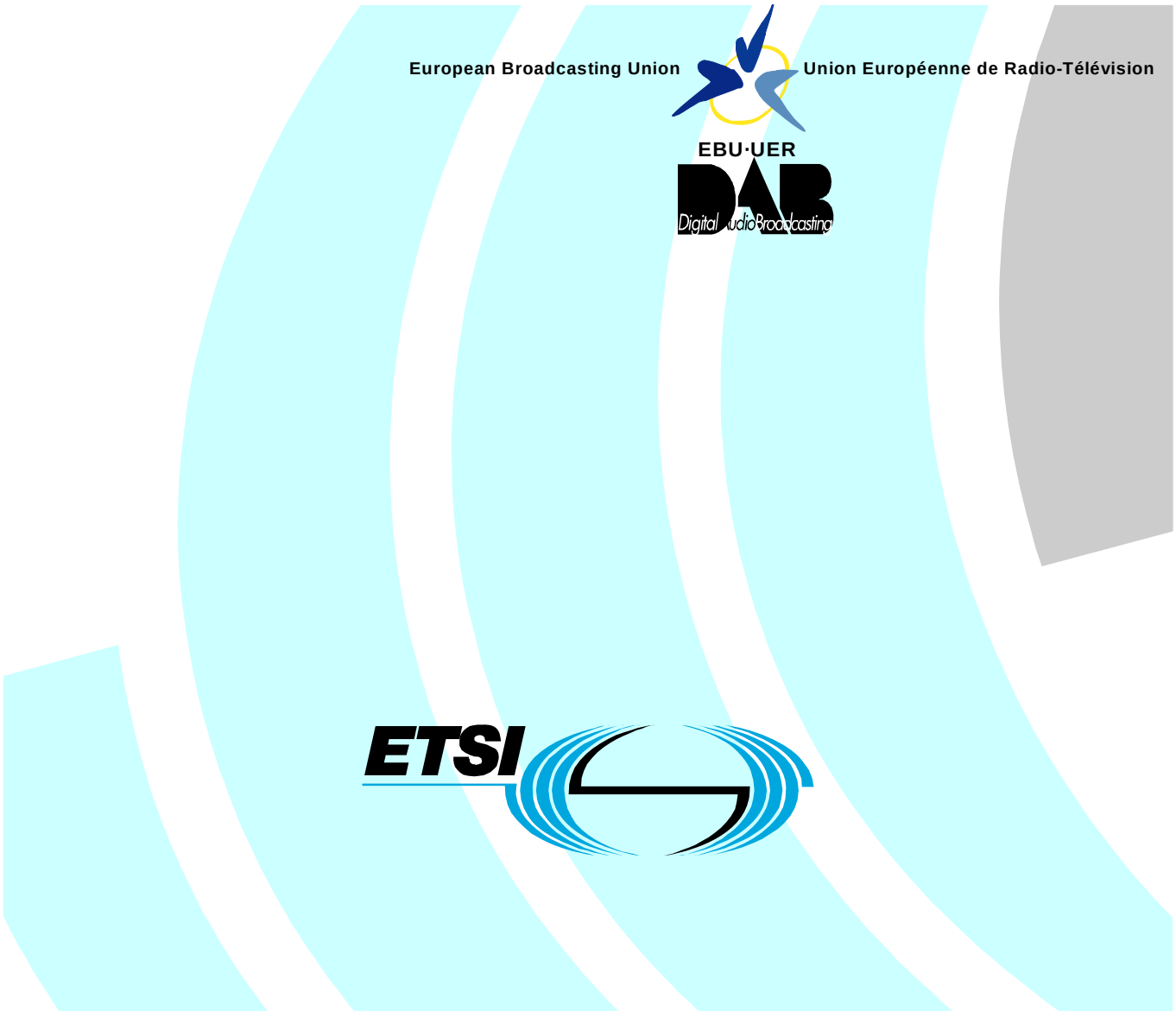
ETSI TS 102 979 V1.1.1 (2008-06)

Technical Specification

Digital Audio Broadcasting (DAB); Journaline; User application specification



Reference
DTS/JTC-DAB-53

Keywords
digital, DAB, DRM, multimedia, radio, text

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, please send your comment to one of the following services:
http://portal.etsi.org/chaircor/ETSI_support.asp

Copyright Notification

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 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 purpose of the present document is to describe and define the XML based low profile information service "Journaline®".

Journaline is a text based data application highly optimized for digital broadcast systems such as DAB (Digital Audio Broadcasting [1]) and DRM (Digital Radio Mondiale [3]).

The present document defines the overall functionality of the user application Journaline, specifies the transmission format and describes the expected behaviour of a Journaline compliant receiver.

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.

For online referenced documents, information sufficient to identify and locate the source shall be provided. Preferably, the primary source of the referenced document should be cited, in order to ensure traceability. Furthermore, the reference should, as far as possible, remain valid for the expected life of the document. The reference shall include the method of access to the referenced document and the full network address, with the same punctuation and use of upper case and lower case letters.

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 EN 300 401: "Radio Broadcasting Systems; Digital Audio Broadcasting (DAB) to mobile, portable and fixed receivers" .

[2] ETSI TS 101 756: "Digital Audio Broadcasting (DAB); Registered Tables".

[3] ETSI ES 201 980: "Digital Radio Mondiale (DRM); System Specification".

[4] International Phonetic Association: "International Phonetic Alphabet".

NOTE: Available at http://www.arts.gla.ac.uk/IPA.

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

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

[7] ISO/IEC 10646: "Information technology - Universal Multiple-Octet Coded Character Set (UCS)".

[8] W3C Recommendation: "Extensible Markup Language (XML) 1.0 (Second Edition)".

NOTE: Available at http://www.w3.org/TR/2000/REC-xml-20001006.

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.

Not applicable.

3 Definitions, abbreviations and conventions

3.1 Definitions

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

broadcaster/encoder: instance that encodes and provides a Journaline service

content section: part of a JML object after the header section that carries the useful content

DS payload: block of data contained in escape sequences of type "Data section Start" and "Data section Continuation"

header section: part of a JML object preceding the content section and carrying descriptive information on the JML object

hierarchy level: number of object IDs traversed on the shortest navigation path from the root object to the current JML object

history path: list of object IDs the user has traversed from the root object to the current JML object

hot button: function (e.g. a key that can be pressed) that allows a user to initiate a form of interaction linked to a JML object

Journaline Markup Language (JML): XML based binary encoding scheme used for the transmission of JML objects

JML object: encoded representation of an object

main menu: root object if it is of type "menu"

menu object: JML object of type "menu"

message object: JML object of type "plain-text message", "title-only message" or "list message"

object: information unit that can be rendered on screen, like a menu, a plain-text message, etc

object ID: unique identifier of a JML object within the Journaline service

point of entry: object that shall initially be presented to the user if no other point of entry is defined by other means; normally the root object

receiver: instance that decodes a Journaline service and presents it to the user

root object: JML object with object ID "0x0000", serving as the default point of entry to the Journaline service

static flag: signalling to the receiver that a JML object with a specific object ID will in future carry the same type of content. Therefore the user might choose to store the object ID as a favourite (bookmark) for quick access

TOC block: one block of management data carrying, among other information, a list of TOC entities, which represent all JML objects currently broadcast as part of the Journaline service

TOC fragment: if the complete list TOC entities cannot be carried in a single TOC block, the information is split up into multiple TOC blocks referred to as TOC fragments

TOC entity: description of a single JML object within a TOC block

3.2 Abbreviations

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

CRC	Cyclic Redundancy Check
DAB	Digital Audio Broadcasting
DRM	Digital Radio Mondiale
ID	IDentifier
IPA	International Phonetic Alphabet
JML	Journaline Markup Language
LSB	Least Significant Bit
MOT	Multimedia Object Transfer
MSb	Most Significant bit
MSB	Most Signigicant Byte
MSC	Main Service Channel
PAD	Programme Associated Data
RSS	Real Simple Syndication

NOTE: (XML based news distribution format on the Internet).

