag</i>	1	CRC present
<i>Session flag</i>	1	Last flag and segment number present
<i>User access flag</i>	1	User access field present
<i>Data group type</i>	0100	MOT body (DSCTy == 60)
<i>Continuity index</i>	0000	Continuity index from the previous data group of same type + 1, modulo 16
<i>Repetition index</i>	0	No data group repetition
<i>Extension field</i>	None	Omitted since no CA
<i>Last</i>	0	first segment
<i>Segment number</i>	00000000 00000000	first segment
<i>Rfa</i>	000	
<i>TransportId flag</i>	1	TransportId present
<i>Length indicator</i>	0010	Two bytes of TransportId, no End user address
<i>TransportId</i>	11110000 11110000	TransportId number according to the rules described in the present document
<i>End User address field</i>	None	No End user address
<i>MSC data group data field</i>	000 00001111 10100 -----	RepetitionCount, 3 bits no object repetitions SegmentSize, 13 bits 500 bytes MOT body segment, 500 bytes, first half of file
<i>MSC data group CRC</i>	xxxxxxx xxxxxxx	2 bytes CRC check sum dependent on the content of the data group. Calculated as specified in [1]

Coding of third MSC data group (MOT body segment 2)

Parameter	Binary coding	Explanation
<i>Extension flag</i>	0	No extension field
<i>CRC flag</i>	1	CRC present
<i>Session flag</i>	1	last flag and segment number present
<i>User access flag</i>	1	User access field present
<i>Data group type</i>	0100	MOT body (DSCTy == 60)
<i>Continuity index</i>	0001	Continuity index from the previous data group of same type + 1, modulo 16
<i>Repetition index</i>	0	No data group repetition
<i>Extension field</i>	None	Omitted since no CA
<i>Last</i>	1	Last segment
<i>Segment number</i>	00000000 00000001	Second (and last) segment
<i>Rfa</i>	000	
<i>TransportId flag</i>	1	TransportId present
<i>Length indicator</i>	0010	Two bytes of TransportId, no End user address
<i>TransportId</i>	11110000 11110000	TransportId number according to the rules described in the present document
<i>End User address field</i>	None	No End user address
<i>MSC data group data field</i>	000 00001111 10100 -----	RepetitionCount, 3 bits no object repetitions SegmentSize, 13 bits 500 bytes MOT body segment, 500 bytes, second half of file
<i>MSC data group CRC</i>	xxxxxxx xxxxxxx	2 bytes CRC check sum dependent on the content of the data group. Calculated as specified in [1]

A.1.2.3 Example 3: MOT directory, no segmentation

We will transmit a MOT directory for a data carousel containing 2 objects:

Object 1

File size	30 bytes
File type	Text file, ISO Latin 1
File name	Testfile.txt
TransportId	1010101010101010

Object 2

File size	1 000 bytes
File type	HTML file
File name	Test_html.htm
TransportId	1111000011110000

The following transport attributes for the MOT directory will be used:

Segmentation of MOT directory	None
Repetition	None on transport or object level
Transport	Packet Mode
NumberOfObjects	2
CarouselPeriod	1,5 seconds
SegmentSize (in data carousel)	variable
DirectoryExtension	None
TransportId (for MOT directory)	1100110011001100

Coding of the MSC data group (MOT directory)

Parameter	Binary coding	Description
<i>Extension flag</i>	0	No extension field
<i>CRC flag</i>	1	CRC present
<i>Session flag</i>	0	No last flag and segment number
<i>User access flag</i>	1	User access field present
<i>Data group type</i>	0110	MOT directory (DSCTy == 60)
<i>Continuity index</i>	0000	Continuity index from the previous data group of same type + 1, modulo 16
<i>Repetition index</i>	0	No data group repetition
<i>Extension field</i>	None	Omitted since no CA
<i>Last</i>	None	Omitted since only one segment. The complete MOT directory is in the MSC data group
<i>Segment number</i>	None	Omitted since only one segment. The complete MOT directory is in the MSC data group
<i>Rfa</i>	000	
<i>TransportId flag</i>	1	TransportId present
<i>Length indicator</i>	0010	Two bytes of TransportId, no End user address
<i>TransportId</i>	11001100 11001100	TransportId number according to the rules described in the present document
<i>End User address field</i>	None	No End user address

Parameter	Binary coding	Description	
<i>MSC data group data field</i>	000 00000001 11110	RepetitionCount, 3 bits SegmentSize, 13 bits	no object repetitions size of the MOT directory segment, 62 bytes
<i>MOT directory</i>	00 00000000 00000000 00000000 111110 00000000 00000010 00000000 00000000 00001111 0 00	Rfu, 2 bits DirectorySize, 30 bits NumberOfObjects, 16 bits CarouselPeriod, 24 bits Rfu, 1 bit Rfa, 2 bits SegmentSize, 13 bits	shall be set to zero size of the MOT directory, 62 bytes 2 files in the data carousel 1,5 seconds shall be set to zero shall be set to zero size of MOT body segments in the data carousel, set to zero since differs between objects in the data carousel
<i>Directory Entry 1</i>	00000000 00000000 00000000 00000000 00000000 00000001 1110 00000000 10110 000001 000000001 11001100 00001101 00000000 "Testfile.txt"	DirectoryExtensionLength TransportId, 16 bits BodySize, 28 bits HeaderSize, 13 bits ContentType, 6 bits ContentSubType, 9 bits ContentName, 2 + 13 bytes TransportId, 16 bits	no DirectoryExtension field TransportId of Testfile.txt 30 bytes 22 bytes text ISO Latin 1 Testfile.txt TransportId of Test_html.htm
<i>Directory Entry 2</i>	11110000 11110000 00000000 00000000 00111101 000 00000000 10111 000001 000000010 11001100 00001110 00000000 "Test_html.htm"	BodySize, 28 bits HeaderSize, 13 bits ContentType, 6 bits ContentSubType, 9 bits ContentName, 2 + 14 bytes	1 000 bytes 23 bytes text html Test_html.htm
<i>MSC data group CRC</i>	xxxxxxx xxxxxxx	2 bytes CRC check sum dependent on the content of the data group. Calculated as specified in [1]	

History

Document history		
V1.1.1	July 2002	Publication