INTERNATIONAL ORGANIZATION FOR STANDARDIZATION
ORGANISATION INTERNATIONALE NORMALISATION
ISO/IEC JTC 1/SC 29/WG 11
CODING OF MOVING PICTURES AND AUDIO

ISO/IEC JTC 1/SC 29/WG 11

MPEG2006/N8632

October 2006, Hangzhou, China

Source:	**Audio Subgroup**
Status:	**Proposed**
Title:	**ISO/IEC FCD 14496-23:200x, Symbolic Music Representation**
Editors:	**Pierfrancesco Bellini, Paolo Nesi, Maurizio Campanai, Giorgio Zoia**
	Public

Document type: International Standard
Document subtype:
Document stage: (40) Enquiry
Document language: E

STD Version 2.1c2

ISO/IEC JTC 1/SC 29

Date: 2006-10-27

ISO/IEC FCD 14496-23

ISO/IEC JTC 1/SC 29/WG 11

Secretariat:

Information technology — Coding of audio-visual objects — Part 23: Symbolic Music Representation

Élément introductif — Élément central — Partie 23: Titre de la partie

Contents

Page

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*

— *Part 8: Carriage of ISO/IEC 14496 contents over IP networks*

— *Part 9: Reference Hardware description*

— *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 (LASeR) and Simple Aggregation Format (SAF)*

— *Part 21: MPEG-J Graphical Framework eXtension (GFX)*

— *Part 22: Open Font Format*

— *Part 23: Symbolic Music Representation*

Information technology — Coding of audio-visual objects — Part 23: Symbolic Music Representation

1 Scope

1.1 Introduction

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.

1.2 Organisation of the document

The SMR technology is composed of different tools, which are described in specific normative clauses:

- **SMR Bitstream:** Clause 7 contains the normative description of the syntax and semantics of the SMR bitstream.
- **Symbolic Music Extensible Format (SM-XF):** Clause 8 contains the normative description of the syntax and semantics of the SMR format.
- **Symbolic Music Synchronization Information (SM-SI):** Clause 9 contains the normative description of the syntax and semantics of the synchronisation Information between the SMR elements and the other audiovisual elements.
- **Symbolic Music Formatting Language (SM-FL):** Clause 10 contains the normative description of the syntax and semantics of the rendering rules that are applied to the SMR XML format for rendering.
- **Relationship of SMR with other parts of the standard**: Clause 11 contains the normative description of the relationships of SMR with other parts of the MPEG-4 standard.
- **SMR Object Types for Profiles**: Clause 12 contains the normative description of 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 14496-1:2005, *Information technology — Coding of audio-visual objects — Part 1: Systems*

ISO/IEC 14496-3:2005, *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*

ISO/IEC 14977:1996*, Information technology — Syntactic metalanguage — Extended BNF*

ISO/IEC 15938-1, *Information technology -- Multimedia content description interface -- Part 1: Systems*

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 Part 4, Conformance Testing.

4 Terms and Definitions

For the purposes of this document, the following terms and definitions apply.

4.1
Accidental (or Alteration)

A sign that prescribes the alteration of a note when located before it. Accidentals are: sharp and double sharp; flat and double flat; and natural.

4.2
Bar

Within the staff the bar is meant as the space existing between two strings located vertically with respect to the staff lines. 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

The set of tools specified in ISO/IEC 14496-11 (MPEG-4 Scene Description) for the composition of media object data in interactive scenes.

4.4
Bitstream stream

An 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.

4.6
Clef

A 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

A moment of silence in music. It can have a determined or an undetermined duration (fermata).

4.11
SM-FL Binary Format

The binary format of Symbolic Music Formatting Language.

4.12 SMR Decoder

An embodiment of the SMR decoding process.

4.13
SM-SI Binary Format

The binary format of Symbolic Music Synchronization Information.

4.14
SM-XF Binary Format

The binary format of Symbolic Music Extensible Format.

4.15
Staff
A set of five horizontal lines and four interline spaces. Notes and rests are written both on the lines and among the spaces.

4.16
Symbolic Music Extensible Format (SM-XF)
The format for coding symbolic music representation.

4.17
Symbolic Music Formatting Language (SM-FL)
The language to formalize the rule for formatting symbolic music representation.

4.18
Symbolic Music Synchronization Information (SM-SI)
The format coding the synchronization information from symbolic music representation and audiovisual elements.

4.19
XML
Extensible Markup Language

For additional terms and definition refer to [2] (see Bibliography).

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.

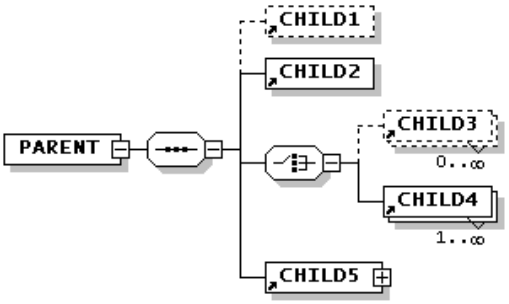
5.2 Documentation convention

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

<table>
<tr><td>Diagram</td><td colspan="3"> 
</td></tr>
<tr><td>Children</td><td colspan="3"><CHILD1> <CHILD2> <CHILD3> <CHILD4> <CHILD5></td></tr>
<tr><td>Used by</td><td colspan="3"><GRANDPARENT1> <GRANDPARENT2></td></tr>
<tr><td rowspan="2">Attributes</td><td>Name</td><td>Type</td><td>Description</td></tr>
<tr><td>ID</td><td>ID</td><td>A unique ID value, which can be referenced by another element.</td></tr>
<tr><td>Source</td><td colspan="3"><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></td></tr>
</table>

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

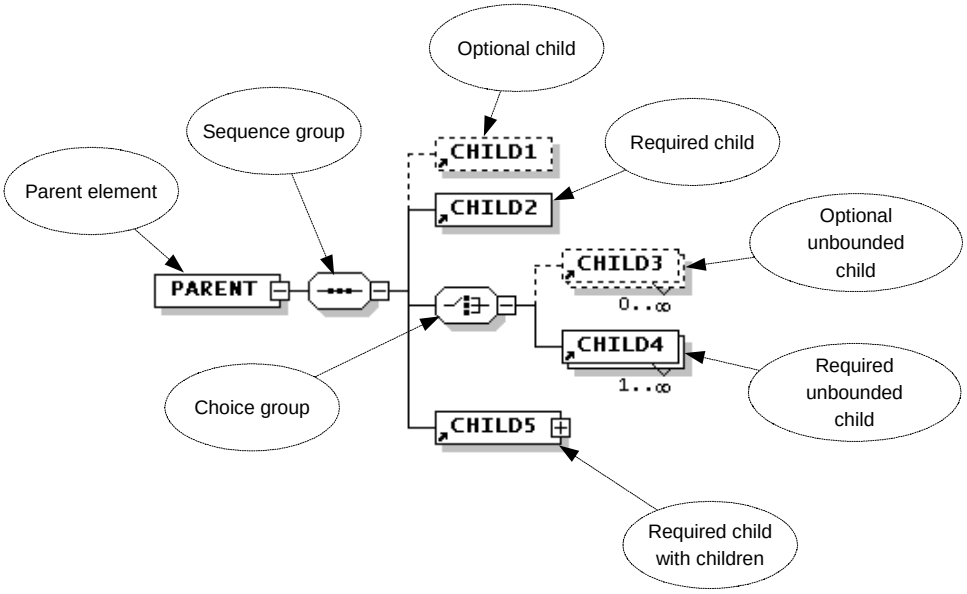


Figure 1 — Example element syntax diagram

Non-normative 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.

boolType: 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

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.

7.1 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.

7.1.1 AudioSpecificConfig

This subclause is Informative section.