TOC	Table Of Contents
UTF-8	8-bit UCS/Unicode Transformation Format
XML	eXtensible Markup Language

3.3 Conventions

All numerical values shall be encoded as MSB-first and MSb-first
(MSB: Most Significant Byte, MSb: Most Significant bit).

Hexadecimal numbers are preceded by the string "0x", i.e. 0x20 is equivalent to 20_{16} (hexadecimal) or 32_{10} (decimal).

All text characters (including all textual parameter values) shall be encoded as UTF-8 [7] unless stated otherwise.

The order of bits and bytes within each description shall use the following notation unless otherwise stated:

- in figures, the bit or byte shown in the left hand position is considered to be first;
- in tables, the bit or byte shown in the left hand position is considered to be first;
- in byte fields, the Most Significant bit (MSb) is considered to be first and denoted by the higher number. For example, the MSb of a single byte is denoted "b_7" and the Least Significant bit (LSb) is denoted "b_0".

4 Overview

Journaline is a text based information service for digital radio, optimized for simple data aggregation and re-use, as well as highly efficient broadcast transmission. It supports the widest range of receiver types, from low-cost solutions with a small text display up to high-end receivers with graphical user interfaces and optional text-to-speech playback.

The radio user can instantly and interactively access all information provided by the radio station, comparable to teletext for TV. The core information is provided in simple textual form with the option for richer graphical representation including a future extension to multimedia elements like images or video sequences. The flexible coding of Journaline allows for future extensions as well as the provision of additional non-textual information in a fully backward compatible way.

The information is hierarchically organized based on menus. Every menu contains a list of sub-menus and/or messages. Messages carry one piece of information each (e.g. a list of football scores, a news item, a cooking recipe, a traffic message, etc.).

Messages and (sub-)menus will be referred to as "objects" ("menu objects" and "message objects") in the remaining document.

Journaline is based on common public standards:

- Objects can be generated and fed into a Journaline enabled transmission system as XML files (see [8]).
- Journaline capable receivers can easily restore an XML representation of each object for further processing (see [8]), or directly parse the transmitted binary representation for increased efficiency.

Journaline is highly optimized for use in digital broadcast systems and provides:

- Added value for listeners. Immediate access to textual information everywhere.
- Ease of use. Simple navigation through a minimum user interface consisting of a text screen plus up/down, select and back keys.
- Great benefit for all types of receivers. From low-cost receivers with small text display to high-end receivers with graphical user interface and optional speech playback.
- Increased listener loyalty for broadcasters. Minimum additional bit rate for extended listener support.
- Automatic re-use of existing information sources and feeds. Many text-based information sources like RSS feeds can be broadcast with minimum transformation effort.

The following technical principles ensure the stated functional highlights:

- All aspects of the Journaline specification are tailored towards low-footprint implementations even in low-complexity receivers as well as ease of use.
- To reduce the decoder footprint on low-complexity receivers, an object's main visual content is formatted as pure textual information.
- Every object is self-describing and self-contained. No global data structures need to be assembled or maintained in the receiver.
- Objects are broadcast in form of a data carousel. Data caching in the receiver is supported but not mandatory.
- Object transmission repetition can depend on individual access time requirements (priority class concept). For example, menu objects may be broadcast more frequently than message objects.
- To prevent unnecessary data overhead (and thereby minimize transmission time), objects are re-formatted for transmission into JML - the XML structured binary encoding "Journaline Markup Language".
- The provided receiver behaviour guidelines are a recommendation that should be fulfilled by any receiver according to its abilities and technical infrastructure. However, Journaline has a very carefully selected set of required functionality that encapsulates the core use to the user and that can easily be implemented by the simplest receiver.
- Receiver implementations can easily distinguish themselves by providing for example a richer graphical representation, Favourites functionality, or additional navigation support (like next and previous keys to speed up navigation through messages within the same menu).
- All aspects of the Journaline specification can easily be extended in a fully backward compatible way to support new types of information or extended functionality.

4.1 JML objects

Each piece of information (i.e. every menu, news article, ticker message, etc.) is transmitted to the receiver as a self-contained and self-describing "JML object". To minimize the decoder footprint in the receiver, the receiver does not need to build or store any overall hierarchy or index information to render a JML object on screen and to provide the user with the defined methods of interaction (like menu selections).

4.2 Types of JML objects

JML ("Journaline Markup Language"), an XML based binary coded content representation, structures a JML object's content into blocks of information like title, body text, menu items, etc.

Journaline currently specifies 4 types of JML objects:

- Menus.
- Plain text messages.
- Title-only messages.
- List messages.

These JML object types differ in their preferred way of presentation to the user as well as their behaviour in the case of a content update being received while the JML object is displayed.

The JML object types also support individual types of JML information blocks. For example, a menu consists of one title block and a range of menu items, while a title-only message only carries a title block.

Encoded JML objects of type menu will be referred to as "menu objects", while the different types of messages will be referred to as "message objects".

4.2.1 Menu object

A JML object of type menu ("menu object") consists of a title block followed by a list of link items. Link items consist of a visible label plus a reference to another JML object.

The user must be able to navigate through the list of link items and select one of them. If the user activates one of the provided links, the referenced JML object shall be presented (thereby replacing the currently visible JML object).

Content update behaviour:

- When an update of a menu object is received while this JML object is currently visible on screen, the new content should be rendered immediately. Thereby the user's original scrolling situation should be restored (thus the same link item index shall appear at the same screen location, provided that the link item is still present in the updated menu object).

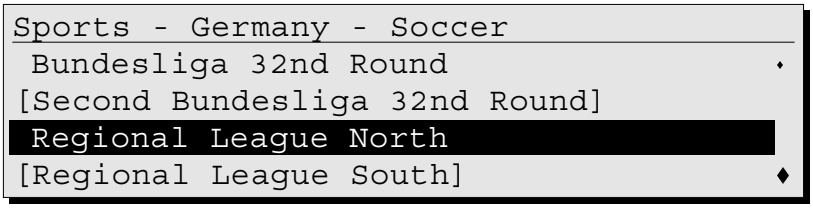


Figure 1: Menu object screen rendering example

Description of figure 1:

- The inverted line indicates the current user selection (cursor indicator).
  Lines with squared brackets indicate referenced JML objects that are not currently present in the local cache (and therefore the user would need to wait for the next successful reception of this JML object).
  Up/down arrows on the right indicate that more item links are available further up and down.

4.2.2 Message objects

4.2.2.1 Plain text message

Plain text messages consist of a title block followed by a block of body text. This type of JML object is intended for the presentation of longer messages.

When presented to the user, the title text and the following body text should be rendered on the screen with automatic line wrapping. The user must be able to access the full text (e.g. by manual scrolling). The title may be presented attached to the text (and therefore eventually scroll out of view along with the body text).

Content update behaviour:

- If an update to the currently presented JML object is received, the new content should not be displayed automatically. Instead, the user might be informed by an indication announcing that new content is available.

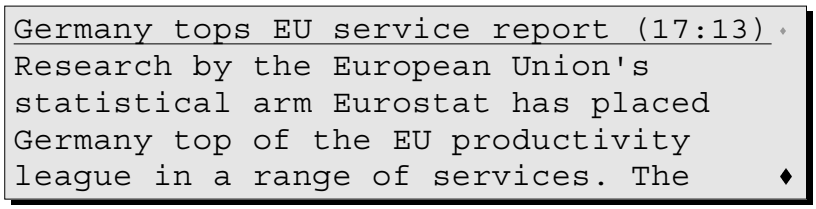


Figure 2: Plain text message screen rendering example

Description of figure 2:

- The light up arrow character on the top right indicates that the head of the message text (including its title) is shown. If the user scrolled, the title would scroll out of view along with the body text.

4.2.2.2 Title-Only message

Title-Only messages consist of one title block. The intended functionality is that of an automatically updated news ticker.

The title text should be rendered on the screen with automatic line wrapping. Whenever possible, the full text should be visible on screen at once. The user must be able to access the full text in case it exceeds the screen space (e.g. by manual scrolling).

If an update to the currently presented JML object is received, the new content should be displayed automatically and instantly, i.e. without explicit user interaction.

