
**Information technology — Coding of
audio-visual objects —**

**Part 23:
Symbolic Music Representation**

*Technologies de l'information — Codage des objets audiovisuels —
Partie 23: Représentation symbolique de la musique*



Reference number
ISO/IEC 14496-23:2008(E)

AVA Documents
Document Sharing Hub
© ISO/IEC 2008

PDF disclaimer

This PDF file may contain embedded typefaces. In accordance with Adobe's licensing policy, this file may be printed or viewed but shall not be edited unless the typefaces which are embedded are licensed to and installed on the computer performing the editing. In downloading this file, parties accept therein the responsibility of not infringing Adobe's licensing policy. The ISO Central Secretariat accepts no liability in this area.

Adobe is a trademark of Adobe Systems Incorporated.

Details of the software products used to create this PDF file can be found in the General Info relative to the file; the PDF-creation parameters were optimized for printing. Every care has been taken to ensure that the file is suitable for use by ISO member bodies. In the unlikely event that a problem relating to it is found, please inform the Central Secretariat at the address given below.



COPYRIGHT PROTECTED DOCUMENT

© ISO/IEC 2008

All rights reserved. Unless otherwise specified, no part of this publication may be reproduced or utilized in any form or by any means, electronic or mechanical, including photocopying and microfilm, without permission in writing from either ISO at the address below or ISO's member body in the country of the requester.

ISO copyright office
Case postale 56 • CH-1211 Geneva 20
Tel. + 41 22 749 01 11
Fax + 41 22 749 09 47
E-mail copyright@iso.org
Web www.iso.org

Published in Switzerland

Contents

Page

Foreword.....	iv
1 Scope	1
2 Normative references	1
3 Conformance	2
4 Terms and Definitions	2
5 Conventions	4
5.1 Naming convention.....	4
5.2 Documentation convention	4
6 Symbols and abbreviations	5
7 SMR Bitstream	6
7.1 Introduction	6
7.2 SMR Bitstream Introduction	6
7.3 SMR Bitstream Description	8
7.4 Coding XML Segments.....	12
7.5 Decoding Process.....	14
8 Symbolic Music eXtensible Format (SM-XF): the Symbolic Music Representation	16
8.1 Symbolic Music eXtensible Format (SM-XF) introduction.....	16
8.2 SM-XF Definitions	16
8.3 Single Part (SMXF_Part).....	26
8.4 Main Score (SMXF_Main)	96
8.5 Lyrics (SMXF Lyric)	105
9 Symbolic Music Synchronization Information (SM-SI)	113
9.1 Symbolic Music Synchronization Information (SM-SI) Introduction	113
9.2 SM-SI Binary Format.....	113
10 Symbolic Music Formatting Language (SM-FL)	114
10.1 SMR Formatting Introduction	114
10.2 General architecture of the formatting engine	114
10.3 The SMR Rendering Rule Approach	116
10.4 Syntax of rules and conditions	118
10.5 SM-FL Examples	149
10.6 Rules and conditions for beams on multiple staves.....	154
11 Relationship of SMR with other parts of the standard.....	167
11.1 Introduction	167
11.2 SMR and MPEG-4 Systems	168
11.3 SMR and MIDI (through MPEG-4 Structured Audio)	171
11.4 SMR and MPEG fonts	171
12 SMR Object Types for Profiles	171
12.1 Introduction	171
12.2 Simple Object Type.....	172
12.3 Main Object Type	172
13 List of digital annexes	172
Bibliography	173

Foreword

ISO (the International Organization for Standardization) and IEC (the International Electrotechnical Commission) form the specialized system for worldwide standardization. National bodies that are members of ISO or IEC participate in the development of International Standards through technical committees established by the respective organization to deal with particular fields of technical activity. ISO and IEC technical committees collaborate in fields of mutual interest. Other international organizations, governmental and non-governmental, in liaison with ISO and IEC, also take part in the work. In the field of information technology, ISO and IEC have established a joint technical committee, ISO/IEC JTC 1.

International Standards are drafted in accordance with the rules given in the ISO/IEC Directives, Part 2.

The main task of the joint technical committee is to prepare International Standards. Draft International Standards adopted by the joint technical committee are circulated to national bodies for voting. Publication as an International Standard requires approval by at least 75 % of the national bodies casting a vote.

Attention is drawn to the possibility that some of the elements of this document may be the subject of patent rights. ISO and IEC shall not be held responsible for identifying any or all such patent rights.

ISO/IEC 14496-23 was prepared by Joint Technical Committee ISO/IEC JTC 1, *Information technology*, Subcommittee SC 29, *Coding of audio, picture, multimedia and hypermedia information*.

ISO/IEC 14496 consists of the following parts, under the general title *Information technology — Coding of audio-visual objects*:

- *Part 1: Systems*
- *Part 2: Visual*
- *Part 3: Audio*
- *Part 4: Conformance testing*
- *Part 5: Reference software*
- *Part 6: Delivery Multimedia Integration Framework (DMIF)*
- *Part 7: Optimized reference software for coding of audio-visual objects* [Technical Report]
- *Part 8: Carriage of ISO/IEC 14496 contents over IP networks*
- *Part 9: Reference hardware description* [Technical Report]
- *Part 10: Advanced Video Coding*
- *Part 11: Scene description and application engine*
- *Part 12: ISO base media file format*
- *Part 13: Intellectually Property Management and Protection (IPMP) extensions*
- *Part 14: MP4 file format*

- *Part 15: Advanced Video Coding (AVC) file format*
- *Part 16: Animation Framework eXtension (AFX)*
- *Part 17: Streaming text format*
- *Part 18: Font compression and streaming*
- *Part 19: Synthesized texture stream*
- *Part 20: Lightweight Application Scene Representation (LSeR) and Simple Aggregation Format (SAF)*
- *Part 21: MPEG-J Graphics Framework eXtensions (GFX)*
- *Part 22: Open Font Format*
- *Part 23: Symbolic Music Representation*
- *Part 24: Audio and systems interaction* [Technical Report]

Information technology — Coding of audio-visual objects —

Part 23: Symbolic Music Representation

1 Scope

This International Standard defines the Symbolic Music Representation technology. By capitalising the Symbolic Music Representation technology the acronym “SMR” has been derived.

A symbolic representation of music is a logical structure based on symbolic elements representing audiovisual events, the relationship between those events, and aspects related to how those events can be rendered and synchronized with other media types.

Many symbolic representations of music exist, including different styles of notation for chant, classical music, jazz and 20th-century styles; percussion notation; and simplified notations formats for children and vision-impaired readers. MPEG-4 SMR does not standardize one or more of these representations, but instead is an extensible language allowing those representations and many more which share a common underlying structure of music representation.

MPEG-4 SMR allows the synchronization of symbolic music elements with audiovisual events that existing standardized MPEG technology can represent and render.

The SMR technology is composed of different tools, which normative descriptions are in Clauses 7 to 12.

- **SMR Bitstream:** the syntax and semantics of the SMR bitstream.
- **Symbolic Music Extensible Format (SM-XF):** the syntax and semantics of the SMR format.
- **Symbolic Music Synchronization Information (SM-SI):** the syntax and semantics of the synchronisation Information between the SMR elements and the other audiovisual elements.
- **Symbolic Music Formatting Language (SM-FL):** the syntax and semantics of the rendering rules that are applied to the SMR XML format for rendering.
- **SMR Object Types for Profiles:** the object types of SMR to be used for the definition of Profiles.

2 Normative references

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

ISO/IEC 14496-1:2004, *Information technology — Coding of audio-visual objects — Part 1: Systems*

ISO/IEC 14496-3:2001, *Information technology — Coding of audio-visual objects — Part 3: Audio*

ISO/IEC 14496-11:2005, *Information technology — Coding of audio-visual objects — Part 11: Scene description and application engine*

ISO/IEC 23001-1, *Information technology — MPEG systems technologies — Part 1: Binary MPEG format for XML*

IETF RFC 1952: *GZIP File Format Specification Version 4.3*, P. Deutsch, May 1996

Extensible Markup Language 1.0 (second edition), W3C Recommendation, 6 October 2000, <http://www.w3.org/TR/2000/REC-xml-20001006>

XML Schema Part 1: Structures and Part 2: Datatypes, W3C Recommendation, 2 May 2001, <http://www.w3.org/TR/2001/REC-xmlschema-1-20010502>, <http://www.w3.org/TR/2001/REC-xmlschema-2-20010502>

Canonical XML version 1.0, W3C Recommendation, 15 March 2001, <http://www.w3.org/TR/2001/REC-xml-c14n-20010315>

3 Conformance

The issues related to the conformance are reported in ISO/IEC 14496-4, *Information technology — Coding of audio-visual objects — Conformance testing*.

4 Terms and Definitions

For the purposes of this document, the terms and definitions given in the Music notation glossary from MUSICNETWORK¹⁾ and the following apply.

4.1 accidental alteration

sign that prescribes the alteration of a note when located before it

NOTE Accidentals are: sharp and double sharp; flat and double flat; and natural.

4.2 bar

space existing between two strings located vertically with respect to the staff lines

NOTE Within the bar there is a series of music figures whose sum of duration values is equal to the indication set at the beginning of the music text right after the clef. Such an indication is expressed through a fraction (4/4, 2/4, 3/8, etc.) of the measurement unit: the whole.

4.3 BIFS

set of tools specified in ISO/IEC 14496-11 for the composition of media object data in interactive scenes

4.4 bitstream stream

ordered series of bits that forms the coded representation of the data

4.5 BNF Backus-Naur form

BNF is a description for context-free grammars of programming languages

1) <http://www.interactivemusicnetwork.org/glossary/index.htm>.

4.6**clef**

reference sign set at the beginning or within the music line with the aim of identifying a line with a given sound so to assign each note a valid position in the whole staff

4.7**CWMN**

Common Western Music Notation

4.8**figure**

generic terms for musical symbols such as notes, rest, etc.

4.9**note**

event with a start and a duration that produces sound

4.10**rest**

moment of silence in music

NOTE A rest can have a determined or an undetermined duration (fermata).

4.11**SM-FL binary format**

binary format of Symbolic Music Formatting Language

4.12**SMR decoder**

embodiment of the SMR decoding process

4.13**SM-SI binary format**

binary format of Symbolic Music Synchronization Information

4.14**SM-XF binary format**

binary format of Symbolic Music eXtensible Format

4.15**staff**

set of five horizontal lines and four interline spaces

NOTE Notes and rests are written both on the lines and among the spaces.

4.16**Symbolic Music eXtensible Format****SM-XF**

format for coding symbolic music representation

4.17**Symbolic Music Formatting Language****SM-FL**

language to formalize the rule for formatting symbolic music representation

4.18**Symbolic Music Synchronization Information****SM-SI**

format coding the synchronization information from symbolic music representation and audio-visual elements

4.19
XML
eXtensible Markup Language

5 Conventions

5.1 Naming convention

The Symbolic Music Formatting Language (SM-FL) specification (clause 10) contains several definitions that are used throughout the SMR normative text. When required, the naming adopted in clause 10 shall be considered as the ultimate naming convention for SMR concepts and names.