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**	**bslbf**
if (samplingFrequencyIndex == 0xf) {		
samplingFrequency;	**24**	**uimsbf**
}		
channelConfiguration;	**4**	**bslbf**
sbrPresentFlag = -1;		
if (audioObjectType == 5) {		
extensionAudioObjectType = audioObjectType;		
sbrPresentFlag = 1;		
extensionSamplingFrequencyIndex;	**4**	**uimsbf**
if (extensionSamplingFrequencyIndex == 0xf)		
extensionSamplingFrequency;	**24**	**uimsbf**

```
        audioObjectType = GetAudioObjectType();
    }
    else {
        extensionAudioObjectType = 0;
    }
    switch (audioObjectType) {
    case 1:
    case 2:
    case 3:
    case 4:
    case 6:
    case 7:
    case 17:
    case 19:
    case 20:
    case 21:
    case 22:
    case 23:
        GASpecificConfig();
        break:
    case 8:
        CelpSpecificConfig();
        break;
    case 9:
        HvxcSpecificConfig();
        break:
    case 12:
        TTSSpecificConfig();
        break;
    case 13:
    case 14:
    case 15:
    case 16:
        StructuredAudioSpecificConfig();
        break;
    case 24:
        ErrorResilientCelpSpecificConfig();
        break;
    case 25:
        ErrorResilientHvxcSpecificConfig();
        break;
    case 26:
    case 27:
        ParametricSpecificConfig();
        break;
    case 28:
        SSCSpecificConfig();
        break;
    case 32:
    case 33:
    case 34:
        MPEG_1_2_SpecificConfig();
        break;
```

```
    case 35:
        DSTSpecificConfig();
        break;
    case 36:
    case 37:
        SymbolicMusicSpecificConfig();
    default:
        /* reserved */
    }
    switch (audioObjectType) {
    case 17:
    case 19:
    case 20:
    case 21:
    case 22:
    case 23:
    case 24:
    case 25:
    case 26:
    case 27:
        epConfig;                                                   2       bslbf
        if ( epConfig == 2 || epConfig == 3 ) {
            ErrorProtectionSpecificConfig();
        }
        if ( epConfig == 3 ) {
            directMapping;                                          1       bslbf
            if ( ! directMapping ) {
                /* tbd */
            }
        }
    }
    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
                }
            }
        }
    }
}
```

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

7.2 SMR Bitstream Description

7.2.1 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) pictureHight; // 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 `url`MIDIStream`_length` is zero the **ulrMIDIStream** is not present.

For the semantics of **codingType, decoderInitConfig and decoderInit** please refer to subclause 7.3

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 and fonts 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 file. 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.

7.2.2 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;

  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;
            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 or font 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 are concatenated in coding order. The same rule applies for synch_file chunks, and lyrics_file chunks.

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 semantincs and process specified in ISO/IEC 14496-3 Subpart 5 .

7.3 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 codingType	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, 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	`startType`
mainscore_file	SMXF_MainType
part_file	SMXF_PartType
format_file	SMFLType
lyrics_file	SMXF_LyricType

The decoding of BiM encoded SMR chunks requires configuration information in the `SymbolicMusicSpecificConfig`, which is detailed in subsection 7.3.1. 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.

7.3.1 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	5	11111
AdvancedFeaturesFlags_Length	8	0x08
InsertFlag	1	0
AdvancedOptimizedDecodersFlag	1	1
ReservedBitsZero	6	000000

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

7.3.2 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 — Conding Formats for SMR datatype

Data Type to be coded	description
boolean	it can assume two values true and false.
boolType	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.

7.4 Decoding Process

This subclause describes the algorithmic Symbolic Music Representation decoding process, in which a bitstream conforming to object type 36 or 37 is rendered.

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

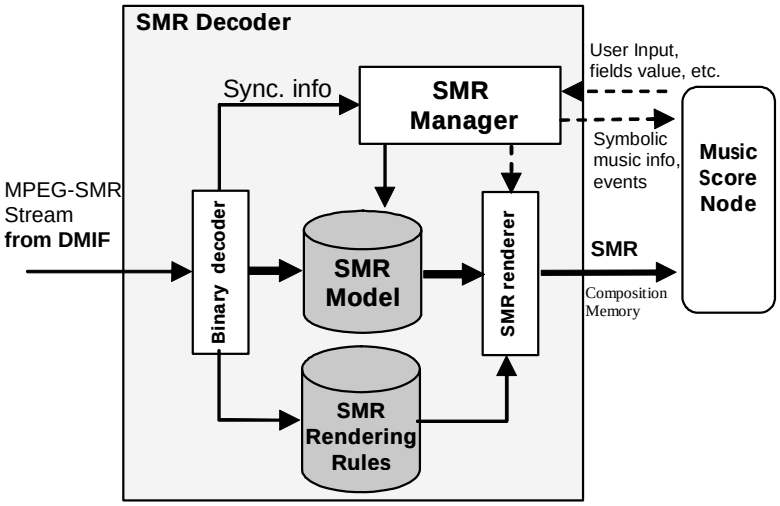


Figure 2 — MPEG SMR DECODER

7.4.1 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 and in this case the 0 char code is used as stream terminator. 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 7.2.1.

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.

7.4.2 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 7.2.2) 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.

7.4.3 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 stanrdard. 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.

8.2 SM-XF Definitions

8.2.1 Simple Types

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

8.2.1.1 boolType

boolType is a simple type for Boolean values

```
<xs:simpleType name="boolType">
   <xs:restriction base="xs:NMTOKEN">
      <xs:enumeration value="TRUE"/>
      <xs:enumeration value="FALSE"/>
   </xs:restriction>
</xs:simpleType>
```

8.2.1.2 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>
```

8.2.1.3 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>
```

8.2.1.4 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>
```

8.2.1.5 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>
```

8.2.1.6 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>
```

8.2.1.7 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>
```

8.2.1.8 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	

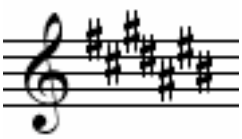
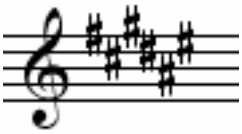
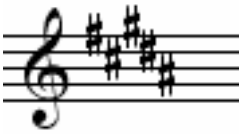
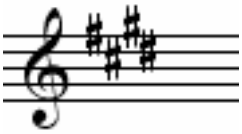
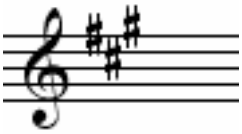
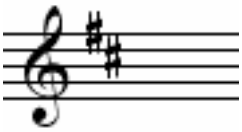
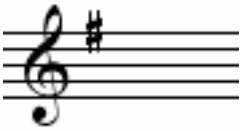
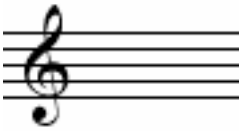
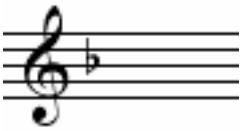
```
<xs:simpleType name="clefTypesType">
      <xs:restriction base="xs:NMTOKEN">
         <xs:enumeration value="BARITONE"/>
         <xs:enumeration value="BASS"/>
         <xs:enumeration value="BASSOLD"/>
         <xs:enumeration value="ALTO"/>
         <xs:enumeration value="MEZZOSOPRANO"/>
         <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"/>
```

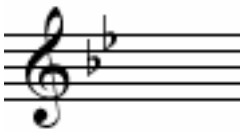
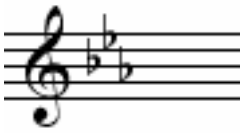
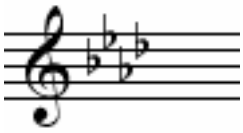
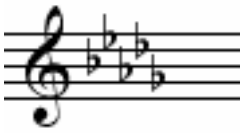
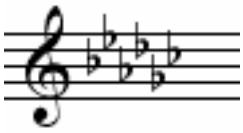
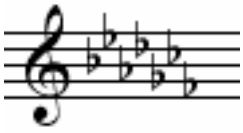
```
      <xs:enumeration value="PERCUSBOX"/>
      <xs:enumeration value="PERCUS2LINES"/>
      <xs:enumeration value="TAB"/>
    </xs:restriction>
  </xs:simpleType>
```

8.2.1.9 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		
Major	Minor	Example
DOdM	LAdm	
FAdM	REdm	
SIM	SOLdm	
MIM	DOdm	
LAM	FAdm	
REM	SIm	
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: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>
```

8.2.1.10 justificationTypesType

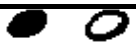
justificationTypesType is a simple type used to indicate the type of justification algorithm to be used.

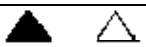
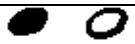
```
<xs:simpleType name="justificationTypesType">
   <xs:restriction base="xs:NMTOKEN">
      <xs:enumeration value="LINEAR"/>
      <xs:enumeration value="LOG"/>
   </xs:restriction>
</xs:simpleType>
```

8.2.1.11 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="THINRYTHMIC"/>
      <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.

8.2.1.12 dxdyAttributes

dxdyAttributes 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="dxdyAttributes">
   <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>
```