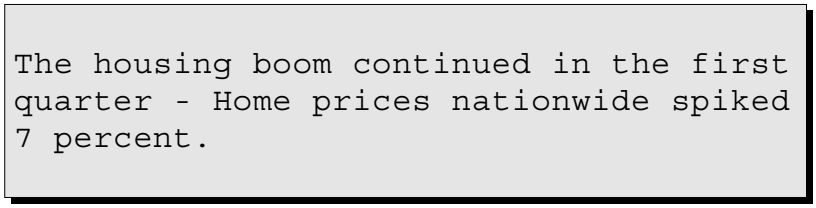


Figure 3: Title-Only message screen rendering example

Description of figure 3:

- The title text might for example be displayed in a centred mode, both horizontally and vertically.

4.2.2.3 List message

List messages consist of a title block and a list of content lines. Their purpose is to present information in list form (like sports results, stock market information, etc.) in an ordered fashion.

The title block may remain on screen even if the user scrolls through the lines of information in the body section. Columns of information can optionally be distinguished to provide a simple table functionality. Columns should be displayed in a way so that the horizontal start of columns is consistent among all currently visible lines. A mono-spaced font may be used for rendering this type of JML object. If the title or any text line exceeds the length of one line on the screen, its content might be horizontally scrolled, truncated or wrapped into multiple lines.

Content update behaviour:

- When an update of a list message is received while this JML object is currently visible on screen, the new content should be rendered immediately. Thereby the user's original scrolling situation should be restored (thus the same content line index should appear at the same screen location, provided that the updated List message carries at least this number of content lines).

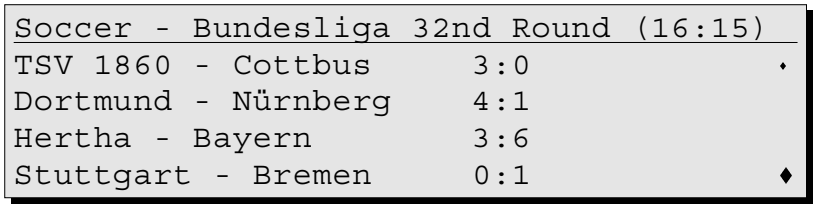


Figure 4: List message screen rendering example

Description of figure 4:

- The arrow characters on the right side of the screen indicate that more content is available above and below the current screen. The title text fits in one line and is kept on screen even if the user scrolls through content lines. A content update is automatically presented on screen, while the user's scroll position is maintained. So in the example it may seem as if just the result numbers were updated.

4.3 JML object hierarchy

The logical structure of a Journaline service is presented to the user as a tree of menus, sub-menus and messages.

All JML objects that form a Journaline service can be distinguished by their unique object ID. A Journaline service must at least comprise one JML object with the object ID value 0x0000, the "root object".

When the user initially accesses a Journaline service and no point of entry is explicitly defined by other means, the root object is presented. Typically this root object will be a menu object (the "main menu"), but it could also be any other type of JML object.

Every JML object must be referenced from at least one other JML object, with the exception of the root object and JML objects that serve as alternative points of entry to the service (accessed by external means).

Every JML object may be referenced from multiple JML objects of different hierarchy levels (for example the main menu might be referenced from every menu object in the service).

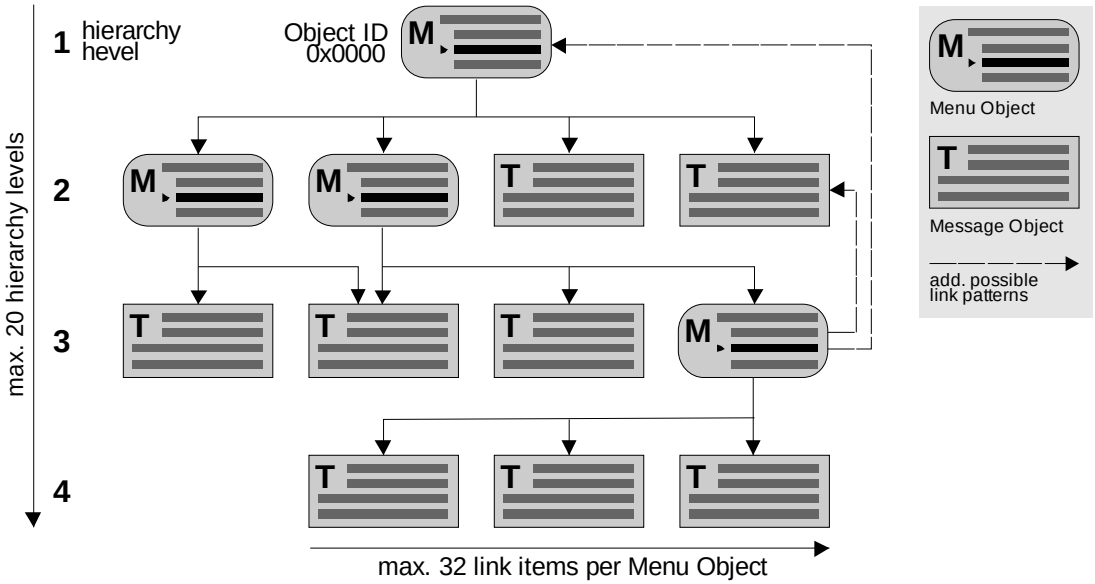


Figure 5: JML Object hierarchy example

The list of traversed object IDs when navigating from the root object to any JML object in the Journaline service (including the root object and the targeted JML object themselves) is called the "history path".

The "hierarchy level" of a JML object is defined as the shortest possible path from the root object to the JML object. The Journaline service is restricted to a **maximum of 20 hierarchy levels**, i.e. the shortest possible history path from the root object to any JML object in the Journaline service must not exceed 20 object IDs.

A receiver must be able to keep track of a path history of at least 20 object IDs to provide a "return to previous menu" navigation function to the user. However, since the selection of link targets in a menu object is not restricted and can point to any JML object in the Journaline service, the user may be able to reach a JML object after traversing more than 20 JML object with a unique object ID each. In this case the receiver may remove the oldest entries in its path history if required. If a user requests to return to a previous menu while no previous menu entry is available in the path history (i.e. the path history contains the current object ID as the first and only item), the root object shall be presented.

The previously described mechanism does not apply if a JML object references JML objects from its own history path (i.e. one of its parent menus). A typical example for this situation is a link to the main menu from every menu object. If the user follows this link to the higher hierarchy level JML object, the receiver is supposed to shorten its history path to the target object ID.

5 JML object - format specification

5.1 Overview

Every JML object is self-descriptive. No external information (like filename or type specification) is required to reference and access a JML object and to present it to the user.

An JML object consists of the "header section" followed by the "content section".

The size of a JML **object is limited to 4 092 bytes**. This size comprises the object header and the uncompressed content section.

The content section may be compressed to improve transmission efficiency. However, a JML object **shall not be compressed** if the total size of the compressed version of the JML object exceeds the size of the uncompressed version.

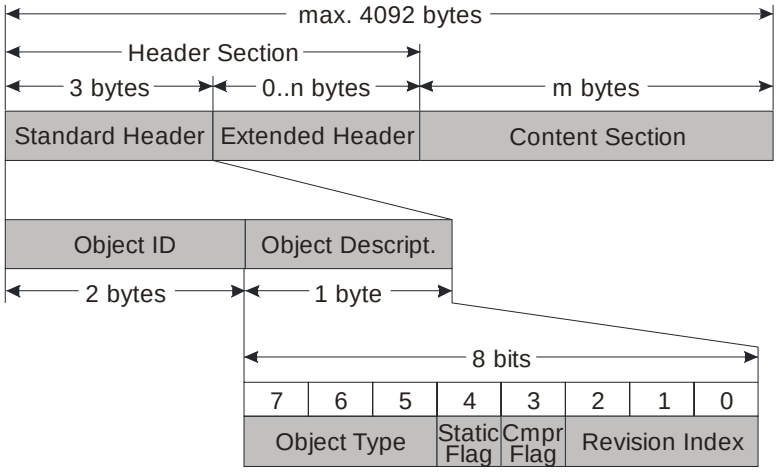


Figure 6: Components of a JML object

5.2 Header section

The header section of a JML object consists of a mandatory "standard header" and an optional "extended header".