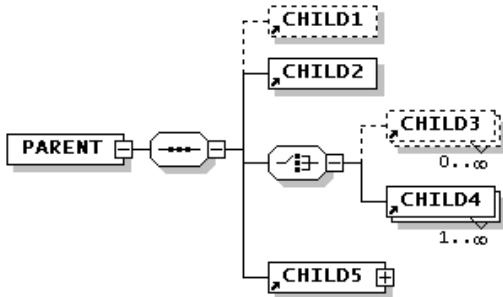
5.2 Documentation convention

The syntax of each element in the SMR is specified using the constructs provided by XML Schema (see XML Schema Part 1: Structures and Part 2: Datatypes).

Element names and attribute names in the representation are in SMALL CAPS. Throughout the document, *italics* are used when referring to elements defined in the SMR (see for example clause 8), hereafter known as the Model.

The syntax of each element in the SMR is specified using the following format.

Table 1 — Example element specification

Diagram			
Children	<CHILD1> <CHILD2> <CHILD3> <CHILD4> <CHILD5>		
Used by	<GRANDPARENT1> <GRANDPARENT2>		
Attributes	Name	Type	Description
	ID	ID	A unique ID value, which can be referenced by another element.
Source	<pre><xsd:element name="PARENT"> <xsd:complexType> <xsd:sequence> <xsd:element ref="CHILD1" minOccurs="0"/> <xsd:element ref="CHILD2"/> <xsd:choice> <xsd:element ref="CHILD3" minOccurs="0" maxOccurs="unbounded"/> <xsd:element ref="CHILD4" minOccurs="1" maxOccurs="unbounded"/> </xsd:choice> <xsd:element ref="CHILD5"/> </xsd:sequence> <xsd:attribute name="ID" type="xsd:id"/> </xsd:complexType> </xsd:element></pre>		

The Language Definition clause contains syntax diagrams for each element. Here is an example syntax diagram with annotations:

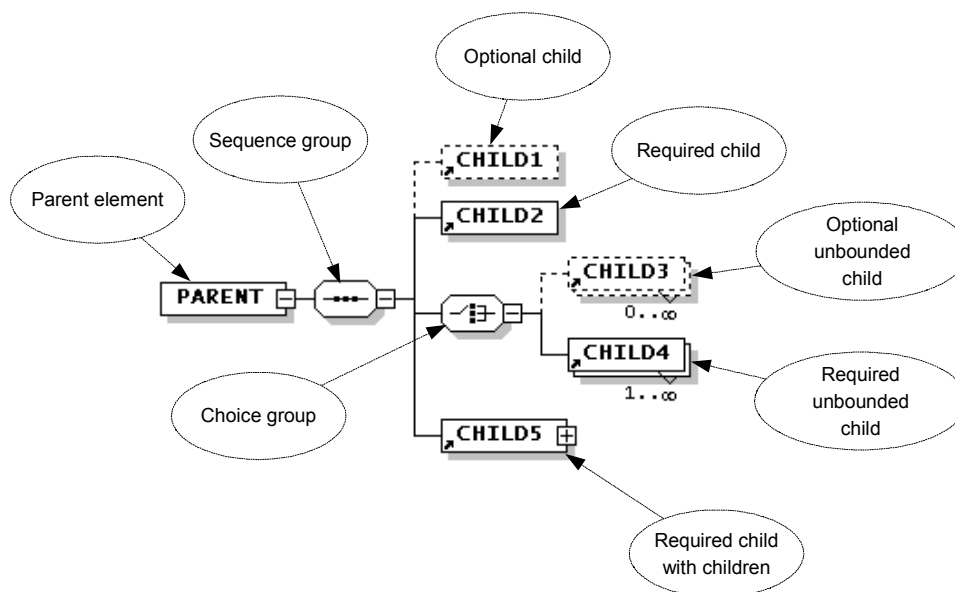


Figure 1 — Example element syntax diagram

Examples are included in separate clauses, and are shown in this document using a separate font and background:

```
<Example attribute1="example attribute value">
  <Element1>example element content</Element1>
</Example>
```

6 Symbols and abbreviations

The mathematical operators used to describe this subpart of ISO/IEC 14496-3 are similar to those used in the C/C++ programming language.

+	addition
-	subtraction
x or *	multiplication
/	division
exp	exponential function (base e)
log	natural logarithm
log10	base-10 logarithm
abs	absolute value
floor(x)	greatest integer less than or equal to x
ceil(x)	least integer greater than or equal to x
>	greater than
<	less than
>=	greater than or equal to
<=	less than or equal to
<> or !=	not equal to

Numeric formats and data types used in the XML schemas in this subpart are identified as follows:

boolean: it can assume two values: true and false.

enum: enumerate collection of possible values. List of possible values are reported in the XML schema.

nonNegativeInteger: an integer number that may range from 0.

positiveInteger: an integer number greater than zero.

string: a string of characters.

integer: a number that includes positive, negative and zero values without fractional part.

decimal: a floating point number.

7 SMR Bitstream

7.1 Introduction

The decoding process is described in SDL (see ISO/IEC 14496-1). This subclause describes the semantics of global decoding classes and the representation of types.

7.2 SMR Bitstream Introduction

The header streams are transported via MPEG-4 systems ISO/IEC 14496-1. These streams contain configuration information, which is necessary for the decoding process and parsing of the raw data streams. However, an update is only necessary if there are changes in the configuration.

The payloads contain all information varying on a frame to frame basis and therefore carry the actual audio information.

7.2.1 AudioSpecificConfig

This clause is informative, the AudioSpecificConfig definition can be found in ISO/IEC 14496-3:2005/Amd.5.

AudioSpecificConfig() extends the abstract class DecoderSpecificInfo, as defined in ISO/IEC 14496-1; when DecoderConfigDescriptor.objectTypeIndication refers to streams complying with ISO/IEC 14496-3 in this case the existence of AudioSpecificConfig() is mandatory.

Table 2 — Syntax of AudioSpecificConfig()

Syntax	No. of bits	Mnemonic
AudioSpecificConfig ()		
{		
audioObjectType = GetAudioObjectType();		
samplingFrequencyIndex;	4	uimsbf
if (samplingFrequencyIndex==0xf)		
samplingFrequency;	24	uimsbf
channelConfiguration;	4	uimsbf
sbrPresentFlag = -1;		
psPresentFlag = -1;		
if (audioObjectType == 5		
audioObjectType == 29) {		
extensionAudioObjectType = audioObjectType;		
sbrPresentFlag = 1;		
if (audioObjectType == 29) {		
psPresentFlag = 1;		
}		
extensionSamplingFrequencyIndex;	4	uimsbf
if (extensionSamplingFrequencyIndex==0xf)		
extensionSamplingFrequency;	24	uimsbf
audioObjectType = GetAudioObjectType();		
if (audioObjectType == 22)		
extensionChannelConfiguration;	4	uimsbf
}		
else {		
extensionAudioObjectType = 0;		
}		
switch (audioObjectType) {		
...		
case 28:		
SSCSpecificConfig();		
break;		
case 30:		
sacPayloadEmbedding;	1	uimsbf
SpatialSpecificConfig();		
break;		
case 32:		
case 33:		
case 34:		
...		
case 40:		
case 41:		
SymbolicMusicSpecificConfig()		
break;		
default:		
/* reserved */		
}		
...		

if (extensionAudioObjectType != 5 && bits_to_decode() >= 16) {		
syncExtensionType;	11	bslbf
if (syncExtensionType == 0x2b7) {		
extensionAudioObjectType = GetAudioObjectType();		
if (extensionAudioObjectType == 5) {		
sbrPresentFlag;	1	uimsbf
if (sbrPresentFlag == 1) {		
extensionSamplingFrequencyIndex;	4	uimsbf
if (extensionSamplingFrequencyIndex == 0xf) {		
extensionSamplingFrequency;	24	uimsbf
}		
if (bits_to_decode() >= 12) {		
syncExtensionType;	11	bslbf
if (syncExtensionType == 0x548) {		
psPresentFlag;	1	uimsbf
}		
}		
}		
}		
}		
if (extensionAudioObjectType == 22) {		
sbrPresentFlag;	1	uimsbf
if (sbrPresentFlag == 1) {		
extensionSamplingFrequencyIndex;	4	uimsbf
if (extensionSamplingFrequencyIndex == 0xf)		
extensionSamplingFrequency;	24	uimsbf
}		
extensionChannelConfiguration;	4	uimsbf
}		
}		

For the exact semantics of AudioSpecificConfig() please refer to ISO/IEC 14496-3 Subpart 1.

7.3 SMR Bitstream Description

7.3.1 SMR Bitstream Header

The `SymbolicMusicSpecificConfig` class is attached to an `AudioSpecificConfig` when an SMR object is coded. It is specified by the following semantics:

```

/*****
    bitstream formats
*****/

```

```

class SymbolicMusicSpecificConfig { // the bitstream header
    bit(4) version; //version of this specification is 0b0000

    unsigned int(12) pictureWidth; // rendering window X size
    unsigned int(12) pictureHeight; // rendering window Y size

    bit(1) isScoreMultiwindow; // 0: one window only - 1: multiple windows

    unsigned int(8) numberOfParts; // parts of the main score

    unsigned int(3) notationFormat; // CWMN or other sets

    vluimsbf8 urlMIDIStream_length; //length in bytes
    byte(urlMIDIStream_length) urlMIDIStream; // reference to the MIDI stream, as url

```

```

bit(2) codingType; // coding of the XML chunks

vluimsbf8 length; //length in bits of decoder configuration, unsigned integer
/** start of decoderConfiguration **/
if (codingType == 0b11) {
    bit(3) decoderInitConfig;
    if (decoderInitConfig == 0b000) {
        bit(length-3) decoderInit;
    }
} else {
    bit(length) reserved;
}
/** end of decoderConfiguration **/

bit more_data; // 1 if yes, 0 if no

while (more_data) {
    aligned bit(3) chunk_type;
    bit(5) reserved; // for alignment
    vluimsbf8 chunk_length; // length of the chunk in byte
    switch (chunk_type) {
        case 0b000 :
            mainscore_file sco;
            break;
        case 0b001 :
            bit(8) partID; // ID of the part at which the following info refers
            part_file npf;
            break;
        case 0b010 :
            // this segment is always in binary as stated in Section 9
            synch_file sync;
            break;
        case 0b011 :
            format_file fmt;
            break;
        case 0b100 :
            bit(8) partID;
            bit(8) lyricID;
            lyrics_file lyr;
            break;
        case 0b101 :
            // this segment is always in binary as stated in Section 11.4
            font_file fon;
            break;
        case 0b110 : reserved;
        case 0b111 : reserved;
    }
    aligned bit(1) more_data;
    bit(7) reserved; //for alignment
}
}

```

The **SymbolicMusicSpecificConfig** class contains all the information required to configure and start up a Symbolic Music Representation decoder. It specifies first the header information which shall be decoded with the following semantics:

version denotes the version of specification that this binary stream adheres to.

pictureWidth and **pictureHeight** specify the rendering size in points or pixels.

isScoreMultiWindow specifies if the score shall be rendered with one single window for all parts or by individual windows.

numberOfParts specifies how many parts are contained in the score.

notationFormat specifies if the score is a Common Western Music Notation format or other. For values of this field see Table 3.