8.2.1.13 dxdyrAttributes

dxdyrAttributes 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 dxdyAttributes): 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="dxdyrAttributes">
   <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>
```

8.3 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.),

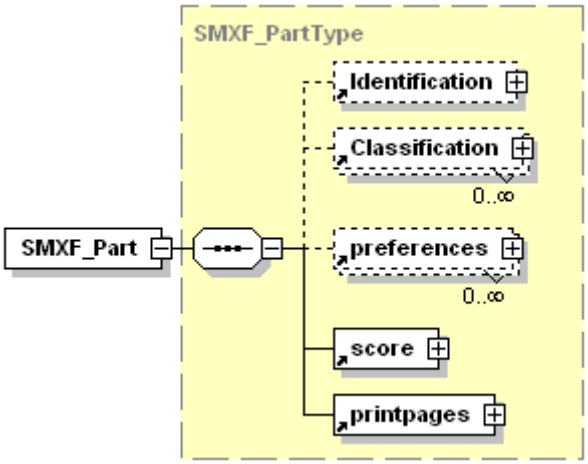
— An optional sequence of zero or more CLASSIFICATION elements with multilingual metadata on the part (e.g. title, genre, author, etc.),

— 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

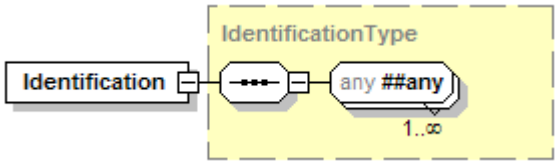
Diagram	
Children	<Identification> <Classification> <preferences> <score> <printpages>

Source	`<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>`

8.3.2 <Identification>

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

Table 13 — IDENTIFICATION element syntax

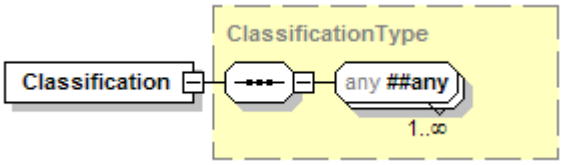
Diagram	
Children	any XML content
Used by	<SMXF_Part>

Source	`<xs:element name="Identification" type="IdentificationType"/>`
	`<xs:complexType name="IdentificationType">`
	`<xs:sequence>`
	`<xs:any namespace="##any" processContents="lax" maxOccurs="unbounded"/>`
	`</xs:sequence>`
	`</xs:complexType>`

8.3.3 <Classification>

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

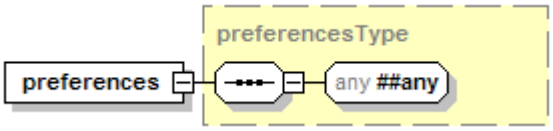
Table 14 — CLASSIFICATION element syntax

Diagram			
Children	any XML content		
Used by	<SMXF_Part>		
Attributes	**Name**	**Type**	**Description**
	Description	string	Contains a textual description for the classification information
Source	<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>		

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	<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>
```

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;

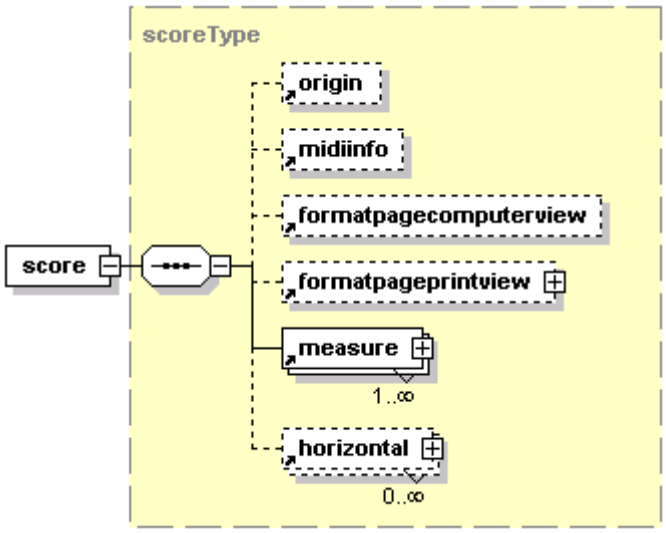
— An optional FORMATPAGECOMPUTERVIEW element with information on how to display the score in computer view;

— 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
	LYRIC	string	the ID of the lyrics that should be shown after loading

Source

```
   <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="LYRIC" type="xs:string"/>
    </xs:complexType>
```

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	<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>		

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	midiinfo		
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	<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>		

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:

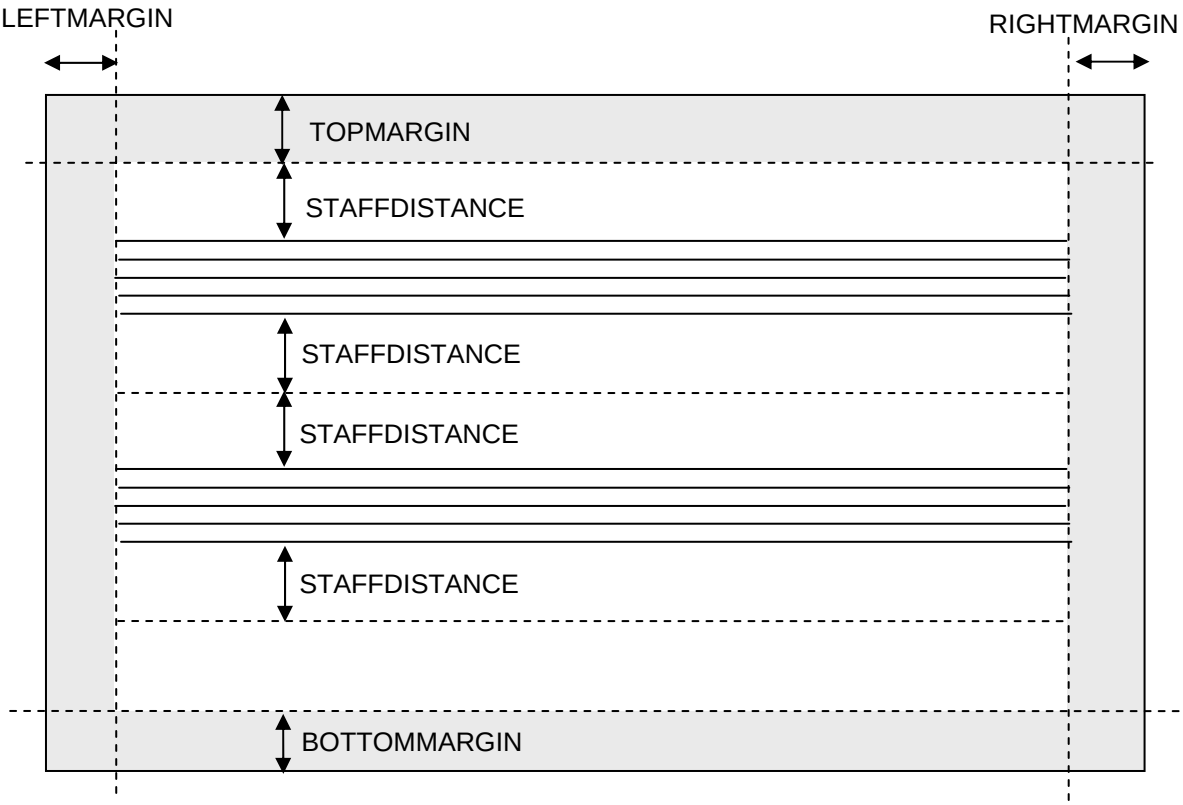


Figure 3 — FORMATPAGECOMPUTERVIEW parameters

Table 19 — FORMATPAGECOMPUTERVIEW element syntax

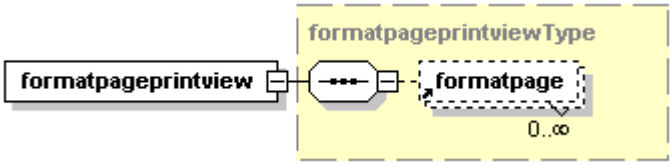
Diagram	formatpagecomputerview		
Used by	<score>		
Attributes	**Name**	**Type**	**Description**
	TOPMARGIN	nonNegativeInteger	number of pixels to be left as top margin
	BOTTOMMARGIN	nonNegativeInteger	number of pixels to be left as bottom margin
	LEFTMARGIN	nonNegativeInteger	number of pixels to be left 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
Source	<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:complexType>
```

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