The standard header comprises the object ID and the object description.

The presence of an extended header is indicated as part of the service signalling (see clause8.1.2). If present, it has an equal size for all JML objects of the Journaline service. A receiver shall discard the extended header if it does not know how to interpret the content.

The standard header contains the following parameters:

- Object ID 16 bits.
- Object description: 8 bits.

Where the object description consists of:

- Object Type 3 bits.
- Static flag 1 bit.
- Compress flag 1 bit.
- Revision index 3 bits.

The following definitions apply:

- **Object ID:** this 16-bit value uniquely identifies on JML object within a Journaline service:
  - 0x0000: assigned to the JML object serving as the default point of entry; the only object ID that must always be present in a Journaline service.
  - 0x0001 to 0xEFFF: freely assigned to JML objects by the broadcaster/encoder.
  - 0xF000 to 0xFFFF: reserved for future definition of specific content types.
- **Object type:** this 3-bit value indicates the type of JML object:
  - 000: reserved.
  - 001: Menu object.
  - 010: Plain Text message.
  - 011: Title-Only message.

- 100: List message.
- 101: reserved.
- 110: reserved.
- 111: reserved.

The receiver shall ignore JML objects with unknown object Type.

Static flag: this 1-bit value indicates whether the content provided with this object ID is static and therefore the type of content will remain the same in future. If a JML object is signalled as static, the user may be allowed to set a favourite (bookmark link) to this object ID for immediate access in future:

0: random type of content, user must not set a Favourite (bookmark link) to this temporary object ID.

1: type of content will remain static in future, user may set a Favourite (bookmark link) to this static object ID.

Compress flag: this 1-bit value indicates whether the content section is compressed:

0: Content section is uncompressed.

1: Content section is compressed.

If the content section is compressed, an extra 1-byte value is inserted between the header section and the content section indicating the compression method.

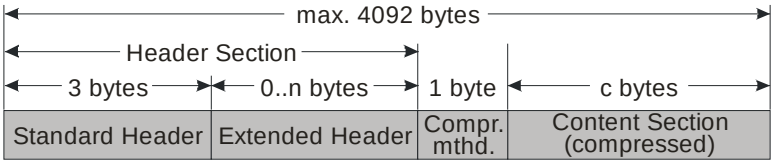


Figure 7: JML object with compressed content section

The **compression method** value can take the following values:

0x08: deflate compression method as specified in [6] with a fixed window size of 4 096 bytes.

All other values are reserved.

Revision index: this 3-bit unsigned integer value is incremented by 1 (modulo 8) whenever the JML object's content changes.

A receiver can easily determine whether a newly received JML object was changed compared to its cached version, for example by comparing the full header section and the JML object's total size. If both indicators are identical to the currently cached JML object, the receiver may assume that the JML object's content has not changed and discard the received JML object update.

If the receiver detects an update to a JML object's content, it shall process the new JML object and update its cache. If the JML object is currently presented to the user, the rules stated in clause 4.2 for the object type specific update behaviour should be applied.

5.3 Content section

The content section of a JML object contains text information to be presented on screen, and may contain escape sequences to implement extended functionality.

The text information carried in the content section is formatted according to [7] in UTF-8 encoding.

The content section of every JML object is structured by JML codes (JML: "Journaline Markup Language"), which indicate the start and end of a certain block of information (e.g. title or body text).

The content section may carry escape sequences to introduce text formatting information (e.g. highlighted text, line breaks, etc.) as well as additional functionality, for example by inserting areas of information which may simply be ignored by receivers not capable of dealing with the extended functionality.

Both JML codes and escape sequence codes are 1-byte codes from the range of 0x00 to 0x1F (UTF-8 "control character codes"):

- JML codes comprise the range from 0x00 to 0x0F.
- Escape Sequence codes range from 0x10 to 0x1F.

Depending on the value of the escape sequence code, additional bytes may follow to form the full escape sequence.

5.3.1 JML codes

An JML object's content section is divided into logical blocks of information (e.g. title, body text) by JML codes.

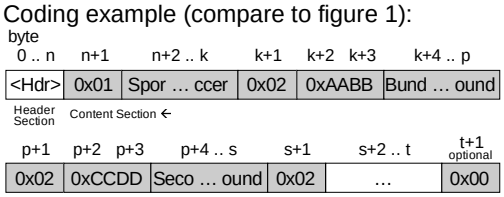
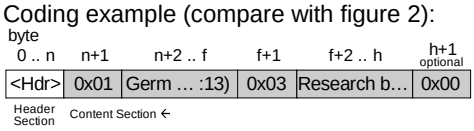
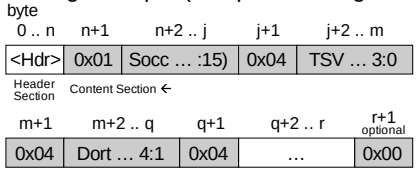
Every block is started with a type specific JML code and ended by:

- the JML code "end",
- the occurrence of another JML code (starting a new block and thereby implicitly ending the previous), or
- the end of the content section.

If a receiver does not know how to interpret a specific JML code it must ignore this code, but may still present the block's content.

The following JML codes are currently defined. All other JML code values are reserved for future definition.

Table 1: JML code definitions

JML code	Name	Description	Applicable object type
0x00	End	Ends the previously started block of information.	Menu, Plain Text, Title-Only, List
0x01	Title	Starts the title block. Exactly one title block must be present per JML object for the applicable object Types and it must be the first block within the content section. The title block must contain at least one character to be displayed. If the title exceeds the length of one line on the screen, the content may be truncated. Any available font may be used for rendering this type of information.	Menu, Plain Text, Title-Only, List
0x02	Link Item	Starts a link item within a menu object. Carries the target address and the text to be displayed per menu item in the following format: • 2 bytes object ID of the target JML object (i.e. the JML object that shall be displayed if the user selects this menu item). • n bytes target description (presented to the user). Coding example (compare to figure 1):  A **minimum of 1 and a maximum of 32 Link Items** must be contained in a menu object. Each target description must contain at least one visible character. If a target description exceeds the length of one line on the screen, the text may be truncated. Any available font may be used for rendering this type of information.	Menu
0x03	Body Text	Starts the body text block within a plain text message. Contains the text to be displayed as the plain text message's body. Coding example (compare with figure 2):  **Exactly one body text block** is contained in a plain text message. This block must contain at least one visible character. The user must be able to access the full text, e.g. by manual scrolling. Any available font may be used for rendering this type of information.	Plain Text
0x04	List Item	Starts a list item within a list message. Each list Item block contains one line of text to be displayed. Coding example (compare with figure 3):  **At least 1 list Item** must be contained in a list message. Each list Item must contain at least one visible character. A mono-space font should be used for rendering this type of information.	List

JML code	Name	Description	Applicable object type
0x05	List column	Indicates the start of the next column within a list Item to support a simple table structure of list messages. Columns should be displayed starting at the same horizontal position across all list Items currently visible on the screen. Each column's content must be visually separated from the content of other columns (e.g. by a space character width). If the list column code cannot be rendered as defined by any Journaline receiver, it shall at least be replaced by a space character. **0 or more list column codes** can be carried within each list Item. If a list Item carries less list column codes than other list Items within the same list message, the content for the remaining list columns shall be considered empty. To indicate empty intermediate columns within a list Item, multiple list column codes may immediately follow each other. At least one list Item must carry visible textual content for each of the list columns of the list message.	List

5.3.2 Escape Sequences

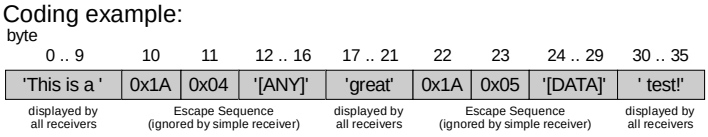
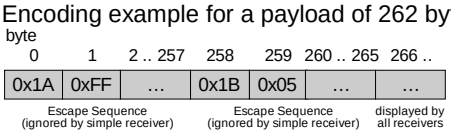
An JML object's content section may carry any number of escape sequences to introduce text formatting information and rendering hints (e.g. highlighted text, preferred line breaks, etc.) as well as additional functionality.