Table 3 — Identification of Coding Type for SMR chunks

Value of Notation Format	Notation Format
0b000	reserved
0b001	CWMN
0b010	BRAILLE
0b011	SPOKENMUSIC
0b100	OTHER

urlMIDIStream specifies a possibly associated MIDI (i.e. MPEG4-SA Object Type 13) data stream to be rendered as score according to the specification in clause 11, Relationship with other parts of the standard. If a stream ID is passed to the decoder by an associated MusicScore node, this field is ignored.

Please note that **vluimsbf8**: Variable length code unsigned integer, with most significant bit first. The size of vluimsbf8 is a multiple of one byte. The first bit (Ext) of each byte specifies if set to 1 that another byte is present for this vluimsbf8 code word. The unsigned integer is encoded by the concatenation of the seven least significant bits of each byte belonging to this vluimsbf8 code word. If `urlMIDIStream_length` is zero the **ulrMIDIStream** is not present.

For the semantics of **codingType**, **decoderInitConfig** and **decoderInit** please refer to subclause 7.4

A sequence of one or more chunks follow, where each chunk belongs to one of the following file types: SM-XF file (main score and parts), SM-SI, SM-FL, lyrics, fonts; one or multiple chunks of each of these types may occur in the bitstream according to the following rules:

One and only one main SM-XF mainscore_file shall be transmitted in the bitstream. A sequence of one or more part_file chunks representing single parts can be present in the bitstream together with the main score. Multiple successive part_file chunks referring to the same partID are concatenated in coding order.

Lyrics chunks are referred to a specific part. If the partID refers to a part_file that has not been decoded yet these chunks shall be ignored.

Synchronisation chunks are referred to the main score and thus to the global synchronization of parts. This is valid for all parts.

If a format_file chunk is not present or a file is missing, the decoder shall use its own format file. Thus the decoder has to provide internally all the formatting rules to cover all the predefined symbols.

The font_file fon contains one or more OpenType fonts as defined in ISO/IEC 14496-18. If a font_file chunk is not present or a file is missing, the decoder shall use its own font files. Thus the decoder has to provide all the font files internally for covering all the predefined symbols.

The main score chunk shall be the first chunk in the bitstream. All other chunks may follow in any order provided that lyrics and font chunks referring to a partID come after the related part_file.

7.3.2 SMR Access Unit

The Symbolic Music Access Unit contains real-time streaming information to be provided to a running SMR decoding process. It may contain as many chunks as desired and as permitted by the available bandwidth. It may contain a sequence of one or more chunks as specified by the following code:

```
class SMR_access_unit {          // the streaming data

    bit(1) more_data = 1;
    bit(7) padding = 0000000;

    while (more_data) { // shall have at least one chunk
        aligned bit(3) chunk_type;
        bit(5) reserved; // for alignment
        vluimsbf8 chunk_length; // length of the chunk in byte
        switch (chunk_type) {
            case 0b000 :
                mainscore_file sco;
                break;
            case 0b001 :
                bit(8) partID; // ID of the part at which the following info refers
                part_file npf;
                break;
            case 0b010 :
                // this segment is always in binary as stated in Section 9
                synch_file sync;
                break;
            case 0b011 :
                format_file fmt;
                break;
            case 0b100 :
                bit(8) partID;
                bit(8) lyricID; // ID of the lyrics at which the following info refers
                lyrics_file lyr;
                break;
            case 0b101 :
                // this segment is always in binary as stated in Section 11.4
                font_file fon;
                break;
            case 0b110 : reserved;
            case 0b111 : reserved;
        }
        aligned bit(1) more_data;
        bit(7) reserved; // for alignment
    }
}
```

The same semantics apply as specified in **SymbolicMusicSpecificConfig**. In addition, lyrics chunks referring to a partID shall be considered for decoding if referring to part_file files received in the **SymbolicMusicSpecificConfig** in Access Units preceding the current Access Unit or in the same Access Unit.

Multiple successive part_file chunks referring to the same partID in the same AU are concatenated in coding order. The same rule applies for mainscore_file chunks, synch_file chunks, format_file chunks, font_chunks and lyrics_file chunks with the same lyricID and partID.

If additional format_file chunks and/or font_file chunks are received in successive AUs, this shall be interpreted as a replacement of the previous format_file and/or font_file.

An SA AU carrying MIDI information shall be decoded according to semantics and process specified in ISO/IEC 14496-3 Subpart 5.

7.4 Coding XML Segments

The XML segments of the SMR bitstream, namely: `mainscore_file`, `part_file`, `lyrics_file`, and `format_file`, can be coded in three ways. They can be packed into the bitstream as 1) uncompressed XML segments, or 2) compressed with GZIP IETF RFC 1952 or 3) compressed with BiM as specified in ISO/IEC 23001-1. The coding method used for all XML chunks is specified in the `SymbolicMusicSpecificConfig` using the `codingType` field – note that this does not apply to non XML chunks, i.e., `synch_file` and `font_file`. The different coding alternatives correspond to different application requirements of compression ratio and decoding simplicity for the SMR bitstream chunks. Table 4 gives the possible values of the `codingType` field.

Table 4 — Identification of Coding Type for SMR chunks

Value of <code>codingType</code>	Encoding Type
0b00	reserved
0b01	XML
0b10	GZip
0b11	BiM

In case of BiM encoding (`codingType` = 0b11), the decoding syntax of each chunk is described by `FragmentUpdatePayload(startType)` in subclause 7.3 of ISO/IEC 23001-1:2006, where `startType` identifies the XML data type of the encoded chunk according to Table 5.

Table 5 — Identification of `startType` for coding SMR chunks with BiM

chunk type	<code>startType</code>
<code>mainscore_file</code>	<code>SMXF_MainType</code>
<code>part_file</code>	<code>SMXF_PartType</code>
<code>format_file</code>	<code>SMFLType</code>
<code>lyrics_file</code>	<code>SMXF_LyricType</code>

The decoding of BiM encoded SMR chunks requires configuration information in the `SymbolicMusicSpecificConfig`, which is detailed in subclause 7.4.1. The `length` field in `SymbolicMusicSpecificConfig` allows SMR clients that are only interested in non-XML fragments (i.e., `synch_file` and `font_file`) to skip such information and provide room for future extensions.

7.4.1 BiM DecoderInit

The `DecoderInit` specified in this section is used to transmit initialisation information required by the BiM Decoder in the case where the `EncodingVersion` is set to '0b11' in the SMR codingType.

Table 6 —decoderInit for BiM encoding

Value of decoderInitConfig	Semantics
0b000	inline decoderInit
0b001	default decoderInit
0b010-0b111	reserved

The Syntax of the decoderInit is specified in ISO/IEC 23001-1. As shown in Table 6, the decoderInit is transmitted to the decoder when decoderInitConfig is set to 0b000. The value 0b001 corresponds to the default decoderInit, while other values are reserved. The default predefined decoderInit is conformant to ISO/IEC 23001-1 with the following constants:

Field	No. of bits	Value
SystemsProfileLevelIndication	8	'0x00'
UnitSizeCode	3	000
NoAdvancedFeatures	1	0
ReservedBits	4	0000
AdvancedFeaturesFlags_Length	8	0x01
InsertFlag	1	0
AdvancedOptimizedDecodersFlag	1	0
ReservedBitsZero	6	000000

The referenced schemas shall be the following schemas: smxf_main.xsd, smxf_part.xsd, smxf_lyric.xsd and smfl.xsd.

7.4.2 SMR Datatype Codecs

The following defines a set of codecs that shall be used by default for the encoding of SMR XML Fragments if BiM is used for the representation of SMR XML Fragments.

In the MPEG-7 framework, the use of a specific codec for a specific type is signalled using the codec configuration mechanism defined in ISO/IEC 23001-1. This mechanism associates a codec using its URI with a list of schema types. For that purpose, a URI is assigned to each codec in a classificationScheme, which defines the list of the specific codecs.

Table 7 — Coding Formats for SMR datatype

Data Type to be coded	description
boolean	it can assume two values true and false.
Enum	enumerate collection of possible values. List of possible values are reported in the XML schema.
nonNegativeInteger	an integer number that may range from 0.
positiveInteger	an integer number greater than zero.
string	a string of characters.
Integer	a number that include positive, negative and zero value without fractional part.
Decimal	a floating point number.

7.5 Decoding Process

This subclause describes the algorithmic Symbolic Music Representation decoding process, in which a bitstream conforming to object type 40 or 41 is rendered.

The general architecture of an SMR decoder is presented in Figure 2:

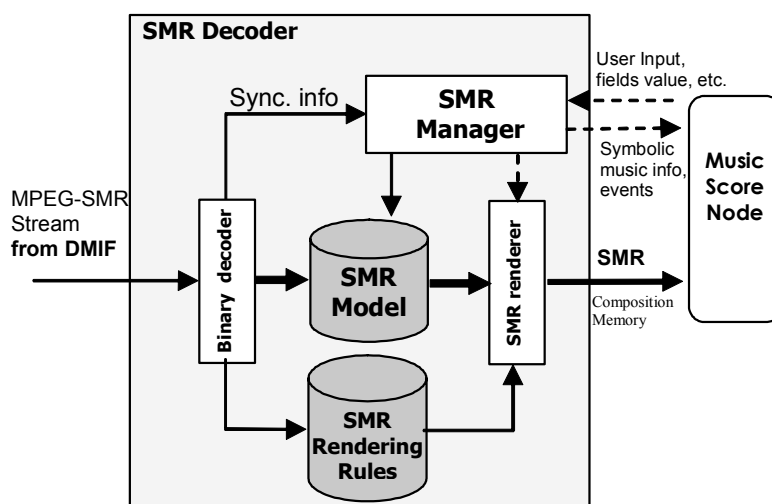


Figure 2 — MPEG SMR DECODER

7.5.1 Decoder Configuration

When an SMR decoder is instantiated following the opening of an SMR elementary stream, an object of class `SymbolicMusicSpecificConfig` is provided to that decoder as configuration information. At this time, the

decoder shall be initialized and then it shall parse the decoder configuration information into its chunk component and use them as follows:

- When a `mainscore_file` is found in the configuration, it shall be decoded by the adequate binary decoder as specified by **codingType**. If **codingType** is equal to 0x01 the content of the bitstream will be sent to the SMR Model parser. If **codingType** is equal to 0x10 or 0x11 the following bitstream will be sent to the gzip or Bim decoder respectively and their output will be fed to the SMR Model parser.
- The same rule as above applies to chunks of type `part_file`, `lyrics_file`, and `format_file`. In the case of `format_file`, information shall be sent to the formatting engine to interpret rendering rules.
- When a `synch_file` is found in the configuration, the information contained will be parsed and sent to the manager unit for synchronization (mapping logic timed events to real-time events)
- When a `font_file` is received, this shall be decoded according to the rules specified in ISO/IEC 14496-18 (see subclause 11.4) and the output shall be sent to the rendering engine.

For semantics of the different bitstream elements and in which order and number they can be presented to the decoder, please refer to subclause 7.3.1.

In the event that **MIDistream** contains a valid value, information referred to by this URL shall be used as specified in subclauses 7.5.2 and 11.3.

If the header information does not contain any chunk, the decoder shall delay its font configuration until the first access unit with valid chunks is received. At that moment the decoder decides if custom fonts shall be used (i.e., a `font_file` chunk is received in the first and/or following access units) or if the available fonts for predefined symbols shall be used instead.