```
<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>
```

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	boolType	indicates if the page number has to be printed or not
	INSTITLE	boolType	indicates if the title has to be printed or not
Source	<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="boolType"/>
      <xs:attribute name="INSTITLE" type="boolType"/>
    </xs:complexType>
```

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,

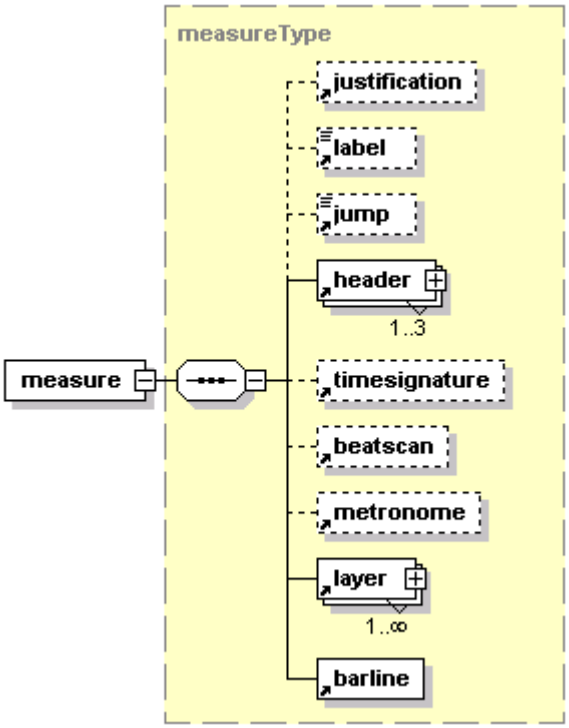
— An optional BEATSCAN element,

— 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

<table>
<tr><td>Diagram</td><td> 
</td></tr>
<tr><td>Children</td><td><justification> <label> <jump> <header> <timesignature> <beatscan> <metronome> <layer> <barline></td></tr>
<tr><td>Used by</td><td><score></td></tr>
<tr><td>Attributes</td><td colspan="2"><table>
<tr><th>Name</th><th>Type</th><th>Description</th></tr>
<tr><td>PROGRESSIVE</td><td>positiveInteger</td><td>progressive number of the measure, starting from the beginning</td></tr>
<tr><td>ID</td><td>positiveInteger</td><td>a unique number identifying the measure within the score part</td></tr>
<tr><td>NUMBEROFSTAFFS</td><td>positiveInteger</td><td>the number of staffs used by the measure (1, 2 or 3)</td></tr>
<tr><td>LINK</td><td>string</td><td>a link to some external information source (e.g., a URL)</td></tr>
</table></td></tr>
<tr><td>Source</td><td><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"/>
</pre></td></tr>
</table>

```
      <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>
```

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	justification		
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	boolType	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	boolType	indicates if the measure is line breaking for single part computer view

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

8.3.13 <label>

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

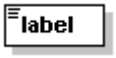
— CODA, indicated with symbol  ,

— SEGNO, indicated with symbol  ,

— 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 dxdyAttributes 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
Source	<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:extension> </xs:simpleContent> </xs:complexType>		

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 dxdyAttributes 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	dxdyAttributes	specify offsets (for the different views) of the symbol position from the default one	
Source	<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="dxdyAttributes"/> </xs:extension> </xs:simpleContent> </xs:complexType>		

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>

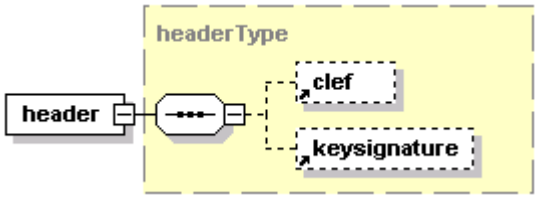
The HEADER element contains the information on the heading of the measure for a specific staff. It contains:

— 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	<clef> <keysignature>
Used by	<measure>

Source	`<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>`

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	clef		
Used by	<header>		
Attributes	**Name**	**Type**	**Description**
	TYPE	clefTypesType	the type of clef to be used
	SIZE	sizeType	the size of the clef, NORMAL or SMALL

Source	`<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>`		

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	keysignature		
Used by	<header>		
Attributes	**Name**	**Type**	**Description**
	TYPE	keysignatureTypesType	the type of keysignature (see subclause 8.2.1.9)

Source	`<xs:element name="keysignature" type="keysignatureType"/>`
	`<xs:complexType name="keysignatureType">`
	`<xs:attribute name="TYPE" type="keysignatureTypesType" use="required"/>`
	`</xs:complexType>`		

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

```
<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>
```

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.

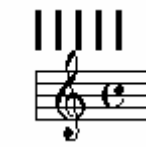


Figure 4 — BEAT SCAN EXAMPLE

Table 30 — BEATSCAN element syntax

Diagram	beatscan		
Used by	<measure>		
Attributes	**Name**	**Type**	**Description**
	NBEATS	positiveInteger	number of beats into which the measure is divided

Source

```
<xs:element name="beatscan" type="beatscanType"/>
<xs:complexType name="beatscanType">
   <xs:attribute name="NBEATS" type="xs:positiveInteger" use="required"/>
 </xs:complexType>
```

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 dxdy attributes can be used to move the indication from the default position.

Table 31 — METRONOME element syntax

Diagram	metronome		
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	dxdyAttributes		specify offsets (for the different views) of the symbol position from the default one
Source	<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="dxdyAttributes"/> </xs:complexType>		

8.3.20.1 Example

As an example, the following exemplar states that in a minute there has to be 100 dotted quarter notes, thus a dotted quarter note has a duration of 0.6 seconds.

```
<metronome MOVEMENTTEXT="Allegro" NOTETYPE="D1_4" DOTS="1" VALUE="100" />
```

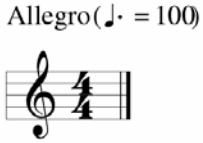


Figure 5 — Measure with metronome example

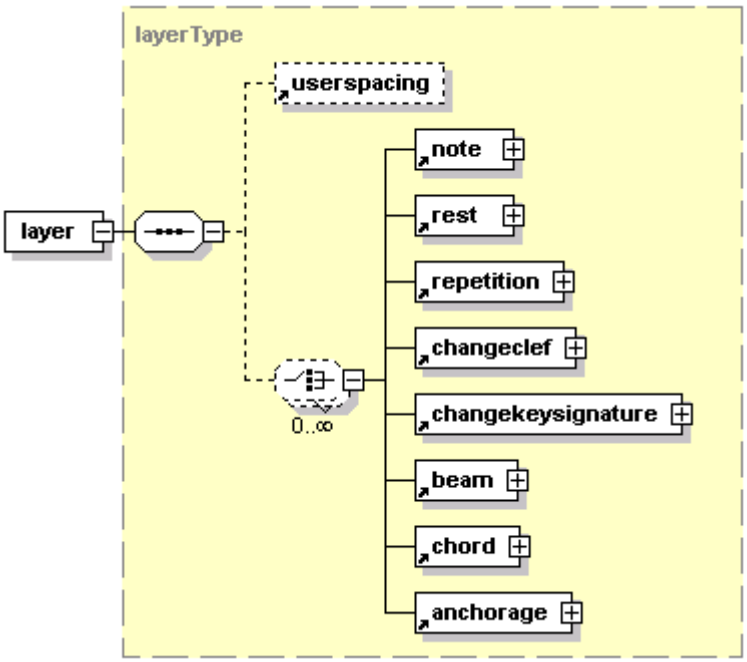
8.3.21 <layer>

The LAYER element is used to contain the musical symbols of a voice and thus to be played in sequence. It contains:

— An optional USERSPACING element with the user's alteration of the default space to the first symbol,

— A sequence of NOTE, REST, REPETITION, CHANGECLEF, CHANGEKEYSIGNATURE, BEAM, CHORD and ANCHORAGE elements to be displayed one after the other.