A receiver is free to evaluate the content and meaning of escape sequences. If a receiver is not able to interpret the content or meaning of a certain escape sequence, it shall ignore the full escape sequence.

Every escape sequence consists of at least the 1-byte escape sequence code, which describes the type of escape sequence. Some escape sequences consist of multiple bytes. Therefore a receiver must be able to at least determine the length of every escape sequence.

The following escape sequence codes are currently defined. All other escape sequence code values are reserved for future definition.

Table 2: Escape sequence code definitions

ESC code	Name	Description
0x10	Preferred line break	If applicable, the receiver shall render the following text starting in a new line. If not applicable for a certain type of rendering device, a space character shall be inserted instead.
0x11	Preferred word break	If a word does not fully fit in the remaining space of a screen line, a receiver may introduce a hyphen character and a line break at this position. Otherwise this code is ignored (without being replaced by another character). It is recommended to insert "preferred word break" escape sequence codes in words with more than 15 characters.
0x12	Highlight start	If applicable, the receiver should display the text between the "highlight start" and the '"highlight stop"' code in a highlighted form. If not applicable for a certain type of rendering device, these codes shall be ignored.
0x13	Highlight stop	
0x14	End of introductory section	This escape sequence code divides an introductory block of text from a more detailed level of information. This may for example be used by a receiver to only present the remaining detailed information upon special user request.
0x1A	Data section start	These escape sequence codes specify a data section within the visual text that shall not be rendered by a Journaline receiver that is not capable of evaluating the data section. A data section may for instance be inserted to provide advanced signalling or any amount of binary data. Each of these two codes is followed by 1 byte specifying the length of the following data section (up to 256 bytes) as an unsigned integer with the value (<number of bytes> - 1). Example text with inserted non-printable information: "This is a [ANY]great[DATA] test!" Expected visual representation: `This is a great test!` Coding example:  If a data section's payload comprises more than 256 bytes, additional data section continuation escape sequences are inserted immediately after the data section start escape sequence. An additional data section continuation code can only be inserted if the previous data section block is completely filled with 256 bytes. A data section continuation code cannot be inserted anywhere else. Encoding example for a payload of 262 bytes:  The internal payload data structure is defined in clause 5.3.2.2. Receivers that cannot evaluate a data section's content must ignore the whole escape sequence.
0x1B	Data section continuation	
0x1C	Extended code begin	These 2-byte escape sequences allow the signalling of up to 256 additional Escape codes. The full escape sequence consists of the escape sequence code "extended code begin"or "extended code end" followed by a one-byte extended code. These extended codes are specified in clause 5.3.2.1. If a receiver cannot interpret an extended code, it may safely ignore the 2-byte escape sequence.
0x1D	Extended code end	

5.3.2.1 Escape sequence 'extended codes'

Extended codes allow the extension of the range of available escape sequence codes by another 256 values.

If an 'extended code' exists in a "begin" and an "end" version (e.g. to signal the start and end of a special formatting rule), then the same extended code value is used: To start the sequence it is preceded by the "extended code begin" escape sequence code, and to end it by the "extended code end" escape sequence code. If an extended code consists only of a single code without an explicit begin/end version, it shall always be preceded by the "extended code begin" escape sequence code.

The following extended code values are currently defined.

Table 3: Extended code definitions

Extended code	Name	Description	including "end" version
0x00 to 0xFF		reserved for future definition	yes

5.3.2.2 Escape sequence "data section" structure and content

Data sections are special multi-byte escape sequences that allow the insertion of extended signalling or binary data to an object's content section, while ensuring full backwards compatibility to any Journaline receiver that cannot interpret the Data section's content. Data sections shall completely be discarded by a receiver if it cannot interpret their content.

This clause describes the general and type specific structure of information that can be carried in a Data section.

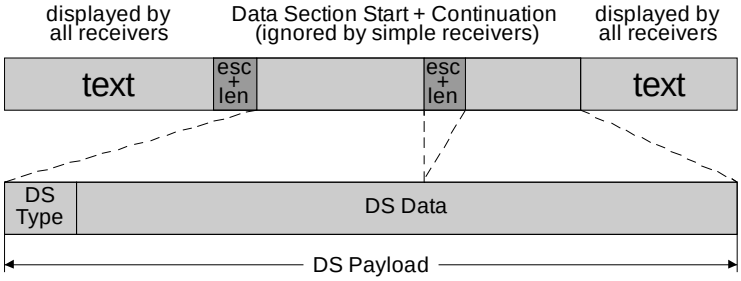


Figure 8: Data sections and payload structure

The "DS payload" is the combined content of the "data section start" and all immediately following "data section continuation" escape sequences. It has the following structure:

- DS type 8 bits.
- DS data $n \times 8$ bits.

The following definitions apply:

DS type: this 8-bit field identifies the type of content carried in the DS data section.

DS data: this *n*-Byte field carries the useful information. Its structure and meaning is defined by the DS type parameter.

The following DS types are currently defined. All other DS type values are reserved for future definition.

Table 4: DS type definitions

DS type	Name	DS data description	Location in content section
JML object management types:			
0x00	Padding	Carries padding bytes. The content must be ignored.	anywhere
0x01	Absolute timeout	Defines the absolute presentation timeout of a JML object. If this timeout has passed, the JML object shall no longer be presented to the user. This escape sequence code is intended for marking individual JML objects. A general timeout for the Journaline service should be provided by the TOC management data (see clause 6.2). If both types of timeouts are provided, the JML object specific value applies. The DS data section contains the number of multiples of 15 minutes since 2000-01-01 00:00 UTC as a 24-bit unsigned integer number (covering more than 450 years).	maximum one before first visual text character
0x02	Relative timeout	Defines the relative presentation timeout of a JML object. If this timeout has passed, the JML object shall no longer be presented to the user. The timeout re-starts with every reception of a JML object (even if already stored in cache). This escape sequence code is intended for marking individual JML objects. A general timeout for the Journaline service should be provided by the TOC management data (see clause 6.2). If both types of timeouts are provided, the JML object specific value applies. The DS data section contains the number of minutes as a 16-bit unsigned integer number (covering more than 45 days).	maximum one before first visual text character
0x03	General link target	Defines a general link target, i.e. a target that may be presented/activated if the user explicitly request to proceed ("hot button" for user interaction). A detailed description is given in clause 5.3.2.3. More than one link target may be provided and supported by a receiver. The receiver shall be able to handle this case (e.g. by asking the user).	any number before first visual text character
0xFF	Proprietary	Any proprietary content to be ignored by receivers.	anywhere
Content management types:			
0x20	Keyword	Marks a following (visual) keyword along with an optional keyword description. The keyword may for example be used to create a receiver-based search index. The DS data section has the following format: • 1 byte keyword length, value is (<length>-1) as unsigned integer number. Identifies the number of visual text characters following this data section that shall be treated as the keyword. • n bytes description. The optional description may be indexed in addition to the keyword and/or presented to the user.	anywhere
0x21	Macro definition	A macro allows the definition of text sections (including optional escape sequences), that can be inserted multiple times anywhere in the content section with a simple reference. For example, the macro may define speech descriptions that shall be presented surrounding textual list elements. The DS data section has the following format: • 1 byte macro ID (0 to 255). Identifies the following Macro definition. • n bytes macro definition. Contains the text (including escape sequences) that shall be inserted whenever this macro is referenced by its ID. Note that macros should not be used for essential information, since they may be ignored by receivers.	maximum one per Macro ID value, before first visual text character
0x22	Macro reference	Virtually inserts the macro definition referenced by its ID at the location of this escape sequence for presentation to the user. The total JML object length may exceed 4 092 bytes after all macro references have been expanded! The DS data section carries the 1-byte Macro ID (0..255) referencing a macro definition.	anywhere