7.5.2 Run-time Decoding Process

At each time step during the systems operation, the Systems ISO/IEC 14496-1 layer may present to the SMR decoder an Access Unit containing data conforming to the `SMR_access_unit` class, as well as structured audio access units (`SA_access_unit`, see ISO/IEC 14496-3 Subpart 5) containing MIDI information. It is responsibility of the SMR decoder to receive these AU data elements, to parse and understand them as the various SMR bitstream data chunks, to produce the corresponding normative output, and to present to the systems layer the Composition Unit.

The SMR decoding time step does not necessarily correspond to the systems time step, and it may or may not be synchronized with it. Nevertheless, the SMR decoder shall be able to receive Access Units according to the systems layer time step and provide output buffers at that rate.

At each internal time step, the SMR decoder shall first check for possible events received from the systems layer during the previous time period (see subclause 7.5.3) and then check for possible Access Units received during the same time period. `SMR_access_unit` data shall be decoded as specified in this standard. `SA_access_unit` data containing MIDI information shall be decoded as specified in ISO/IEC 14496-3 Subpart 5 with the limitations specified in subclause 11.3.

The several bitstream components contained in Access Units (as specified in subclause 7.3.2) shall be parsed and decoded following the same rules described in subclause 7.5.1. If a `font_file` containing graphics for custom symbols, or overloading for the predefined symbols, is not received before or at the same time of the first of each other chunk described above, it may be ignored by the decoder and in any case the rendering is not normatively specified. No normative rule for the display of custom symbols is defined if the corresponding `font_file` is not received prior or at the same time of the first symbol.

After parsing and decoding, the different decoder modules shall interpret each component according to syntax and semantics specified in clauses 8 to 11 and produce an output buffer through the SMR renderer.

If all the necessary elements to produce an output (as specified in clauses 8 to 11) are not present at a decoding step, the output of the decoder is unspecified.

7.5.3 Interaction with the Systems layer

At each time step during the systems operation the Systems layer may present to the SMR decoder events coming from the MusicScore node interface and at the same time request events possibly being generated by the SMR decoder during its decoding process.

Events coming from the MusicScore node interface shall be considered before the decoding of a possible Access Unit as described in subclause 7.5.2. Events generated by the SMR decoder will be presented to the Systems layer after decoding the possible Access Unit and after having accomplished all the tasks described in subclause 7.5.2.

Semantics of how these events shall be interpreted by the SMR decoder are described in subclause 11.2

8 Symbolic Music eXtensible Format (SM-XF): the Symbolic Music Representation

8.1 Symbolic Music eXtensible Format (SM-XF) introduction

Symbolic Music eXtensible Format is a core part of MPEG SMR standard. Its structure is defined in terms of (i) SM-XF Definitions reporting basic types used in the rest of the XML formalisation; (ii) specification of the single part structure and elements of the music, please refer to SMXF_Part schema, (iii) specification of the main score structure and elements of the MPEG SMR, please refer to SMXF_Main Schema, (iv) specification of the lyrics that can be plugged on single parts, please refer to SMXF_Lyric schema.

8.1.1 XML Namespaces

The following namespaces are defined for the SMXF:

urn:mpeg:mpeg-4:schema:smr:smxf-mainscore:2007 for the smxf_main.xsd schema

urn:mpeg:mpeg-4:schema:smr:smxf-part:2007 for the smxf_part.xsd schema

urn:mpeg:mpeg-4:schema:smr:smxf-lyric:2007 for the smxf_lyric.xsd schema

8.2 SM-XF Definitions

8.2.1 Simple Types and attribute groups

In this section the simple types and attribute groups used in the definition of the SM-XF format are defined.

8.2.1.1 sizeType

sizeType is a simple type for the size of some musical symbols which can be normal size or of in a reduced size.

```
<xs:simpleType name="sizeType">
  <xs:restriction base="xs:NMTOKEN">
    <xs:enumeration value="NORMAL"/>
    <xs:enumeration value="SMALL"/>
  </xs:restriction>
</xs:simpleType>
```

8.2.1.2 directionType

directionType is a simple type to express the position of music symbols with respect to the staff or to another symbol. It can assume values of UP, DOWN or AUTO. UP and DOWN mean above and below the referred symbol, respectively. AUTO means that the direction should be obtained using the formatting rules coded in SM-FL (see clause 10).

```

<xs:simpleType name="directionType">
  <xs:restriction base="xs:NMTOKEN">
    <xs:enumeration value="UP"/>
    <xs:enumeration value="DOWN"/>
    <xs:enumeration value="AUTO"/>
  </xs:restriction>
</xs:simpleType>

```

8.2.1.3 durationType

durationType is a simple type to express the duration of a note or rest.

```

<xs:simpleType name="durationType">
  <xs:restriction base="xs:NMTOKEN">
    <xs:enumeration value="D4M"/>
    <xs:enumeration value="D2M"/>
    <xs:enumeration value="D2"/>
    <xs:enumeration value="D1"/>
    <xs:enumeration value="D1_2"/>
    <xs:enumeration value="D1_4"/>
    <xs:enumeration value="D1_8"/>
    <xs:enumeration value="D1_16"/>
    <xs:enumeration value="D1_32"/>
    <xs:enumeration value="D1_64"/>
    <xs:enumeration value="D1_128"/>
    <xs:enumeration value="DGENERIC"/>
  </xs:restriction>
</xs:simpleType>

```

8.2.1.4 statusType

statusType is a simple type used to indicate the visual status of a note or of a rest symbol (see 8.3.23).

```

<xs:simpleType name="statusType">
  <xs:restriction base="xs:NMTOKEN">
    <xs:enumeration value="NORMAL"/>
    <xs:enumeration value="GRACED"/>
    <xs:enumeration value="HIDDEN"/>
    <xs:enumeration value="GHOSTED"/>
  </xs:restriction>
</xs:simpleType>

```

8.2.1.5 colorType

colorType is a simple type used to indicate the hexadecimal RGB code color (e.g. "#00ff00" for green).

```

<xs:simpleType name="colorType">
  <xs:restriction base="xs:string">
    <xs:pattern value="#[0-9A-Fa-f]{6,6}"/>
  </xs:restriction>
</xs:simpleType>

```

8.2.1.6 accidentalTypesType

accidentalTypesType is a simple type used to indicate the type of accidental. Table 8 shows the possible values.

Table 8 — Accidental types

Symbol	Description	token
#	sharp	SHARP
	flat	FLAT
	natural	NATURAL
	double sharp	DSHARP
	Double flat	DFLAT
	quarter sharp	SHARP1Q
	3/4 sharp	SHARP3Q
	a quarter flat	FLAT1Q
	3/4 flat	FLAT3Q
	3/4 natural	NATURALDOWN
	quarter natural	NATURALUP

```

<xs:simpleType name="accidentalTypesType">
  <xs:restriction base="xs:NMTOKEN">
    <xs:enumeration value="SHARP"/>
    <xs:enumeration value="FLAT"/>
    <xs:enumeration value="NATURAL"/>
    <xs:enumeration value="DSHARP"/>
    <xs:enumeration value="DFLAT"/>
    <xs:enumeration value="SHARP1Q"/>
    <xs:enumeration value="SHARP3Q"/>
    <xs:enumeration value="FLAT1Q"/>
    <xs:enumeration value="FLAT3Q"/>
    <xs:enumeration value="NATURALDOWN"/>
    <xs:enumeration value="NATURALUP"/>
  </xs:restriction>
</xs:simpleType>

```

8.2.1.7 clefTypesType

clefTypesType is a simple type used to indicate the type of clef, the possibilities of which are depicted in Table 9.

Table 9 — Clef types

TYPE	Example
BARITONE	
BASS	
BASSOLD	
ALTO	
MEZZOSOPRANO	
SOPRANO	
TENOR	
TENOR8	
TREBLE	
TREBLE8	
8TREBLE	
BASS8	
8BASS	
EMPTY	
PERCUSBOX	
PERCUS2LINES	
TAB	

```

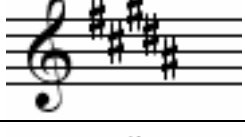
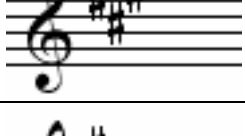
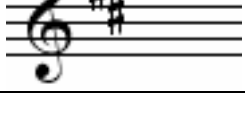
<xs:simpleType name="clefTypesType">
  <xs:restriction base="xs:NMTOKEN">
    <xs:enumeration value="BARITONE"/>
    <xs:enumeration value="BASS"/>
    <xs:enumeration value="BASSOLD"/>
    <xs:enumeration value="ALTO"/>
    <xs:enumeration value="MEZZOSOPRANO"/>
    <xs:enumeration value="SOPRANO"/>
    <xs:enumeration value="TENOR"/>
    <xs:enumeration value="TENOR8"/>
    <xs:enumeration value="TREBLE"/>
    <xs:enumeration value="TREBLE8"/>
    <xs:enumeration value="8TREBLE"/>
    <xs:enumeration value="BASS8"/>
    <xs:enumeration value="8BASS"/>
    <xs:enumeration value="EMPTY"/>
    <xs:enumeration value="PERCUSBOX"/>
    <xs:enumeration value="PERCUS2LINES"/>
    <xs:enumeration value="TAB"/>
  </xs:restriction>
</xs:simpleType>

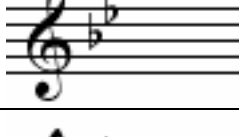
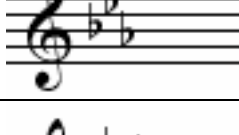
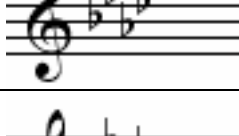
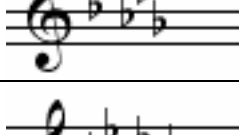
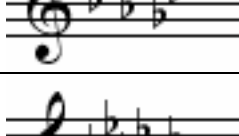
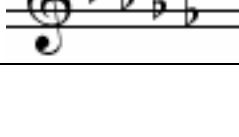
```

8.2.1.8 keysignatureTypesType

keysignatureTypesType a simple type used to indicate the type of key signature, the possibilities of which are that depicted in Table 10

Table 10 — Key signature types

TYPE		Example
Major	Minor	
D _O d _M	L _A d _m	
F _A d _M	R _E d _m	
S _I M	S _O L _d m	
M _I M	D _O d _m	
L _A M	F _A d _m	
R _E M	S _I m	

SOLM	MIm	
DOM	Lam	
FAM	REm	
SIbM	SOLm	
MIbM	Dom	
LAbM	FAm	
REbM	SIbm	
SOLbM	MIbm	
DObM	LAbm	

```

<xs:simpleType name="keysignatureTypesType">
  <xs:restriction base="xs:NMTOKEN">
    <xs:enumeration value="DOdM"/>
    <xs:enumeration value="FAdM"/>
    <xs:enumeration value="SIM"/>
    <xs:enumeration value="MIM"/>
    <xs:enumeration value="LAM"/>
    <xs:enumeration value="REM"/>
    <xs:enumeration value="SOLM"/>
    <xs:enumeration value="DOM"/>
    <xs:enumeration value="FAM"/>
    <xs:enumeration value="SIbM"/>
    <xs:enumeration value="MIbM"/>
    <xs:enumeration value="LAbM"/>
    <xs:enumeration value="REbM"/>
    <xs:enumeration value="SOLbM"/>
    <xs:enumeration value="DObM"/>
  </xs:restriction>
</xs:simpleType>

```

```

<xs:enumeration value="LAdm"/>
<xs:enumeration value="REdm"/>
<xs:enumeration value="SOLdm"/>
<xs:enumeration value="DODm"/>
<xs:enumeration value="FAdm"/>
<xs:enumeration value="SIm"/>
<xs:enumeration value="MIm"/>
<xs:enumeration value="LAm"/>
<xs:enumeration value="REm"/>
<xs:enumeration value="SOLm"/>
<xs:enumeration value="DOm"/>
<xs:enumeration value="FAm"/>
<xs:enumeration value="SIbm"/>
<xs:enumeration value="MIbm"/>
</xs:restriction>
</xs:simpleType>

```

8.2.1.9 justificationTypesType

justificationTypesType is a simple type used to indicate the type of justification algorithm to be used.

```

<xs:simpleType name="justificationTypesType">
  <xs:restriction base="xs:NMTOKEN">
    <xs:enumeration value="LINEAR"/>
    <xs:enumeration value="LOG"/>
  </xs:restriction>
</xs:simpleType>

```

8.2.1.10 noteHeadTypesType

noteHeadTypesType is a simple type used to indicate the type of notehead to be used. The HEAD of the note (see Table 11).