Table 32 — LAYER element syntax

<table>
<tr><td>Diagram</td><td> 
</td></tr>
<tr><td>Children</td><td><userspacing> <note> <rest> <repetition> <changeclef> <changekeysignature> <beam> <chord> <anchorage></td></tr>
<tr><td>Used by</td><td><measure></td></tr>
<tr><td>Attributes</td><td>
<table>
<tr><th>Name</th><th>Type</th><th>Description</th></tr>
<tr><td>NUMBER</td><td>positiveInteger</td><td>number of the layer</td></tr>
</table>
</td></tr>
<tr><td>Source</td><td>

```
<xs:element name="layer" type="layerType"/>
<xs:complexType name="layerType">
   <xs:sequence>
      <xs:element ref="userspacing" minOccurs="0"/>
      <xs:choice minOccurs="0" maxOccurs="unbounded">
         <xs:element ref="note"/>
         <xs:element ref="rest"/>
         <xs:element ref="repetition"/>
         <xs:element ref="changeclef"/>
         <xs:element ref="changekeysignature"/>
         <xs:element ref="beam"/>
         <xs:element ref="chord"/>
         <xs:element ref="anchorage"/>
      </xs:choice>
   </xs:sequence>
   <xs:attribute name="NUMBER" type="xs:positiveInteger" use="required"/>
```

</td></tr>
</table>

```
    </xs:complexType>
```

8.3.22 <userspacing>

The USERSPACING element contains the user's modification of the default horizontal spacing between a symbol and the next one in a layer (e.g., note, rest, etc.). In each view, the user spacing can be different; this is a rationale for having four values to be assigned:

— Main score, computer view (SPACEMAIN),

— Single part, computer view (SPACEPART),

— Main score, print view (SPACEPMAIN),

— Single part, print view (SPACEPPART).

The values can be positive or negative and are expressed in UL which can be transformed in pixel with the formula (ul+50)/100. Thus 3600UL is 36 pixels and 3680 UL is 37 pixels.

Table 33 — USERSPACING element syntax

Diagram	userspacing		
Used by	<layer> <note> <rest> <repetition> <changeclef> <changekeysignature> <chord>		
Attributes	**Name**	**Type**	**Description**
	SPACEMAIN	integer	modification of the space to the next symbol in the main score, computer view.
	SPACEPART	integer	modification of the space to the next symbol in the single part, computer view.
	SPACEPMAIN	integer	modification of the space to the next symbol in the main score, print view.
	SPACEPPART	integer	modification of the space to the next symbol in the single part, print view.
Source	<xs:element name="userspacing" type="userspacingType"/> <xs:complexType name="userspacingType"> <xs:attribute name="SPACEMAIN" type="xs:integer"/> <xs:attribute name="SPACEPART" type="xs:integer"/> <xs:attribute name="SPACEPMAIN" type="xs:integer"/> <xs:attribute name="SPACEPPART" type="xs:integer"/> </xs:complexType>		

8.3.23 <note>

The NOTE element represents a note in a layer or in a beam. It contains:

— An optional PITCH element to be used when the note is played (if not specified the pitch is derived from the context)

— An optional ACCIDENTAL element with the possible accidentals (e.g., sharp, flat, etc.) related to the note head,

— An optional AUGMENTATION element with the possible augmentation dots information of the note head,

— A sequence of zero or more FERMATA, DYNAMICTEXT, TEXT, ANNOTATION, GLASSES, FINGERING, STRINGS, ORNAMENT, KOREAN-ORNAMENT, MARKER, MUTE, HARMONIC, PIANO and FRETBOARD elements representing symbols to be positioned above or below the note,

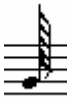
— A USERSPACING element for user's modification of the space to the next symbol in the layer.

The note attributes contain information on the note like:

— The ID uniquely identifying the note within the layer,

— The DURATION of the note (see Table 34)

Table 34— NOTE DURATIONS types

TYPES	EXAMPLE
D2	
D1	
D1_2	
D1_4	
D1_8	
D1_16	
D1_32	
D1_64	
D1_128	

— The STAFF of the measure on which the note has to be positioned. staff 0 is the upper one, staff 1 is the middle one or the lower one in case of two staves, staff 2 is the lower one in case of three staves.

— The HEIGHT is the position of the note on the staff. 0 is the lower staff line, 1 is the first space, 2 is the second staff line (from bottom), 3 is the second space, etc. Figure 6 depicts the relationship between the HEIGHT value and the position of the note on the staff.

— The STATUS attribute indicates if the note is visible or not and if the note has to have a visual duration or not. A note with visual duration will have a space to the following symbol (note, rest, etc.) proportional to the note duration, without visual duration the two symbols will be placed side by side. The status values are:

  — NORMAL: the note is visible and with visual duration,

  — GRACED: the note is visible and without visual duration,

  — HIDDEN: the note is invisible and with visual duration,

  — GHOSTED: the note is invisible and without visual duration.

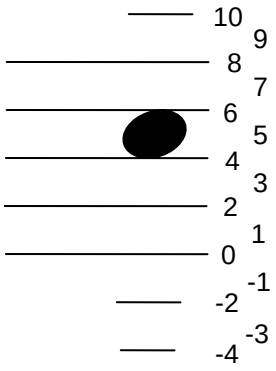
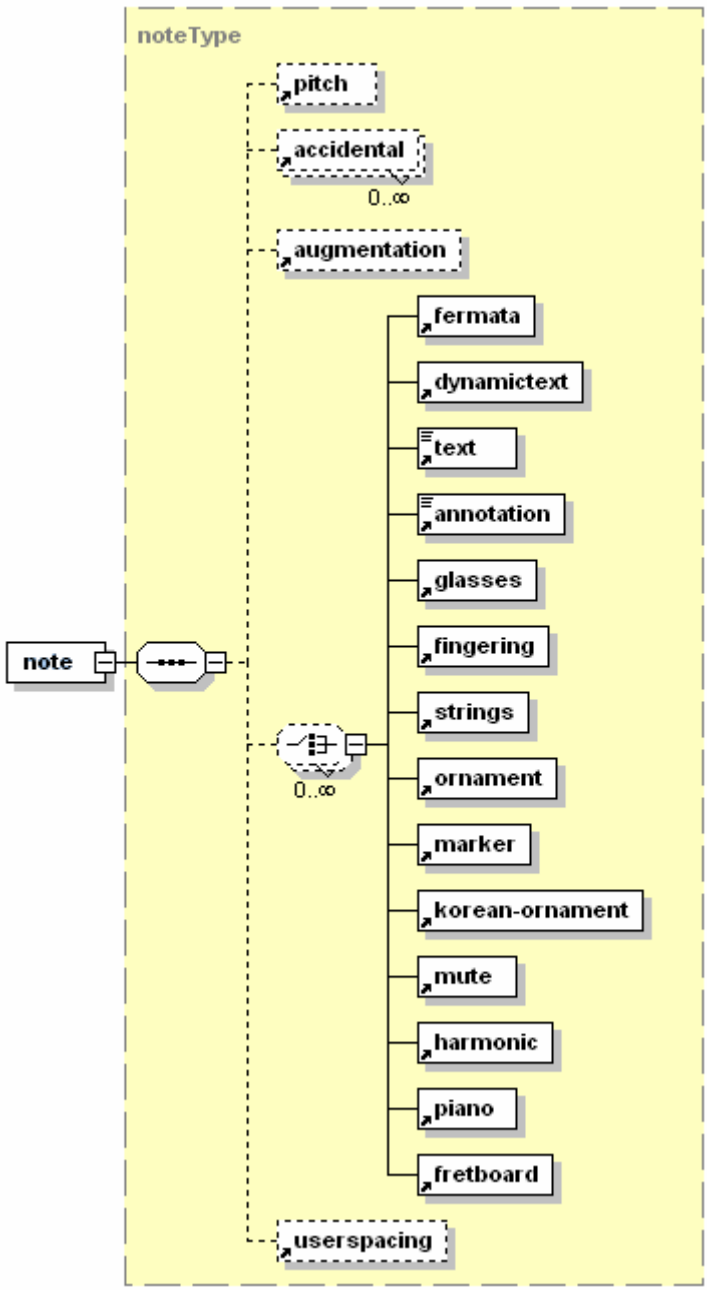


Figure 6 — Position of notes on a staff

— Other attributes are reported in Table 35.

Table 35 — NOTE element syntax

Diagram			
Children	<pitch> <accidental> <augmentation> <fermata> <dynamictext> <text> <annotation> <glasses> <fingering> <strings> <ornament> <korean-ornament> <marker> <mute> <harmonic> <piano> <fretboard>		
Used by	<layer> <beam>		
Attributes	**Name**	**Type**	**Description**
	ID	positiveInteger	a unique note identifier within the layer
	DURATION	durationType	the duration of the note (e.g. "D1_4" for a quarter note, "D1_8" for a eighth note, etc.)