DS type	Name	DS data description	Location in content section
Speech support types:			
0xA0	Default language	Describes the default language of the JML object. The DS data section carries the 3-letter ISO language code (see [5]) as lowercase characters.	maximum one before first visual text character
0xA1	Language section	Defines the language of a text section. The DS data section has the following format: • 1 byte text length, value is (<length>-1) as unsigned integer number. Identifies the number of visual text characters following this Data section to which the language definition applies. • 3 bytes language. Carries the 3-letter ISO language code (see [5]) as lowercase characters.	anywhere
0xA2	Speech phoneme	Defines the phoneme description of a text section using the International Phonetic Alphabet (IPA). The DS data section has the following format: • 1 byte text length, value is (<length>-1) as unsigned integer number. Identifies the number of visual text characters following this Data section which are represented by the phonetic definition. • n bytes IPA phonemes. Contains the phoneme definition as IPA notation (see [4]).	anywhere
0xA3	Speech break	Defines a break for text to speech engines that shall be inserted at the location of the escape sequence. The DS data section carries 1 byte as an unsigned integer value defining the break duration in units of 0,1 seconds.	anywhere
0xA4	Speech characters	Defines a number of visual text characters following the escape sequence that shall be treated by a text to speech engine as individual characters or digits instead of contiguous words or numbers. The DS data section carries 1 byte defining the number of visual text characters as unsigned integer number with the value (<length>-1).	anywhere
Location reference types:			
0xB0	Latitude/longitude positions	Defines a range of positions as latitude/longitude value pairs. Each value pair describes one position reference of the JML object. At least one position reference must be defined. A latitude value is specified as a 24-bit 2's complement integer number, in 1/92000th degrees between -90,0 degrees (south pole) and +90,0 degrees (north pole) (leading to a granularity of ca. 2,4m). A longitude value is specified as a 24-bit 2's complement integer number, in 1/46000th degrees between -180,0 degrees (west) and +180,0 degrees (east) (leading to a minimum granularity of ca. 2,4m). A position's longitude value immediately follows the position's latitude value. Additional position references immediately follow the previous position references.	maximum one before first visual text character
0xB1	Latitude/longitude regions	Defines a region reference for the JML object. The region definition is described as a polygon built from at least 3 latitude/longitude value pairs. Each value pair shall be formatted as described for DS Type "0xB0: Latitude/longitude Positions". If the absolute longitude value difference of two successive position references is larger than 180 degrees, it shall be assumed that the ±180 degree meridian was crossed.	any number before first visual text character

5.3.2.3 Data section "general link target" structure and content

The data section's DS type value 0x03 defines a general link target, i.e. a target that may be presented/activated if the user explicitly request to proceed ("hot button" for user interaction). General link targets may be defined for all types of JML objects. The availability of any general link target for the currently presented JML object should be indicated to the user, if its link Ttype is supported by the receiver.

The DS data section "general link target" has the following format:

- Link type 8 bits.
- Link address $n \times 8$ bits.
- Address end marker 0 or 8 bits.
- Link label $m \times 8$ bits.

The following definitions apply:

Link type: this 8-bit value defines the type of the general link target and thus the format of the remaining DS data section format. The following link type values are currently defined.

Table 5: Link type definitions

Link type value	Description	Link address format
0x00	JML object	two-byte object ID of the target JML object (may contain bytes of value 0x00!)
0x01	DAB-/DRM-URI	URI string pointing to other DAB/DRM multiplexes, services or service elements
0x02	URL	Internet URL string pointing to Internet addresses, web documents, etc. See note
0x03	Telephone	telephone number in human-readable form, pointing to a voice service that can be reached via phone; the number shall start with the international prefix "+<international area code>"
0x04	SMS	mobile phone short message service, consisting of the recipient's telephone number plus the SMS content text, both divided by one '+' character; telephone number in human-readable form, pointing to an SMS recipient; the number shall start with the international prefix "+<international area code>"; the SMS content text section may be empty or carry text in UTF-8 encoding excluding any byte value 0x00
NOTE: The receiver should evaluate the protocol section of the URL to distinguish URL types. Typical protocols are "http:", "https:", "ftp:" or "mailto:".		

All other values are reserved for future definition.

General link targets with unknown or unsupported link type value must be ignored by a receiver.

Link address: this *n*-byte field carries the address of the general link target. If not defined otherwise, all textual address values shall use UTF-8 character encoding. *n* must not be 0.

Address end marker: this 1-byte field indicates the end of the link address field. It is fixed to the value 0x00. This field is not present if the link label is omitted ($m = 0$).

Link label: this *m*-byte text carries a short textual description of the provided link (e.g. "Call the reservation hotline"). It is formatted like any visible text in the JML object and may contain escape sequences. *m* may be 0, thereby omitting the link label as well as the address end marker fields.

6 Management data

Management data can be broadcast in parallel to the JML objects described in the previous clause. Management data provides additional information to the receiver that for example supports advanced management of JML objects on the receiver side.

Note that JML objects (see clauses 4 and 5) are fully self-describing and their evaluation is sufficient for a receiver to present the Journaline service to the user.

The provision of management data is optional for the broadcaster/encoder, as is the evaluation of management data for the receiver. However, providing and evaluating management data may simplify the implementation of Journaline for more advanced receivers.

6.1 General rules

Management data is sent in blocks of information.

For a management data block the same restrictions as for JML objects apply:

- Every block of management data is self-contained and self-describing.
- Every block of management data is limited to 4 092 bytes.
- Every block of management data may be compressed, but shall not be compressed if the compressed size exceeds the size of the uncompressed data. The use of compression may be forbidden for individual types of management data.

Management data blocks are distinguished from JML objects on transport layer (see clause 8).

The first byte of a management data block always carries the 'management data block type' indicator to distinguish between different types of management data blocks.

6.2 TOC - "Table of Contents"

6.2.1 Overview

Blocks of management data carrying TOC ("Table Of Contents") information are called "TOC blocks". They are intended to support the receiver's caching strategy by listing all JML objects currently broadcast in the Journaline service along with their revision. Each description of a JML object is called "TOC entity".

A receiver can easily and immediately remove outdated JML objects from its internal cache as soon as an updated TOC block was received.

The full TOC may consist of multiple blocks of data. In this case all blocks belonging to one TOC are called "TOC fragments" and share the same TOC revision number. However, a receiver can process every TOC block independently from all other TOC blocks. It is never required to assemble the full TOC.

The management data compression feature shall not be used for TOC blocks.

6.2.2 TOC block layout

A TOC block consists of:

- the mandatory TOC standard header; followed by
- the optional TOC extended header; followed by
- the mandatory list of TOC entities (each TOC entity describing one JML object).

Every TOC block has the following layout.

Parameters of the TOC standard header:

- Management data block type 8 bits.
- Revision counter 8 bits.
- Block count 8 bits.
- Block index 8 bits
- Preceding object ID 16 bits.
- Object count 16 bits.
- Timeout 16 bits.

- TOC entity length 8 bits.
- TOC extended header length 16 bits.

Parameters of the TOC extended header (if present):

- *currently not defined.*

Parameters of the list of TOC entities

- j × TOC entity k × 8 bits.

Where each TOC entity comprises the following parameters:

- Object ID 16 bits.
- Object description 8 bits.
- TOC entity extension n × 8 bits.

The following definitions apply for the TOC standard header:

Management data block type: this 8-bit parameter is mandatory in every block of management data to identify its type. For TOC blocks, the value is set to 0x54 (ASCII character "T").

Revision counter: this 8-bit value is treated as an unsigned integer number and increased by 1 (modulo 256) whenever the content of any of the TOC fragments of the current TOC changes.

Block count: this 8-bit value indicates the total number of TOC fragments that are used to describe the full TOC as an unsigned integer number (1 to 255).

Block index: this 8-bit value indicates the index of the current TOC fragment within the list of TOC fragments describing the full TOC as an unsigned integer number in the range 0 to (block count - 1).

Preceding object ID: this 16-bit value indicates the object ID value of the last TOC entity carried in the previous TOC fragment. It shall be set to 0x0000 if the block index of the current TOC block is 0.

Object count: this 16-bit unsigned integer value describes the total number of JML objects that currently form the Journaline service.