Table 11 — NOTE HEAD types

TYPES	EXAMPLE	Notehead examples
CLASSIC		
ALPHANUM		A H 45 4
ALPHANUM_SQUARE		
ALPHANUM_REVERSE		
CIRCLEX		

X		
DSHARP		
DIAMOND		 
CLUSTER		 
DOWNTRIANG		 
LEFTTRIANG		 
RIGHTTRIANG		 
SQUARE		 
TRIANG		 
UPTRIANGROUND		 
DOWNTRIANGROUND		 
UPTRIANG		 
CROIX		
MOON		 
PLUS		
RYTHMIC		 
THINRYTHMIC		
PARENTHESIS		
EMPTY		
DIDAPTIC		 

```

<xs:simpleType name="noteHeadTypesType">
  <xs:restriction base="xs:NMTOKEN">
    <xs:enumeration value="CIRCLEX"/>
    <xs:enumeration value="CLASSIC"/>
    <xs:enumeration value="ALPHANUM"/>
    <xs:enumeration value="ALPHANUM_SQUARE"/>
    <xs:enumeration value="ALPHANUM_REVERSE"/>
    <xs:enumeration value="CLUSTER"/>
    <xs:enumeration value="DIAMOND"/>
    <xs:enumeration value="DOWNTRIANG"/>
    <xs:enumeration value="LEFTTRIANG"/>
    <xs:enumeration value="PLUS"/>
    <xs:enumeration value="RHYTHMIC"/>
    <xs:enumeration value="RIGHTTRIANG"/>
    <xs:enumeration value="SQUARE"/>
    <xs:enumeration value="TRIANG"/>
    <xs:enumeration value="UPTRIANGROUND"/>
    <xs:enumeration value="DOWNTRIANGROUND"/>
    <xs:enumeration value="UPTRIANG"/>
    <xs:enumeration value="X"/>
    <xs:enumeration value="CROIX"/>
    <xs:enumeration value="DSHARP"/>
    <xs:enumeration value="MOON"/>
    <xs:enumeration value="THINRHYTHMIC"/>
    <xs:enumeration value="PARENTHESIS"/>
    <xs:enumeration value="EMPTY"/>
    <xs:enumeration value="DIDAPTIC"/>
  </xs:restriction>
</xs:simpleType>

```

The usage of the black filled versions or of the empty versions of the noteheads may depend on the needs and/or of the duration of the note.

8.2.1.11 dx dyAttributes

dx dyAttributes is an attribute group used by many elements to indicate the offset of the symbol (DX, DY) from the default positions in the different views: main score visual rendering (MAIN), single part visual rendering (PART), main score printing (PMAIN), single part printing (PPART). The default position of each symbol is estimated on the basis of the justification parameters and considering the SM-FL rules. Refer to clause 10.

```

<xs:attributeGroup name="dx dyAttributes">
  <xs:attribute name="MAINDX" type="xs:decimal" default="0"/>
  <xs:attribute name="MAINDY" type="xs:decimal" default="0"/>
  <xs:attribute name="PARTDX" type="xs:decimal" default="0"/>
  <xs:attribute name="PARTDY" type="xs:decimal" default="0"/>
  <xs:attribute name="PMAINDX" type="xs:decimal" default="0"/>
  <xs:attribute name="PMAINDY" type="xs:decimal" default="0"/>
  <xs:attribute name="PPARTDX" type="xs:decimal" default="0"/>
  <xs:attribute name="PPARTDY" type="xs:decimal" default="0"/>
</xs:attributeGroup>

```

8.2.1.12 dx dyrAttributes

dx dyrAttributes is an attribute group used in horizontal symbols (e.g., slurs) to indicate the offset of the right end of the horizontal symbol in the different views (the left end is indicated using the *dx dyAttributes*): main score visual rendering (MAIN), single part visual rendering (PART), main score printing (PMAIN), single part printing (PPART). The default position of each horizontal symbol is estimated on the basis of the justification parameters and considering the SM-FL rules. Refer to clause 10.

```

<xs:attributeGroup name="dx dyrAttributes">
  <xs:attribute name="MAINDXR" type="xs:decimal" default="0"/>
  <xs:attribute name="MAINDYR" type="xs:decimal" default="0"/>
  <xs:attribute name="PARTDXR" type="xs:decimal" default="0"/>
  <xs:attribute name="PARTDYR" type="xs:decimal" default="0"/>

```

```

<xs:attribute name="PMAINDXR" type="xs:decimal" default="0"/>
<xs:attribute name="PMAINDYR" type="xs:decimal" default="0"/>
<xs:attribute name="PPARTDXR" type="xs:decimal" default="0"/>
<xs:attribute name="PPARTDYR" type="xs:decimal" default="0"/>
</xs:attributeGroup>

```

8.2.1.13 fontAttributes

fontAttributes is an attribute group used to specify the font characteristics of the text to be rendered in the score. The attributes are

- FONTNAME, contains the font face name (empty means use default)
- FONTSTYLE, can be DEFAULT, NORMAL or ITALIC (default is DEFAULT)
- FONTWEIGHT, can be DEFAULT, NORMAL, LIGHT or BOLD (default is DEFAULT)
- FONTUNDERLINE, can be DEFAULT, true or false
- FONTSIZE, contains the font size in points (0 means use default value)

The actual value for attributes with default value is obtained from the context. For example the font attributes for a text indication in the score are assumed to be equal to the one specified for the score.

```

<xs:attributeGroup name="fontAttributes">
  <xs:attribute name="FONTNAME" type="xs:string"/>
  <xs:attribute name="FONTSTYLE" default="DEFAULT">
    <xs:simpleType>
      <xs:restriction base="xs:string">
        <xs:enumeration value="DEFAULT"/>
        <xs:enumeration value="NORMAL"/>
        <xs:enumeration value="ITALIC"/>
      </xs:restriction>
    </xs:simpleType>
  </xs:attribute>
  <xs:attribute name="FONTWEIGHT" default="DEFAULT">
    <xs:simpleType>
      <xs:restriction base="xs:string">
        <xs:enumeration value="DEFAULT"/>
        <xs:enumeration value="NORMAL"/>
        <xs:enumeration value="LIGHT"/>
        <xs:enumeration value="BOLD"/>
      </xs:restriction>
    </xs:simpleType>
  </xs:attribute>
  <xs:attribute name="FONTUNDERLINE" default="DEFAULT">
    <xs:simpleType>
      <xs:restriction base="xs:string">
        <xs:enumeration value="DEFAULT"/>
        <xs:enumeration value="false"/>
        <xs:enumeration value="true"/>
      </xs:restriction>
    </xs:simpleType>
  </xs:attribute>
  <xs:attribute name="FONTSIZE" type="xs:nonNegativeInteger" default="0"/>
</xs:attributeGroup>

```

8.3 Single Part (SMXF_Part)

8.3.1 <SMXF_Part>

The SMXF_PART element contains a part for a single instrument. It shall contain:

- An optional IDENTIFICATION element containing identification information (e.g., ISMN, ISRC, etc.),
- An optional sequence of zero or more CLASSIFICATION elements with multilingual metadata on the part (e.g. title, genre, author, etc.),
- An optional zero or more PREFERENCES elements containing preferences to be used for the decoding process,
- A SCORE element with the musical score encoding,
- A PRINTPAGES element containing additional text and images to be used when printing the score.

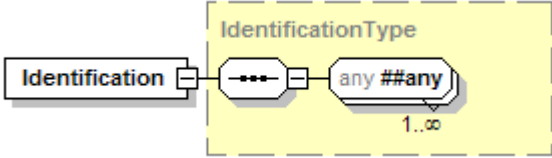
Table 12 — SWF_PART element syntax

Diagram	
Children	<Identification> <Classification> <preferences> <score> <printpages>
Source	<pre><xs:element name="SMXF_Part" type="SMXF_PartType"/> <xs:complexType name="SMXF_PartType"> <xs:sequence> <xs:element ref="Identification" minOccurs="0"/> <xs:element ref="Classification" minOccurs="0" maxOccurs="unbounded"/> <xs:element ref="preferences" minOccurs="0" maxOccurs="unbounded"/> <xs:element ref="score"/> <xs:element ref="printpages"/> </xs:sequence> </xs:complexType></pre>

8.3.2 <Identification>

The IDENTIFICATION element contains any identification information (represented as XML content).

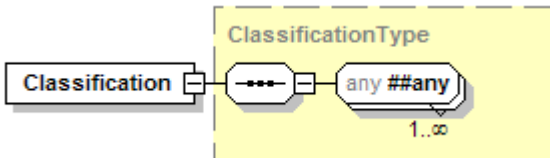
Table 13 — IDENTIFICATION element syntax

Diagram	
Children	any XML content
Used by	<SMXF_Part>
Source	<pre> <xs:element name="Identification" type="IdentificationType"/> <xs:complexType name="IdentificationType"> <xs:sequence> <xs:any namespace="##any" processContents="lax" maxOccurs="unbounded"/> </xs:sequence> </xs:complexType> </pre>

8.3.3 <Classification>

The CLASSIFICATION element contains any identification information (represented as XML content).

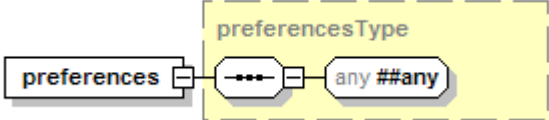
Table 14 — CLASSIFICATION element syntax

Diagram			
Children	any XML content		
Used by	<SMXF_Part>		
Attributes	Name	Type	Description
	Description	string	Contains a textual description for the classification information
Source	<pre><xs:element name="Classification" type="ClassificationType"/> <xs:complexType name="IdentificationType"> <xs:sequence> <xs:any namespace="##any" processContents="lax" maxOccurs="unbounded"/> </xs:sequence> <xs:attribute name="Description" type="xs:string" use="optional"/> <xs:anyAttribute namespace="##any"/> </xs:complexType></pre>		

8.3.4 <preferences>

The PREFERENCES element contains any preferences (represented as XML content) needed for the decoding process for a specific view (CWMN, BRAILLE music, SPOKENMUSIC or OTHER). Tablatures are included into the CWMN as a special case managed directly by chaing the number of line in the staff. This distinction is provided to allow the use of different rendering decoders for different views.