	HEAD	noteHeadTypesType	the type of note head (e.g. "CLASSIC", "X", etc.)
	HEAD_ALPHANUM	string	when HEAD is "ALPHANUM" or "ALPHANUM_SQUARE" or "ALPHANUM_REVERSE" it contains the letter/number to be written in the note head.
	STEM	directionType	the direction of the note stem (e.g. "UP", "DOWN" or "AUTO")
	SIZE	sizeType	the size of the note (e.g. "SMALL" or "NORMAL")
	ACCIACCATURA	boolType	for an eighth note, is TRUE if it is an acciaccatura.
	STATUS	statusType	the status of the note ("NORMAL", "GRACED", "HIDDEN" or "GHOSTED")
	STAFF	nonNegativeInteger	number of staff within the measure where the note has to be put, can be 0,1 or 2
	HEIGHT	integer	position of the note on the staff (see Figure 6)
	LINK	string	a link associated with the note
	COLOR	colorType	the colour of the note

Source

```
<xs:element name="note" type="noteType"/>
<xs:complexType name="noteType">
   <xs:sequence>
      <xs:element ref="pitch" minOccurs="0"/>
      <xs:element ref="accidental" minOccurs="0" maxOccurs="unbounded"/>
      <xs:element ref="augmentation" minOccurs="0"/>
      <xs:choice minOccurs="0" maxOccurs="unbounded">
         <xs:element ref="fermata"/>
         <xs:element ref="dynamictext"/>
         <xs:element ref="text"/>
         <xs:element ref="annotation"/>
         <xs:element ref="glasses"/>
         <xs:element ref="fingering"/>
         <xs:element ref="strings"/>
         <xs:element ref="ornament"/>
         <xs:element ref="korean-ornament"/>
         <xs:element ref="marker"/>
         <xs:element ref="mute"/>
         <xs:element ref="harmonic"/>
         <xs:element ref="piano"/>
         <xs:element ref="fretboard"/>
      </xs:choice>
      <xs:element ref="userspacing" minOccurs="0"/>
   </xs:sequence>
   <xs:attribute name="ID" type="xs:positiveInteger" use="required"/>
   <xs:attribute name="DURATION" type="durationType" use="required"/>
   <xs:attribute name="HEAD" type="noteHeadTypesType" default="CLASSIC"/>
   <xs:attribute name="HEAD_ALPHANUM" type="xs:string"/>
   <xs:attribute name="STEM" type="directionType" default="AUTO"/>
   <xs:attribute name="SIZE" type="sizeType" default="NORMAL"/>
   <xs:attribute name="ACCIACCATURA" type="boolType" default="FALSE"/>
   <xs:attribute name="STATUS" type="statusType" default="NORMAL"/>
   <xs:attribute name="STAFF" type="xs:nonNegativeInteger" default="0"/>
   <xs:attribute name="HEIGHT" type="xs:integer" use="required"/>
```

	<xs:attribute name="LINK" type="xs:string"/> <xs:attribute name="COLOR" type="colorType"/> </xs:complexType>

8.3.24 <pitch>

The PITCH element specifies the musical pitch of the note when played.

Table 36 — PITCH element syntax

Diagram			
Used by	<note> <chordnote>		
Attributes	**Name**	**Type**	**Description**
	BASE	enum	the basic pitch in British notation (A, B, C, D, E, F, G)
	OCTAVE	integer	the octave position (e.g. C3 = middle C)
	SHIFT	enum	the optional accidental (#, *b*)
Source	<xs:element name="pitch" type="pitchType"/> <xs:complexType name="pitchType"> <xs:attribute name="BASE" use="required"> <xs:simpleType> <xs:restriction base="xs:string"> <xs:enumeration value="A"/> <xs:enumeration value="B"/> <xs:enumeration value="C"/> <xs:enumeration value="D"/> <xs:enumeration value="E"/> <xs:enumeration value="F"/> <xs:enumeration value="G"/> </xs:restriction> </xs:simpleType> </xs:attribute> <xs:attribute name="SHIFT" use="optional"> <xs:simpleType> <xs:restriction base="xs:string"> <xs:enumeration value="#"/> <xs:enumeration value="b"/> </xs:restriction> </xs:simpleType> </xs:attribute> <xs:attribute name="OCTAVE" type="xs:integer" use="required"/> </xs:complexType>		

8.3.25 <accidental>

The ACCIDENTAL element contains the optional accidental indication (sharp, double sharp, flat, double flat, etc.) related to a note head. The position of the accidental is normally on the left side of the note head the dxdyAttributes can be used to modify the default position.

Table 37 — ACCIDENTAL types

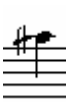
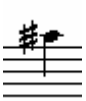
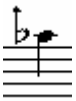
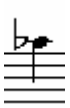
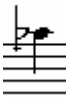
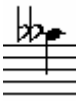
TYPES	EXAMPLE
NATURALUP	
NATURAL	
NATURALDOWN	
SHARP1Q	
SHARP	
SHARP3Q	
FLAT1Q	
FLAT	
FLAT3Q	
DOUBLESHARP	
DOUBLEFLAT	

Table 38 — ACCIDENTAL element syntax

Diagram	accidental		
Used by	<note> <chordnote>		
Attributes	**Name**	**Type**	**Description**
	TYPE	accidentalTypesType	the type of accidental (e.g. "SHARP", "FLAT", "NATURAL", etc.)
	ROUNDBRAKET	enum	can be YES or NO to indicate if the accidental has to be shown in brackets
	SIZE	sizeType	the size of accidental, can be "SMALL" or "NORMAL"
Attribute Group	dxdyAttributes		specify offsets (for the different views) of the accidental position from the default one
Source	<xs:element name="accidental" type="accidentalType"/> <xs:complexType name="accidentalType"> <xs:attribute name="TYPE" type="accidentalTypesType" use="required"/> <xs:attribute name="ROUNDBRACKET" default="NO"> <xs:simpleType> <xs:restriction base="xs:NMTOKEN"> <xs:enumeration value="YES"/> <xs:enumeration value="NO"/> </xs:restriction> </xs:simpleType> </xs:attribute> <xs:attribute name="SIZE" type="sizeType" default="NORMAL"/> <xs:attributeGroup ref="dxdyAttributes"/> </xs:complexType>		

8.3.26 <augmentation>

The AUGMENTATION element indicates the number of augmentation dots related to a note or to a chord.

Table 39 — AUGMENTATION element syntax

Diagram	augmentation		
Used by	<note> <rest> <chord>		
Attributes	**Name**	**Type**	**Description**
	DOTS	nonNegativeInteger	number of augmentation dots
Attributes Group	dxdyAttributes		specify offsets (for the different views) of the dots position from the default one
Source	<xs:element name="augmentation" type="augmentationType"/> <xs:complexType name="augmentationType">		

	<xs:attribute name="DOTS" type="xs:nonNegativeInteger" use="required"/> <xs:attributeGroup ref="dxdyAttributes"/> </xs:complexType>

8.3.27 <fermata>

The FERMATA element represents a fermata symbol associated with a note or a rest. Three kinds of fermata are possible:

Table 40 — FERMATA types

TYPES	EXAMPLE
SHORT	
MEDIUM	
LONG	

The symbol can be positioned above or below the note/rest/chord. The default position can be changed using the dxdyAttributes. When the symbols are placed below&down with respect to the staff they are depicted upside down.

Table 41 — FERMATA element syntax

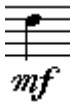
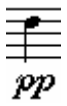
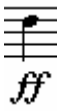
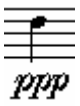
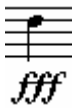
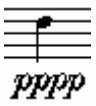
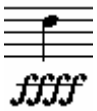
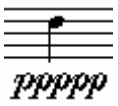
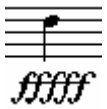
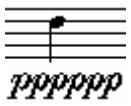
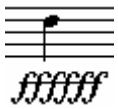
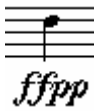
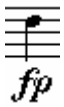
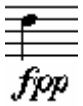
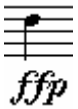
Diagram	fermata		
Used by	<note> <rest> <chord>		
Attributes	**Name**	**Type**	**Description**
	TYPE	enum	the type of fermata (SHORT, MEDIUM or LONG)
	UPDOWN	directionType	the position of fermata symbol (UP, DOWN or AUTO)
Attribute Group	dxdyAttributes		specify offsets (for the different views) of the fermata position from the default one
Source	<xs:element name="fermata" type="fermataType"/> <xs:complexType name="fermataType"> <xs:attribute name="TYPE" default="MEDIUM"> <xs:simpleType> <xs:restriction base="xs:NMTOKEN"> <xs:enumeration value="SHORT"/> <xs:enumeration value="MEDIUM"/> <xs:enumeration value="LONG"/>		

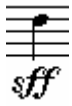
```
            </xs:restriction>
          </xs:simpleType>
        </xs:attribute>
        <xs:attribute name="UPDOWN" type="directionType" default="AUTO"/>
        <xs:attributeGroup ref="dxdyAttributes"/>
      </xs:complexType>
```

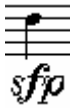
8.3.28 <dynamictext>

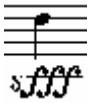
The DYNAMICTEXT element contains an indication of a musical dynamic related to a note, rest or chord.