Timeout: this 16-bit parameter defines an amount of minutes as an unsigned integer number (covering more than 45 days). The value indicates how many minutes after reception of this TOC block the data set building the Journaline service can still be considered valid. Outdated information shall not be presented to the user. A timeout can occur for example in case of permanent reception loss. Every new reception of a TOC block extends the timeout by the indicated number of minutes. A value of 0 disables the timeout feature, content will never expire.

TOC entity length: this 8-bit value indicates the length of each TOC entity carried in the current TOC block. Its value is currently set to 3, but may increase in a later definition.

TOC extended header length: this 16-bit value indicates the size of the TOC extended header in the current TOC block as an unsigned integer number. The value is currently set to 0, but may increase in a later definition.

The following definitions apply for the list of TOC entities:

- This mandatory section of a TOC block carries a contiguous list of TOC entities. Every TOC block must carry at least one TOC entity description ($j \geq 1$). The TOC entities shall be sorted in ascending order of the object ID parameter value.
  The length of each TOC entity is indicated by the TOC standard header's parameter "TOC entity length" ($k \geq 3$).

The following definitions apply for each TOC entity:

Object ID: this 16-bit value carries the object ID of the JML object described by the TOC entity.

Object description: this 8-bit value carries the parameters of the object description of the JML object described by the TOC entity.

TOC entity extension: this data block consists of *n* additional bytes extending the TOC entity description. The number *n* is current fixed to 0 and may be increased along with a definition of the carried parameters in future. A receiver shall discard these bytes of a TOC entity if it cannot interpret them.

6.2.3 TOC block transmission

If more than one TOC block is required to carry the full list of TOC entities, the TOC content may be split up in multiple TOC fragments. In this case multiple TOC blocks will be sent in carousel mode.

In case of TOC fragmentation the following rules shall apply:

- Each TOC block may be sent at any time in any order at any repetition rate.
- The TOC standard header parameter revision counter is incremented by 1 (modulo 256) whenever the content of any TOC fragment is changed.
- Each TOC block is formatted according to the TOC block layout (see clause 6.2.2).
- The following parameter values of the TOC standard header must vary between TOC fragments:
  Block index, Previous object ID.
- The following parameter values of the TOC standard header may vary between TOC fragments:
  TOC entity length, TOC extended header length
- The content of the TOC extended header of each TOC fragment may be different.
- Every JML object shall be described only once per list of TOC entities and only in one TOC fragment.
- Within one TOC block all TOC entities must be ordered in ascending order of the object ID value. The object ID of the first TOC entity of a TOC block must be larger than the object ID value of the last TOC entity of the previous TOC block (i.e. of the standard header's Preceding object ID parameter value).
- The TOC standard header parameter previous object ID shall be set to the object ID value of the last TOC entity of the previous TOC block. If the current TOC standard header parameter block index is 0, then the previous object ID value shall be set to 0x0000.
  The previous object ID parameter indicates that all object ID values between this value and the first TOC entity in the current TOC block are currently not used in the Journaline service and therefore may be removed from the receiver's cache.
  On receiver side this parameter permits to handle each TOC block independently from any other TOC block - the assembly or storage of the complete list of TOC blocks is not required for the receiver's cache management. The receiver may implement a mechanism to evaluate each TOC fragment (identified by the TOC standard header parameter block index) only once per value of the revision counter.

7 Expected receiver behaviour

Some general rules apply to Journaline compliant receivers.

7.1 Mandatory functionality

Support for the following functionality is a minimum requirement for a Journaline compliant receiver:

- The standard header of a JML object must fully be interpreted. If the extended header is present and its content cannot be interpreted, the extended header shall be discarded.
- Deflate decompression for the content section must be supported.
- If the user activates Journaline and no point of entry is defined by other means (e.g. as part of the user application signalling), the first JML object to be displayed is the root object with object ID 0x0000.
- The object types menu, plain text message, title-only message and list message must be supported. JML objects with unknown object types must be ignored.

- The user can navigate through menus and select any of the provided link items. The receiver replaces the currently presented JML object with the one defined by the link item.
- The user can navigate back to previously visited menus.
- The user can access the full visual text of any object, e.g. by scrolling. This does not apply to characters that cannot be displayed due to a limited character font.
- Unknown escape sequences shall be discarded if their content cannot be interpreted.
- If caching of JML objects is supported by a receiver, the timeout rules per JML object and for the whole Journaline service (as defined in the TOC management data) shall apply. If both a relative and an absolute timeout value are defined, the absolute timeout has highest priority and must be obeyed if the receiver can support absolute timeouts.

7.2 Recommended functionality

A Journaline compliant receiver should support as much of the Journaline functionality as possible according to its screen and user interface capabilities.

A Journaline compliant receiver does not need to evaluate escape sequences or management data, with the exceptions stated in clause 7.1.

The following behavioural descriptions clarify some functional aspects of Journaline and should be supported by receivers if possible.

7.2.1 User history

The receiver should keep a record of all object IDs traversed by the user from the root object (object ID 0x0000) or an alternative point of entry object ID (on initial access of the Journaline service) to the currently presented JML object (the history path). In this way it can easily provide a "back to previous menu" functionality to the user (see clause 4.3).

While the JML object with object ID 0x0000 is the default point of entry to the service (the default root object), other points of entry may be defined by external means, such as the user application signalling, external DAB-/DRM-URLs, or the stored location where the user previously left the Journaline service.

Generally, if the user requests to return to a higher-level menu while the history path is empty, or if a previously visited higher-level menu object does no longer exist in the Journaline service broadcast carousel, the receiver should return to the root object.

7.2.2 Content update behaviour

An update to a cached version of a JML object (with identical object ID) can for example be identified by a different revision index or object size. If a JML object with a certain object ID is currently presented to the user when an updated version is received, it should be signalled to the user that an updated version of this JML object was received (e.g. by placing a flashing "Update received - select OK to show it" icon on the screen).

In case of "menu objects" and "list messages" it is recommended to immediately update the screen with the new object's content while maintaining the user's current relative position within the list at the same position on screen. This position could be identified by the list item index, the target object ID (in case of menu objects), or other means.

In case of "title-only messages" it is recommended to immediately update the screen with the new object's content.

7.2.3 Reception status indication

If the user selects a link item of a menu object which points to a JML object that is not yet available in the receiver's cache, the user may need to wait for the target JML object's next reception. This may take some time and should be indicated to the user (e.g. by presenting a "receiving - please wait" text or icon).

To prevent the user from selecting link items from a menu that may take some time to display, these link items should be clearly marked in the menu object's rendering. This could for example be achieved by a greyed presentation of the link description, or by surrounding the link description with brackets.

If the receiver supports caching and some of the previously marked link items become available in the receiver's cache, the currently presented menu object should be updated immediately.

7.2.4 Favourites functionality

The "favourites" functionality greatly enhances the Journaline functionality perceived by the user by enabling the user to store certain JML objects for quick and direct future access. This storage does not refer to the content of the JML object, but to its object ID, i.e. when recalling the favourite, then the current content of the JML object should be presented.

To enable the "favourites" feature on receiver side, the JML object must signal whether the type of content will be static in future for the object ID, or whether the object ID is chosen from a random set of IDs to transport temporarily valid content. This signalling is provided by the object header's "static flag". For example, a JML object carrying "BBC financial headlines" may always be provided by a specific broadcaster as object ID "0x1234". In this case the static flag should be enabled by the broadcaster/encoder, and the receiver should allow the user to store this page as a "favourite". In future, the user can immediately return to this news ticker page by just recalling the corresponding "favourite" (e.g. by pressing a radio's station key).

The static flag may be enabled for all types of JML objects.

If the static flag is disabled, the receiver should not allow the user to store the JML object as a "favourite".

A receiver providing the "favourites" functionality should store the full history path of object IDs from the root object to the stored JML object along with the favourite's object ID. Otherwise the receiver will not be able to provide sophisticated 'return to higher menu level' functionality to the user the next time the "favourite" JML object has been recalled by the user. The JML object's "title text" may serve as the default "favourite" label.