Table 15 — PREFERENCES element syntax

Diagram			
Children	any XML content		
Used by	<SMXF_Part>		
Attributes	Name	Type	Description
	for	enum	indicates the view for which the preferences are used (CWMN, BRAILLE, SPOKENMUSIC or OTHER)
Source	<pre><xs:element name="preferences" type="preferencesType"/> <xs:complexType name="preferencesType"> <xs:sequence> <xs:any namespace="##any"/> </xs:sequence> <xs:attribute name="for"> <xs:simpleType> <xs:restriction base="xs:string"> <xs:enumeration value="CWMN"/> <xs:enumeration value="BRAILLE"/> <xs:enumeration value="SPOKENMUSIC"/> <xs:enumeration value="OTHER"/> </xs:restriction> </xs:simpleType> </xs:attribute> </xs:complexType></pre>		

8.3.5 <score>

The SCORE element contains the music score for a single part. It contains:

- An optional ORIGIN element for information about the source from which it has been produced;
- An optional MIDIINFO element for information on how to produce MIDI from the score;
- An optional FORMATPAGECOMPUTERVIEW element with information on how to display the score in computer view;
- An optional FORMATPAGEPRINTVIEW element with information on how to print the score;
- A sequence of MEASURE elements containing all the measures in the score;
- A sequence of HORIZONTAL elements, for all the horizontal symbols (e.g., slurs, crescendo, diminuendo, etc.) in the score.

Table 16 — SCORE element syntax

Diagram			
Children	<origin> <midiinfo> <formatpagecomputerview> <formatpageprintview> <measure> <horizontal>		
Used by	<SWF_Part>		
Attributes	Name	Type	Description
	ID	positiveInteger	a number uniquely identifying each part in the piece
	NUMBEROFSTAFFS	positiveInteger	Number of staves can be 1, 2 or 3.
	TYPE	NMTOKEN	One of (NORMAL, PERCUS, TABLATURE). NORMAL means a staff, PERCUS means a single percussion line, TABLATURE means a representation of strings.
	TABLATURESIZE1	positiveInteger	Number of staff lines for staff 1
	TABLATURESIZE2	positiveInteger	Number of staff lines for staff 2 (if present)
	TABLATURESIZE3	positiveInteger	Number of staff lines for staff 3 (if present)
	INSTRUMENT	string	Name of the instrument
	SHORTINSTRNAME	string	Short instrument name (if empty the INSTRUMENT attribute should be used)
	LYRIC	string	the ID of the lyrics that should be shown after loading
	TRANPOSE	integer	the number of semitones to transpose the note pitches when generating audio (for transposing instruments)
	SIZE	sizeType	the size of the score (SMALL for a reduced size score)

Attribute Groups	fontAttributes	indicate font attributes for rendering text in the score (e.g. the Instrument name), if not specified or default a Times New Roman like font with a 10 pt size have to be used.
Source	<pre> <xs:element name="score" type="scoreType"/> <xs:complexType name="scoreType"> <xs:sequence> <xs:element ref="origin" minOccurs="0"/> <xs:element ref="midiinfo" minOccurs="0"/> <xs:element ref="formatpagecomputerview" minOccurs="0"/> <xs:element ref="formatpageprintview" minOccurs="0"/> <xs:element ref="measure" maxOccurs="unbounded"/> <xs:element ref="horizontal" minOccurs="0" maxOccurs="unbounded"/> </xs:sequence> <xs:attribute name="ID" type="xs:positiveInteger" use="required"/> <xs:attribute name="NUMBEROFSTAFFS" type="xs:positiveInteger" default="1"/> <xs:attribute name="TYPE" use="required"> <xs:simpleType> <xs:restriction base="xs:NMTOKEN"> <xs:enumeration value="NORMAL"/> <xs:enumeration value="PERCUS"/> <xs:enumeration value="TABLATURE"/> </xs:restriction> </xs:simpleType> </xs:attribute> <xs:attribute name="TABLATURESIZE1" type="xs:positiveInteger" default="5"/> <xs:attribute name="TABLATURESIZE2" type="xs:positiveInteger" default="5"/> <xs:attribute name="TABLATURESIZE3" type="xs:positiveInteger" default="5"/> <xs:attribute name="INSTRUMENT" type="xs:string" use="required"/> <xs:attribute name="SHORTINSTNAME" type="xs:string"/> <xs:attribute name="LYRIC" type="xs:string"/> <xs:attribute name="TRANSPOSE" type="xs:integer" default="0"/> <xs:attribute name="SIZE" type="sizeType" default="NORMAL"/> <xs:attributeGroup ref="fontAttributes"/> </xs:complexType> </pre>	

8.3.6 <origin>

The ORIGIN element is used to contain information about the origin of the score part; it may contain textual information about the software tool used to create it.

Table 17 — ORIGIN element syntax

Diagram			
Used by	<score>		
Attributes	Name	Type	Description
	FROM	string	the name of the tool used to produce the score
	LEVEL	string	The complexity level used in the tool
	RELEASE	string	the release or version of the tool used
Source	<pre> <xs:element name="origin" type="originType"/> <xs:complexType name="originType"> <xs:attribute name="FROM" type="xs:string"/> <xs:attribute name="LEVEL" type="xs:string"/> <xs:attribute name="RELEASE" type="xs:string"/> </xs:complexType> </pre>		

8.3.7 <midiinfo>

The MIDIINFO element contains basic settings (program code, volume and channel) for generation of MIDI events from the score (see Table 18).

Table 18 — MIDIINFO element syntax

Diagram			
Used by	<score>		
Attributes	Name	Type	Description
	CODE	nonNegativeInteger	MIDI program code (see [13])
	VOLUME	nonNegativeInteger	default volume for the score (0-255)
	CHANNEL	nonNegativeInteger	MIDI channel to be used for the score (please note that the score addresses a single part)
Source	<pre><xs:element name="midiinfo" type="midiinfoType"/> <xs:complexType name="midiinfoType"> <xs:attribute name="CODE" type="xs:nonNegativeInteger" use="required"/> <xs:attribute name="VOLUME" type="xs:nonNegativeInteger" use="required"/> <xs:attribute name="CHANNEL" type="xs:nonNegativeInteger" use="required"/> </xs:complexType></pre>		

8.3.8 <formatpagecomputerview>

The FORMATPAGECOMPUTERVIEW element contains information on how to display the score for rendering on a display device for that specific single part view. It contains the margins with respect to the page boundaries and the distance between staves.

Figure 3 represents the relations between the parameters and page shape.

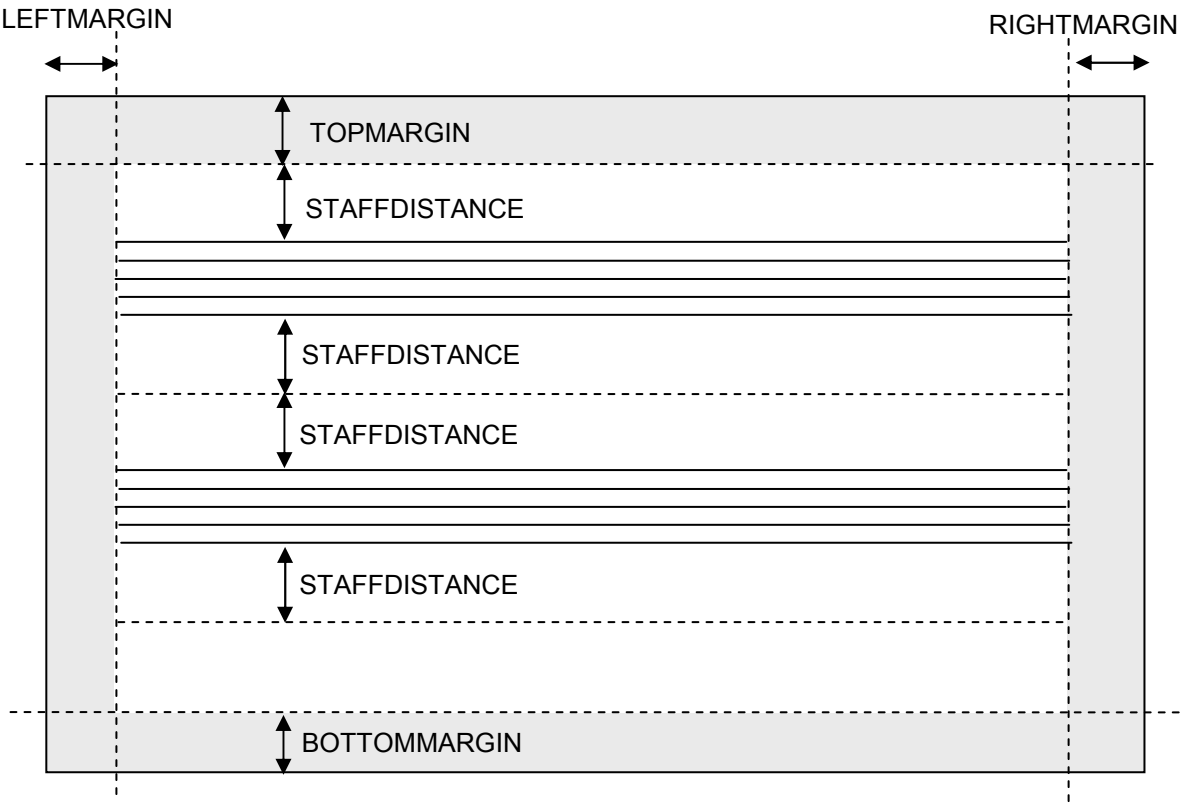


Figure 3 — FORMATPAGECOMPUTERVIEW parameters

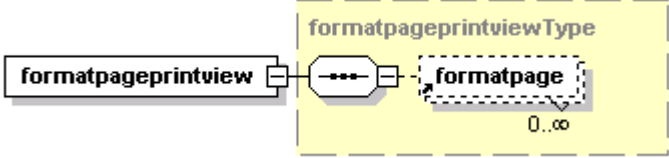
Table 19 — FORMATPAGECOMPUTERVIEW element syntax

Diagram	formatpagecomputerview		
Used by	<score>		
Attributes	Name	Type	Description
	TOPMARGIN	nonNegativeInteger	number of pixels to be left as top margin
	BOTTOMMARGIN	nonNegativeInteger	number of pixels to be left as bottom margin
	LEFTMARGIN	nonNegativeInteger	number of pixels to be left as left margin
	RIGHTMARGIN	nonNegativeInteger	number of pixels to be left as right margin
	STAFFDISTANCE	nonNegativeInteger	number of pixels to be left above and below a staff
	SCALE	decimal	the scaling factor to be applied for rendering
Source	<pre><xs:element name="formatpagecomputerview" type="formatpagecomputerviewType"/> <xs:complexType name="formatpagecomputerviewType"> <xs:attribute name="TOPMARGIN" type="xs:nonNegativeInteger"/> <xs:attribute name="BOTTOMMARGIN" type="xs:nonNegativeInteger"/> <xs:attribute name="LEFTMARGIN" type="xs:nonNegativeInteger"/> <xs:attribute name="RIGHTMARGIN" type="xs:nonNegativeInteger"/> <xs:attribute name="STAFFDISTANCE" type="xs:nonNegativeInteger"/> <xs:attribute name="SCALE" type="xs:decimal" default="1.0"/> </xs:complexType></pre>		

8.3.9 <formatpageprintview>

The FORMATPAGEPRINTVIEW element contains information on how to display the single part score for rendering on a printed page. It contains two FORMATPAGE elements with the information on page margins and the distance between staves, one for the first page and one for all the other pages.