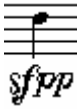
Table 42 — DYNAMICTEXT types

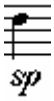
TYPES	EXAMPLE	TYPES	EXAMPLE
MP		MF	
P		F	
PP		FF	
PPP		FFF	
PPPP		FFFF	
PPPPP		FFFFF	
PPPPPP		FFFFFF	
FFPP		FP	
FPP		FFP	

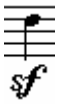
SFF 

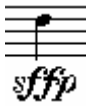
SFP 

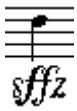
SFFF 

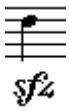
SFPP 

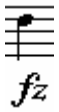
SP 

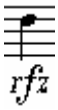
SF 

SFFP 

SFFZ 

SFZ 

FZ 

RFZ 

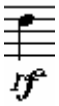
RF 

Table 43 — DYNAMICTEXT element syntax

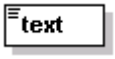
Diagram	dynamictext		
Used by	<note> <rest> <chord>		
Attributes	**Name**	**Type**	**Description**
	DYNAMIC	enum	the dynamics text type (e.g. P, FF, RFZ) (see Table 42)
	UPDOWN	directionType	the position of dynamics text (UP, DOWN or AUTO)
Attribute Group	dxdyAttributes		specify offsets (for the different views) of the dynamics text position from the default one
Source	<xs:element name="dynamictext" type="dynamictextType"/> <xs:complexType name="dynamictextType"> <xs:attribute name="DYNAMIC" use="required"> <xs:simpleType> <xs:restriction base="xs:NMTOKEN"> <xs:enumeration value="P"/> <xs:enumeration value="PP"/> <xs:enumeration value="PPP"/> <xs:enumeration value="PPPP"/> <xs:enumeration value="PPPPP"/> <xs:enumeration value="PPPPPP"/> <xs:enumeration value="MP"/> <xs:enumeration value="MF"/>		

```
            <xs:enumeration value="F"/>
            <xs:enumeration value="FF"/>
            <xs:enumeration value="FFF"/>
            <xs:enumeration value="FFFF"/>
            <xs:enumeration value="FFFFF"/>
            <xs:enumeration value="FFFFFF"/>
            <xs:enumeration value="FFPP"/>
            <xs:enumeration value="FFP"/>
            <xs:enumeration value="FPP"/>
            <xs:enumeration value="RFZ"/>
            <xs:enumeration value="SFF"/>
            <xs:enumeration value="SFFF"/>
            <xs:enumeration value="SFFP"/>
            <xs:enumeration value="SFFZ"/>
            <xs:enumeration value="SFP"/>
            <xs:enumeration value="SFPP"/>
            <xs:enumeration value="SP"/>
            <xs:enumeration value="SF"/>
            <xs:enumeration value="SFZ"/>
            <xs:enumeration value="FZ"/>
            <xs:enumeration value="FP"/>
            <xs:enumeration value="RF"/>
         </xs:restriction>
      </xs:simpleType>
    </xs:attribute>
    <xs:attribute name="UPDOWN" type="directionType" default="AUTO"/>
    <xs:attributeGroup ref="dxdyAttributes"/>
  </xs:complexType>
```

8.3.29 <text>

The TEXT element contains a textual indicator related to a note/rest/chord. The text is in the XML content.

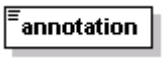
Table 44 — TEXT element syntax

Diagram			
Used by	<note> <rest> <chord> <anchorage>		
Attributes	**Name**	**Type**	**Description**
	UPDOWN	directionType	position of the text with respect to the note/rest/chord, can be UP, DOWN or AUTO
	HIDDEN	boolType	if TRUE the text is not to be shown
Attribute Group	dxdyAttributes		specify offsets (for the different views) of the text position from the default one
Source	<xs:element name="text" type="textType"/> <xs:complexType name="textType"> <xs:simpleContent> <xs:extension base="xs:string"> <xs:attribute name="UPDOWN" type="directionType" default="AUTO"/> <xs:attribute name="HIDDEN" type="boolType" default="FALSE"/> <xs:attributeGroup ref="dxdyAttributes"/> </xs:extension> </xs:simpleContent> </xs:complexType>		

8.3.30 <annotation>

The ANNOTATION element contains a textual annotation related to a note/rest/chord. The text is in the XML content.

Table 45 — ANNOTATION element syntax

Diagram			
Used by	<note> <rest> <chord> <anchorage>		
Attributes	**Name**	**Type**	**Description**
	UPDOWN	directionType	position of the annotation with respect to the note/rest/chord, can be UP, DOWN or AUTO
Attribute Group	dxdyAttributes		specify offsets (for the different views) of the annotation position from the default one
Source	<xs:element name="annotation" type="annotationType"/> <xs:complexType name="annotationType"> <xs:simpleContent> <xs:extension base="xs:string"> <xs:attribute name="UPDOWN" type="directionType" default="AUTO"/> <xs:attributeGroup ref="dxdyAttributes"/> </xs:extension> </xs:simpleContent> </xs:complexType>		

8.3.31 <glasses>

The GLASSES element () represents a “glasses” symbol (with the meaning of “please pay attention”) related to a note/rest/chord.

Table 46 — GLASSES element syntax

Diagram			
Used by	<note> <rest> <chord> <anchorage>		
Attributes	**Name**	**Type**	**Description**
	UPDOWN	directionType	position of the glasses with respect to the note/rest/chord, can be UP, DOWN or AUTO
Attribute Group	dxdyAttributes		specify offsets (for the different views) of the glasses position from the default one
Source	<xs:element name="glasses" type="glassesType"/> <xs:complexType name="glassesType"> <xs:attribute name="UPDOWN" type="directionType" default="AUTO"/> <xs:attributeGroup ref="dxdyAttributes"/> </xs:complexType>		

8.3.32 <fingering>

The FINGERING element represents a fingering indication associated with the note or with a note in a chord. Three kinds of fingering are possible, see Table 47.

Table 47 — FINGERING types

TYPES	EXAMPLE
NORMAL, the finger number is shown normally	
CIRCLED1, the finger number is shown within a circle	
CIRCLED2, the finger number is shown within a thicker circle	

Table 48 — FINGERING element syntax

Diagram	fingering		
Used by	<note> <chordnote>		
Attributes	**Name**	**Type**	**Description**
	FINGER	nonNegativeInteger	the number of the finger
	UPDOWN	directionType	position of the annotation with respect to the note, can be UP, DOWN or AUTO
	TYPE	enum	can be NORMAL, CIRCLED1 or CIRCLED2
Attribute Group	dxdyAttributes		specify offsets (for the different views) of the fingering position from the default one
Source	<xs:element name="fingering" type="fingeringType"/> <xs:complexType name="fingeringType"> <xs:attribute name="FINGER" type="xs:nonNegativeInteger" use="required"/> <xs:attribute name="UPDOWN" type="directionType" default="AUTO"/> <xs:attribute name="TYPE" default="NORMAL"> <xs:simpleType> <xs:restriction base="xs:NMTOKEN"> <xs:enumeration value="NORMAL"/> <xs:enumeration value="CIRCLED1"/> <xs:enumeration value="CIRCLED2"/> </xs:restriction> </xs:simpleType> </xs:attribute> <xs:attributeGroup ref="dxdyAttributes"/>		

```
    </xs:complexType>
```

8.3.33 <strings>

The STRING element represents a string indication related with a note or a chord. The fingering can be shown with Roman or Arabic numbers and additionally can be put in brackets (BRACKETS), put in a circle (CIRCLED1) or put in a thicker circle (CIRCLED2).