7.2.5 Object caching strategy

A Journaline compliant receiver does not need any caching support to present a fully functional Journaline service to the user. Only the currently presented JML object must be decoded (and possibly decompressed) in memory before being presented on screen. Whenever the user requests to proceed to a different JML object (e.g. by selecting a link item in a menu object), the receiver can wait for the next reception of the requested JML object and then present it. This behaviour can be compared to a simple "video text mode" available in older TV sets.

However, if the receiver does provide a certain amount of cache memory, the user experience can greatly be enhanced. Ideally, the whole Journaline service may be stored in cache for immediate responsiveness to user commands. This could even be extended to persistent caching of previously received and decoded Journaline service to enable an immediate availability of Journaline information if the user switches between Journaline services.

If caching of JML objects is supported by a receiver, the timeout rules per JML object and for the whole Journaline service (as defined in the TOC management data) shall apply.

If a receiver has limited caching capacity and runs out of memory, it must start removing JML objects from its cache. This should be performed in a manner least disturbing to the user experience of the Journaline service:

- All JML objects no longer listed in the current TOC management data can immediately be removed from the cache.
- JML objects that should be kept in cache as long as possible are those immediately accessible to the user, like the parent menu object, the link targets contained in the current menu objects (if applicable), and the favourites.
- The second most important JML objects in cache are those easily accessible to the user, like the JML objects contained in the history path (including the root object) and the alternative link targets contains in the parent menu objects.
- Generally, title-only messages (ticker news) could be removed before plain text messages and list messages. menu objects should preferably be kept in cache, since they are required to navigate through the Journaline service and will typically be most static.

7.2.6 Support for text to speech engines

Journaline as a low-profile text based information service is an ideal basis for text-to-speech synthesis on sophisticated high-end receivers.

Text-to-speech guiding information is optionally available in a JML object in form of data section escape sequences (see clause 5.3.2.2).

- "Default language" defines the general language of a JML object and may be used to determine the correct text-to-speech preset.
- "Language section" defines the language for a specific amount of visual text characters. Therefore the text-to-speech engine may temporarily switch its language context to improve the speech quality.
- "Speech phoneme" defines the precise pronunciation rules for a specific amount of visual text characters in the general and vendor-independent IPA ("International phonetic alphabet") notation. This feature may dramatically increase the speech quality for example of geographic or people's names.
- "Speech break" allows the insertion of defined breaks into the speech output.
- "Speech characters" indicates that the text-to-speech engine should treat a specific amount of visual text characters as individual letters or digits instead of contiguous words or numbers.

In addition the "macro" escape sequences from the content management section may be helpful also in the context of text-to-speech playback. Using macro definitions, a certain amount of pure phoneme information could be defined once and easily be re-inserted multiple times within the content section.

For example, a list object carrying sports results could nicely be presented on screen by a plain list of team names and result numbers. However, reading these pure pieces of information might produce a quite irritating result. In this case each line carrying results information could be preceded and trailed by a phoneme definition that surrounds the provided information with useful introductory speech-only information to form full spoken sentences. The mechanism of macro definitions allows the space-efficient encoding of these repeating escape sequences per JML object.

Finally the following escape sequence codes (see clause 5.3.2) may serve to improve the text-to-speech experience.

- "Preferred line break" may serve as an implicit indicator of a paragraph end. If a language requires explicit indicators for word and/or sentence ends, the corresponding (probably non-visual) UTF-8 codes should be inserted.

- "End of introductory section" may serve as a default halt for the text-to-speech presentation of a lengthy message text, unless the user explicitly requested the full content to be presented.

8 Data application definition

8.1 Transport in DAB - Digital Audio Broadcasting

A Journaline service can be broadcast as a user application via DAB - Digital Audio Broadcasting (see [1]).

8.1.1 Data transport

A Journaline service component can be broadcast as PAD as part of the audio stream or in a packet mode subchannel. Both transmission modes can be freely mixed on the same multiplex. It is highly recommended for every receiver supporting the Journaline service component to be able to decode both transport modes. If the Journaline information is transmitted in packet mode, it can be signalled as a stand-alone service or as a secondary service component.

Every single Journaline object (JML object and every block of management data) is transported as one "MSC data group" (see [1] clause 5.3.3).

In general, an MSC data group contains the following items:

- MSC data group header (2 or more bytes).
- Session header (optional, 3+n bytes).
- MSC data group field (m bytes).
- MSC data group CRC (2 bytes).

The following restrictions apply to the current definition of Journaline. If any of the restrictions given below is not obeyed, the receiver shall discard the MSC data group.

The MSC data group header length is fixed to 2 bytes. It shall have the following layout:

- Extension flag = 0.
  Conditional access on MSC data group level is not supported.
- CRC flag = 1.
  The MSC data group CRC must be present.
- Segment flag = 0.
  Segmentation is not supported.
- User access flag = 0.
  User access information is not supported.
- Data group type:

  0000 MSC data group carries one JML object.

  0110 MSC data group carries one block of management data.

  All other values are reserved for future definition.

  MSC data groups with unknown type must be discarded by the receiver.
- Continuity index:
  Increments continuously for every MSC data group. The value may safely be ignored by a Journaline receiver.
- Repetition index:
  This parameter is typically set to the value 0000_b. The value may safely be ignored by a Journaline receiver.

- Extension field:
  This parameter field is not present.

- Session header:
  This parameter field is not present.

The MSC data group field is fixed to a maximum length of 4 092 bytes. MSC data groups with longer MSC data group fields shall be ignored by Journaline receivers.

The MSC data group CRC field is mandatory for Journaline. See [1] "annex E (normative): Calculation of the CRC word".

8.1.2 User application signalling

The following information shall be used to signal the DAB data application "Journaline":

The "user application type", signalled in figure 0/13 shall be set according to the table entry in [2]) for "Journaline".

The "user application data" field shall have the following structure:

- Version 8 bits.
- Length of extended header 8 bits.
- Point of entry object ID 0 or 16 bits.

Additional parameters may be defined in future following the "point of entry object ID" parameter. These parameters shall be ignored if the receiver cannot interpret them.

The following definitions apply:

Version: this 8-bit unsigned integer value indicates the version of the Journaline definition in use. Currently this value is set to 0x00.

All future versions of Journaline will be backward compatible such that a version-0x00 receiver still can decode all version-0x01, 0x02 etc. data, although it may not be able to decode or use the additional information. Additional information parameters may be added in a fully backward compatible way in many places of this Journaline specification, like at the end of the "user application data" parameter field, by activating the extended header of a JML object, by defining additional JML codes or escape sequence codes, by introducing additional JML object types, by defining more MSC data group types, etc.

NOTE: Revisions which are not backwards compatible to the present document will require a new "User Application Tupe" ID to be assigned.

Length of extended header: this 8-bit unsigned integer number defines the length of a JML object's extended header section. The value is currently set to 0x00. The length value applies equally to all JML object of the Journaline service. Every receiver shall discard a JML object's extended header section if its length is larger than 0 and the receiver cannot interpret its parameters.

Point of entry object ID: this 16-bit unsigned integer number, if present, carries the object ID of an alternative point of entry to the Journaline service. If the user accesses the Journaline service based on this user application signalling, and no alternative point of entry is defined by other means (e.g. the object where the user last left the service), the provided object ID shall serve as the default point of entry replacing the default root object. This is useful if multiple user application signalling's reference the same physical service component, each user application signalling providing an individual point of entry. If the value is omitted (i.e. the user application signalling only consists of the "version" and "length of extended header" parameters) the point of entry object ID value shall default to the root object ID 0x0000.

8.2 Transport in DRM - Digital Radio Mondiale

Journaline can be broadcast via DRM - Digital Radio Mondiale (see [3]), as a data service component.

The data application Journaline is signalled in SDC data entity 5 "application information data entity" (see [3], clause 6.4.3.6). It may, in addition, be signalled in the FAC "application identifier" (see [3] clause 6.3.4).

Every single Journaline object is transported in DRM packet mode as one DRM data unit carrying an "MSC data group". See clause 8.1.1.

The Journaline data application is signalled as belonging to the "DAB domain" using the service signalling (i.e. "user application type" ID and "user application data") as specified in clause 8.1.2.

History

Document history		
V1.1.1	June 2008	Publication