Table 20 — FORMATPAGEPRINTVIEW element syntax

Diagram			
Children	<formatpage>		
Used by	<score>		
Attributes	Name	Type	Description
	PAGEFORMAT	enum	indicates the page size, it can be (A4, A3, A4R, A3R, Letter, Legal, B5)
	SCALE	decimal	the scale factor to be used
	RESOLUTION	enum	it can be 300 or 600 dpi
	FING1	string	a string to be written at the bottom of the page
	FING2	string	a string to be written at the bottom of the page
	FING3	string	a string to be written at the bottom of the page
Source	<pre> <xs:element name="formatpageprintview" type="formatpageprintviewType"/> <xs:complexType name="formatpageprintviewType"> <xs:sequence> <xs:element ref="formatpage" minOccurs="0" maxOccurs="unbounded"/> </xs:sequence> <xs:attribute name="PAGEFORMAT" default="A4"> <xs:simpleType> <xs:restriction base="xs:NMTOKEN"> <xs:enumeration value="A4"/> <xs:enumeration value="A3"/> <xs:enumeration value="A4R"/> <xs:enumeration value="A3R"/> <xs:enumeration value="Letter"/> <xs:enumeration value="Legal"/> <xs:enumeration value="B5"/> </xs:restriction> </xs:simpleType> </xs:attribute> <xs:attribute name="SCALE" type="xs:decimal" default="0.8"/> <xs:attribute name="RESOLUTION" default="600"> <xs:simpleType> <xs:restriction base="xs:NMTOKEN"> <xs:enumeration value="300"/> <xs:enumeration value="600"/> </xs:restriction> </xs:simpleType> </xs:attribute> <xs:attribute name="FING1" type="xs:string"/> <xs:attribute name="FING2" type="xs:string"/> <xs:attribute name="FING3" type="xs:string"/> </xs:complexType> </pre>		

8.3.10 <formatpage>

The FORMATPAGE element contains information on page margins and the distance between staves for the first page or for all the other pages (DEFAULT).

Table 21 — FORMATPAGE element syntax

Diagram			
Used by	<formatpageprintview>		
Attributes	Name	Type	Description
	PAGE	enum	indicates to which page the settings apply to, 1 for first page, DEFAULT for all the other pages
	TOPMARGIN	nonNegativeInteger	number of pixels to be left as top margin
	BOTTOMMARGIN	nonNegativeInteger	number of pixels to be left as bottom margin
	LEFTMARGIN	nonNegativeInteger	number of pixels to be left as left margin
	RIGHTMARGIN	nonNegativeInteger	number of pixels to be left as right margin
	STAFFDISTANCE	nonNegativeInteger	number of pixels to be left above and below a staff
	NSTAFFS	positiveInteger	number of staves to be put on the page
	INSPAGENUM	boolean	indicates if the page number has to be printed or not
	INSTITLE	boolean	indicates if the title has to be printed or not
Source	<pre> <xs:element name="formatpage" type="formatpageType"/> <xs:complexType name="formatpageType"> <xs:attribute name="PAGE" use="required"> <xs:simpleType> <xs:restriction base="xs:NMTOKEN"> <xs:enumeration value="1"/> <xs:enumeration value="DEFAULT"/> </xs:restriction> </xs:simpleType> </xs:attribute> <xs:attribute name="TOPMARGIN" type="xs:nonNegativeInteger"/> <xs:attribute name="BOTTOMMARGIN" type="xs:nonNegativeInteger"/> <xs:attribute name="LEFTMARGIN" type="xs:nonNegativeInteger"/> <xs:attribute name="RIGHTMARGIN" type="xs:nonNegativeInteger"/> <xs:attribute name="STAFFDISTANCE" type="xs:nonNegativeInteger"/> <xs:attribute name="NSTAFFS" type="xs:positiveInteger"/> <xs:attribute name="INSPAGENUM" type="xs:boolean"/> <xs:attribute name="INSTITLE" type="xs:boolean"/> </xs:complexType> </pre>		

8.3.10.1 Example

The following example describes a print view for A4 pages scaled down by 0.6 with respect to the natural size, and with a first page with a 400 point top margin, 50 points as bottom, left and right margins, 50 points distance above and below the staff, 4 staves, to print page numbers and not to print the title. For all the other pages it is equal except for a 50 point top margin and 8 staves per page.

```
<formatpageprintview PAGEFORMAT="A4" SCALE="0.6">
  <formatpage PAGE="1" TOPMARGIN="400" BOTTOMMARGIN="50" LEFTMARGIN="50"
  RIGHTMARGIN="50" STAFFDISTANCE="50" NSTAFFS="4" INSPAGENUM="true"
  INSTITLE="false" />
  <formatpage PAGE="DEFAULT" TOPMARGIN="50" BOTTOMMARGIN="50" LEFTMARGIN="50"
  RIGHTMARGIN="50" STAFFDISTANCE="50" NSTAFFS="8" INSPAGENUM="true"
  INSTITLE="false" />
</formatpageprintview>
```

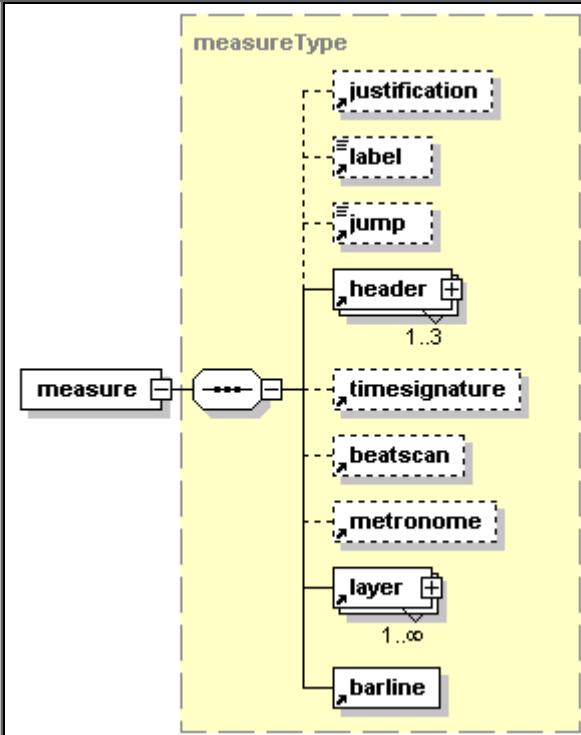
8.3.11 <measure>

The measure element describes a single measure of the score. It contains:

- An optional JUSTIFICATION element with information on how to justify the musical figures (notes/rests) in the measure. If this value is omitted the decoder has to estimate its internal default values,
- An optional LABEL element with the label given to the measure,
- An optional JUMP element with instructions for the software application performing the score (e.g., Da Capo),
- between one and three HEADER elements (one for each possible staff) with clefs and key signatures for the measure,
- An optional TIMESIGNATURE element with time signature indication,
- An optional BEATSCAN element,
- A sequence of LAYER elements. Each one containing a musical voice,
- A BARLINE element specifying the kind of barline to be used for closing the measure,

Other information on the measure is reported in the element attributes, see Table 22.

Table 22 — MEASURE element syntax

Diagram			
Children	<justification> <label> <jump> <header> <timesignature> <beatscan> <metronome> <layer> <barline>		
Used by	<score>		
Attributes	Name	Type	Description
	PROGRESSIVE	positiveInteger	progressive number of the measure, starting from the beginning
	ID	positiveInteger	a unique number identifying the measure within the score part
	NUMBEROFSTAFFS	positiveInteger	the number of staves used by the measure (1, 2 or 3)
	LINK	string	a link to some external information source (e.g., a URL)
Source	<pre><xs:element name="measure" type="measureType"/> <xs:complexType name="measureType"> <xs:sequence> <xs:element ref="justification" minOccurs="0"/> <xs:element ref="label" minOccurs="0"/> <xs:element ref="jump" minOccurs="0"/> <xs:element ref="header" maxOccurs="3"/> <xs:element ref="timesignature" minOccurs="0"/> <xs:element ref="beatscan" minOccurs="0"/> <xs:element ref="metronome" minOccurs="0"/> <xs:element ref="layer" maxOccurs="unbounded"/> <xs:element ref="barline"/> </xs:sequence> <xs:attribute name="PROGRESSIVE" type="xs:positiveInteger" use="required"/> <xs:attribute name="ID" type="xs:positiveInteger" use="required"/> <xs:attribute name="NUMBEROFSTAFFS" type="xs:positiveInteger"/> <xs:attribute name="LINK" type="xs:string"/> </xs:complexType></pre>		

8.3.12 <justification>

The justification element contains information on how to justify the measure; these parameters are used by the rendering algorithm to produce an accurate visualization. The parameters are:

- The justification type (xxxTYPE) which can be linear or logarithmic,
- A stretching parameter (xxxJUST) controlling the compression/expansion of the measure,
- A line breaking parameter (xxxLINEBREAK) stating if the measure is the last of the line,

These parameters can be different for the different views:

- Main score, computer view (xxx=MAIN),
- Single part, computer view (xxx=PART),
- Main score, print view (xxx=PMAIN),
- Single part, print view (xxx=PPART),

Table 23 — JUSTIFICATION element syntax

Diagram			
Used by	<measure>		
Attributes	Name	Type	Description
	MAINTYPE	justificationTypes Type	contains main score computer view justification type (LINEAR or LOG)
	MAINJUST	decimal	contains main score computer view justification parameter
	MAINLINEBREAK	boolean	indicates if the measure is line breaking for main score computer view
	PARTTYPE	justificationTypes Type	contains single part computer view justification type (LINEAR or LOG)
	PARTJUST	decimal	contains single part computer view justification parameter
	PARTLINEBREAK	boolean	indicates if the measure is line breaking for single part computer view
	PMAINTYPE	justificationTypes Type	contains main score print view justification type (LINEAR or LOG)
	PMAINJUST	decimal	contains main score print view justification parameter

	PMAINLINEBREAK	boolean	indicates if the measure is line breaking for main score print view
	PPARTTYPE	justificationTypesType	contains single part print view justification type (LINEAR or LOG)
	PPARTJUST	decimal	contains single part print view justification parameter
	PPARTLINEBREAK	boolean	indicates if the measure is line breaking for single part print view
Source	<pre> <xs:element name="justification" type="justificationType"/> <xs:complexType name="justificationType"> <xs:attribute name="MAINTYPE" type="justificationTypesType"/> <xs:attribute name="MAINJUST" type="xs:decimal"/> <xs:attribute name="MAINLINEBREAK" type="xs:boolean"/> <xs:attribute name="PARTTYPE" type="justificationTypesType"/> <xs:attribute name="PARTJUST" type="xs:decimal"/> <xs:attribute name="PARTLINEBREAK" type="xs:boolean"/> <xs:attribute name="PMAINTYPE" type="justificationTypesType"/> <xs:attribute name="PMAINJUST" type="xs:decimal"/> <xs:attribute name="PMAINLINEBREAK" type="xs:boolean"/> <xs:attribute name="PPARTTYPE" type="justificationTypesType"/> <xs:attribute name="PPARTJUST" type="xs:decimal"/> <xs:attribute name="PPARTLINEBREAK" type="xs:boolean"/> </xs:complexType> </pre>		

8.3.13 <label>

The label element contains information on the label of a measure. Three kinds of labels can be used:

- CODA, indicated with symbol Φ ,
- SEGNO, indicated with symbol S ,
- TEXT, a single letter label contained in the SM-XF content,

The label is positioned by default on the upper left side of the measure; however, the dx dyAttributes group can be used to specify an offset from the default position.

Table 24 — LABEL element syntax

Diagram			
Used by	<measure>		
Attributes	Name	Type	Description
	TYPE	enum	One of CODA, SEGNO or TEXT
Attribute Groups	dxdyAttributes		specify offsets (for the different views) of the symbol position from the default one
	fontAttributes		indicate the font to be used for rendering the label when TYPE is TEXT. The default values are the one specified in the score element. If FONTSIZE is not present or zero the size have to be 3 times the font size specified in the score element.
Source	<pre> <xs:element name="label" type="labelType"/> <xs:complexType name="labelType"> <xs:simpleContent> <xs:extension base="xs:string"> <xs:attribute name="TYPE" use="required"> <xs:simpleType> <xs:restriction base="xs:NMTOKEN"> <xs:enumeration value="CODA"/> <xs:enumeration value="SEGNO"/> <xs:enumeration value="TEXT"/> </xs:restriction> </xs:simpleType> </xs:attribute> <xs:attributeGroup ref="dxdyAttributes"/> <xs:attributeGroup ref="fontAttributes"/> </xs:extension> </xs:simpleContent> </xs:complexType> </pre>		

8.3.13.1 Example

A Segno label:

```
<label TYPE="SEGNO"/>
```

A text label:

```
<label TYPE="TEXT">A</label>
```

8.3.14 <jump>

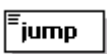
The JUMP element contains indication to the position of the of next measure to be performed. Many kinds of textual indications for jumps are used in the music notation literature, the most used are:

- "D.C." (DC) meaning from the beginning,
- "D.S." (DS) meaning from Segno label,
- "D.C. al Segno" (DCAS) meaning from begin up to the Segno label,

The TYPE attribute is used to indicate the kind of jump (DC, DS, DCAS) or to state that a generic text has to be written, in this case the text is in the SM-XF content.

The jump text is positioned by default on the lower right side of the measure; however, the dx dyAttributes group can be used to specify an offset from the default position.

Table 25 — JUMP element syntax

Diagram			
Used by	<measure>		
Attributes	Name	Type	Description
	TYPE	enum	one of DC, DS, DCAS or TEXT
Attribute Groups	dx dyAttributes	specify offsets (for the different views) of the symbol position from the default one	
	fontAttributes	indicate the font to be used for rendering the jump indication when TYPE is TEXT. The default values are the one specified in the score element.	
Source	<pre><xs:element name="jump" type="jumpType"/> <xs:complexType name="jumpType"> <xs:simpleContent> <xs:extension base="xs:string"> <xs:attribute name="TYPE" use="required"> <xs:simpleType> <xs:restriction base="xs:NMTOKEN"> <xs:enumeration value="DC"/> <xs:enumeration value="DS"/> <xs:enumeration value="DCAS"/> <xs:enumeration value="TEXT"/> </xs:restriction> </xs:simpleType> </xs:attribute> <xs:attributeGroup ref="dx dyAttributes"/> </xs:extension> </xs:simpleContent> </xs:complexType></pre>		

8.3.14.1 Example

A "Da Capo" jump:

```
<jump TYPE="DC"/>
```

A generic jump:

```
<jump TYPE="TEXT">Da Capo a Coda</jump>
```

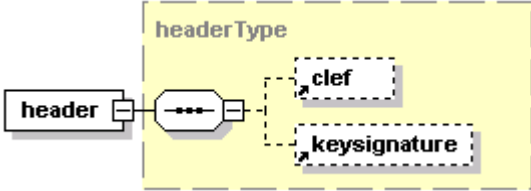
8.3.15 <header>

The HEADER element contains the information on the heading of the measure for a specific staff. It contains:

- An optional CLEF element with the clef to be used for the staff,
- An optional KEYSIGNATURE element with the keysignature to be used for the staff,

where the clef or the keysignature is not specified, the clef/keysignature indicated in the previous measure has to be used.

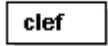
Table 26 — HEADER element syntax

Diagram	
Children	<code><clef></code> <code><keysignature></code>
Used by	<code><measure></code>
Source	<pre> <xs:element name="header" type="headerType"/> <xs:complexType name="headerType"> <xs:sequence> <xs:element ref="clef" minOccurs="0"/> <xs:element ref="keysignature" minOccurs="0"/> </xs:sequence> </xs:complexType> </pre>

8.3.16 <clef>

The CLEF element indicates, in the TYPE attribute, the clef to be used for the staff.

Table 27— CLEF element syntax

Diagram			
Used by	<code><header></code>		
Attributes	Name	Type	Description
	TYPE	clefTypesType	the type of clef to be used
	SIZE	sizeType	the size of the clef, NORMAL or SMALL
Source	<pre> <xs:element name="clef" type="clefType"/> <xs:complexType name="clefType"> <xs:attribute name="TYPE" type="clefTypesType" use="required"/> <xs:attribute name="SIZE" type="sizeType"/> </xs:complexType> </pre>		

8.3.17 <keysignature>

The KEYSIGNATURE element indicates, in the TYPE attribute, the key signature to be used for the staff.

Table 28 — KEYSIGNATURE element syntax

Diagram			
Used by	<header>		
Attributes	Name	Type	Description
	TYPE	keysignatureTypesType	the type of keysignature (see subclause 8.2.1.8)
Source	<pre><xs:element name="keysignature" type="keysignatureType"/> <xs:complexType name="keysignatureType"> <xs:attribute name="TYPE" type="keysignatureTypesType" use="required"/> </xs:complexType></pre>		

8.3.18 <timesignature>

The timesignature element contains information on the time division of the measure. The TYPE attribute is used to state the visual representation used for the time signature:

- with "C" use a 4/4 time represented as: ,
- with "CSLASH" use a 2/2 time represented as: ,
- with "FRACTION" use a numerical fraction, and only in this case the other attributes NUMERATOR, DENOMINATOR, TIMENUMERATOR and TIMEDENOMINATOR are considered.

In case of a normal fraction time (e.g., 2/4) the NUMERATOR and the DENOMINATOR attributes contain respectively the numerator and the denominator of the fraction. In the case of a compound time where the numerator or the denominator are shown as a sequence of added terms (e.g., 3+2/4 or 3/2+4) the NUMERATOR and the DENOMINATOR attributes contain the string to be displayed (e.g., "3+2") and the TIMENUMERATOR and the TIMEDENOMINATOR attributes contains real numerical value to be used for the resultant time signature value (e.g., 5).

Table 29 — TIMESIGNATURE element syntax

Diagram	timesignature		
Used by	<measure>		
Attributes	Name	Type	Description
	TYPE	enum	one of C, CSLASH or FRACTION
	NUMERATOR	string	numerator to be shown
	DENOMINATOR	string	denominator to be shown
	TIMENUMERATOR	nonNegativeInteger	numerator to be used internally
	TIMEDENOMINATOR	nonNegativeInteger	denominator to be used internally
Source	<pre> <xs:element name="timesignature" type="timesignatureType"/> <xs:complexType name="timesignatureType"> <xs:attribute name="TYPE" use="required"> <xs:simpleType> <xs:restriction base="xs:NMTOKEN"> <xs:enumeration value="C"/> <xs:enumeration value="CSLASH"/> <xs:enumeration value="FRACTION"/> </xs:restriction> </xs:simpleType> </xs:attribute> <xs:attribute name="NUMERATOR" type="xs:string"/> <xs:attribute name="DENOMINATOR" type="xs:string"/> <xs:attribute name="TIMENUMERATOR" type="xs:nonNegativeInteger"/> <xs:attribute name="TIMEDENOMINATOR" type="xs:nonNegativeInteger"/> <xs:attribute name="STATUS"> <xs:simpleType> <xs:restriction base="xs:NMTOKEN"> <xs:enumeration value="NORMAL"/> <xs:enumeration value="HIDDEN"/> <xs:enumeration value="ALWAYSVISIBLE"/> </xs:restriction> </xs:simpleType> </xs:attribute> </xs:complexType> </pre>		

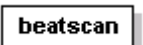
8.3.19 <beatscan>

The BEATSCAN element contains beat scanning information of the measure for its visual rendering. It shall be graphically represented as shown in Figure 4.



Figure 4 — BEAT SCAN EXAMPLE

Table 30 — BEATSCAN element syntax

Diagram			
Used by	<measure>		
Attributes	Name	Type	Description
	NBEATS	positiveInteger	number of beats into which the measure is divided
Source	<pre><xs:element name="beatscan" type="beatscanType"/> <xs:complexType name="beatscanType"> <xs:attribute name="NBEATS" type="xs:positiveInteger" use="required"/> </xs:complexType></pre>		

8.3.20 <metronome>

The METRONOME element contains the metronomic indication of the measure. It then applies to all the following measures until another metronomic indication occurs. The metronome element allows the determination of the real duration (in seconds) of music symbols such as note and rests. It expresses how many of the reference notes are present in one minute (Beats Per Minute or BPM).

The metronome indicator is positioned by default on the upper left side of the measure; however, the dx dy attributes can be used to move the indication from the default position.

Table 31 — METRONOME element syntax

Diagram			
Used by	<measure>		
Attributes	Name	Type	Description
	MOVEMENTTEXT	string	text explaining the movement (e.g. Allegro)
	NOTETYPE	durationType	the reference note duration
	DOTS	nonNegativeInteger	number of dots applied to the reference note
	VALUE	positiveInteger	indicates the number of notes expressed with the NOTETYPE and DOTS attributes in one minute
Attribute Group	dx dyAttributes		specify offsets (for the different views) of the symbol position from the default one
	fontAttributes		indicate the font to be used for rendering the movement text indication. The default values are the one specified in the score element.
Source	<pre><xs:element name="metronome" type="metronomeType"/> <xs:complexType name="metronomeType"> <xs:attribute name="MOVEMENTTEXT" type="xs:string"/> <xs:attribute name="NOTETYPE" type="durationType" use="required"/> <xs:attribute name="DOTS" type="xs:nonNegativeInteger" use="required"/> <xs:attribute name="VALUE" type="xs:positiveInteger" use="required"/> <xs:attributeGroup ref="dx dyAttributes"/> <xs:attributeGroup ref="fontAttributes"/> </xs:complexType></pre>		