Table 49 — STRINGS types

TYPES	EXAMPLE
NORMAL, the string number is shown normally	
BRACKETS, the string number is shown with brackets	
CIRCLED1, the string number is shown within a circle	
CIRCLED2, the string number is shown within a thicker circle	

Table 50 — STRINGS element syntax

Diagram	strings		
Used by	<note> <chord>		
Attributes	**Name**	**Type**	**Description**
	STRING	nonNegativeInteger	the number of the string
	UPDOWN	directionType	position of the strings symbol with respect to the note, can be UP, DOWN or AUTO
	ROMAN	boolType	indicates if strings are to be displayed as Roman numbers (e.g. I, II, III, etc.) or as Arabic numbers (1, 2, 3, …)
	TYPE	enum	can be NORMAL, BRACKETS, CIRCLED1 or CIRCLED2
Attribute Group	dxdyAttributes		specify offsets (for the different views) of the strings symbol position from the default one

Source

```
   <xs:element name="strings" type="stringsType"/>
   <xs:complexType name="stringsType">
      <xs:attribute name="STRING" type="xs:nonNegativeInteger"
use="required"/>
      <xs:attribute name="UPDOWN" type="directionType" default="AUTO"/>
      <xs:attribute name="ROMAN" type="boolType" default="TRUE"/>
      <xs:attribute name="TYPE" default="NORMAL">
         <xs:simpleType>
            <xs:restriction base="xs:NMTOKEN">
               <xs:enumeration value="NORMAL"/>
               <xs:enumeration value="BRACKETS"/>
               <xs:enumeration value="CIRCLED1"/>
               <xs:enumeration value="CIRCLED2"/>
            </xs:restriction>
         </xs:simpleType>
      </xs:attribute>
      <xs:attributeGroup ref="dxdyAttributes"/>
    </xs:complexType>
```

8.3.34 <ornament>

The ORNAMENT element represents an ornamental symbol associated with a note/chord, normally ornaments are symbols placed abobe or below the note/chord. The following types of ornaments are supported:

Table 51 — ORNAMENT types

TYPE	EXAMPLE
Trill (TRILL)	
Turn (TURN)	
Back turn (TURNBACK)	
Up turn (TURNUP)	
Slashed turn (TURNSLASH)	
Mordent (MORDENT)	
Mordent Superior (MORDENTSUP)	

Double Mordent (DMORDENT)

Double Mordent superior (DMORDENTSUP)

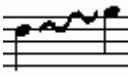
Tremolo (TREMOLO)

with an associated number of lines

Tremolo on stem (STEMTREMOLO)

with an associated number of lines

Glissando with a wavy line (GLISSWAVE)



Glissando with a straight line (GLISSLINE)

Table 52 — ORNAMENT element syntax

Diagram	ornament		
Used by	<note> <chord>		
Attributes	**Name**	**Type**	**Description**
	TYPE	enum	one of TRILL, TURN, TURNBACK, TURNUP, TURNSLASH, MORDENT, MORDENTSUP, DMORDENT, DMORDENTSUP, TREMOLO, STEMTREMOLO, GLISSWAVE or GLISSLINE
	NUMBER	nonNegativeInteger	number of bars for TREMOLO or STEMTREMOLO
	UPDOWN	directionType	position of the ornament symbol with respect to the note, can be UP, DOWN or AUTO, not applicable to TREMOLO, STEMTREMOLO, GLISSWAVE and GLISSLINE
	ACCSUP	accidentalTypesType	indicates the accidental to be drawn over the ornament symbol, not applicable to TREMOLO, STEMTREMOLO, GLISSWAVE and GLISSLINE
	ACCINF	accidentalTypesType	indicates the accidental to be drawn below the ornament symbol, not applicable to TREMOLO, STEMTREMOLO, GLISSWAVE and GLISSLINE
Attribute Group	dxdxAttributes		specify offsets (for the different views) of the ornament symbol position from the default one

Source

```
<xs:element name="ornament" type="ornamentType"/>
<xs:complexType name="ornamentType">
   <xs:attribute name="TYPE" use="required">
      <xs:simpleType>
         <xs:restriction base="xs:NMTOKEN">
            <xs:enumeration value="TRILL"/>
            <xs:enumeration value="TURN"/>
            <xs:enumeration value="TURNBACK"/>
            <xs:enumeration value="TURNUP"/>
            <xs:enumeration value="TURNSLASH"/>
            <xs:enumeration value="MORDENT"/>
            <xs:enumeration value="MORDENTSUP"/>
            <xs:enumeration value="DMORDENT"/>
            <xs:enumeration value="DMORDENTSUP"/>
            <xs:enumeration value="TREMOLO"/>
            <xs:enumeration value="STEMTREMOLO"/>
            <xs:enumeration value="GLISSWAVE"/>
            <xs:enumeration value="GLISSLINE"/>
         </xs:restriction>
      </xs:simpleType>
   </xs:attribute>
   <xs:attribute name="NUMBER" type="xs:nonNegativeInteger" default="1"/>
   <xs:attribute name="UPDOWN" type="directionType" default="AUTO"/>
   <xs:attribute name="ACCSUP" type="accidentalTypesType"/>
   <xs:attribute name="ACCINF" type="accidentalTypesType"/>
   <xs:attributeGroup ref="dxdyAttributes"/>
 </xs:complexType>
```

8.3.35 <korean-ornament>

The KOREAN-ORNAMENT element represents an ornament symbol used for Korean music notation associated with a note/chord, normally ornaments are symbols placed over or below the note/chord. The following types of ornaments are specified:

Table 53 — Unique ornaments for traditional Korean music notations (shigimsae)

Mark	Explanation	TYPE
	Sliding tone to higher	SLIDINGUP
	Sliding tone to lower	SLIDINGDOWN
	Two times Sliding tone to higher	SLIDINGDUP
	Sliding tone to lower and to higher	SLIDINGDOWNUP
   	Nonghyun (vibrato)	NONGHYNINC NONGHYNDEC NONGHYNNARROW NONGHYNWIDE

Table 54 — KOREAN-ORNAMENT element syntax

<table>
<tr><td>Diagram</td><td colspan="3">korean-ornament</td></tr>
<tr><td>Used by</td><td colspan="3"><note> <chord></td></tr>
<tr><td rowspan="3">Attributes</td><td>Name</td><td>Type</td><td>Description</td></tr>
<tr><td>TYPE</td><td>enum</td><td>can be one of: SLIDINGUP, SLIDINGDOWN, SLIDINGDUP, SLIDINGDOWNUP, NONGHYNINC, NONGHYNDEC, NONGHYNNARROW, NONGHYNWIDE</td></tr>
<tr><td>UPDOWN</td><td>directionType</td><td>position of the ornament symbol with respect to the note, can be UP, DOWN or AUTO,</td></tr>
<tr><td>Attribute Group</td><td colspan="2">dxdxAttributes</td><td>specify offsets (for the different views) of the ornament symbol position from the default one</td></tr>
<tr><td>Source</td><td colspan="3"><pre>
<xs:element name="Korean-ornament" type="ornamentType"/>
<xs:complexType name="ornamentType">
  <xs:attribute name="TYPE" use="required">
    <xs:simpleType>
      <xs:restriction base="xs:NMTOKEN">
        <xs:enumeration value="SLIDINGUP"/>
        <xs:enumeration value="SLIDINGDOWN"/>
        <xs:enumeration value="SLIDINGDUP"/>
        <xs:enumeration value="SLIDINGDOWN"/>
        <xs:enumeration value="NONGHYNINC"/>
        <xs:enumeration value="NONGHYNDEC"/>
        <xs:enumeration value="NONGHYNNARROW"/>
        <xs:enumeration value="NONGHYNWIDE"/>
      </xs:restriction>
    </xs:simpleType>
  </xs:attribute>
  <xs:attribute name="UPDOWN" type="directionType" default="AUTO"/>
  <xs:attributeGroup ref="dxdyAttributes"/>
</xs:complexType>
</pre></td></tr>
</table>

8.3.36 <marker>

The MARKER element represents a marking symbol associated with a note/chord, normally marker symbols are placed above or below the note/chord. The following types of markers are supported:

Table 55 — MARKER types

TYPE	EXAMPLE
Tenuto (TENUTO) o Portato (PORTATO),	
Sforzato (SFORZATO)	

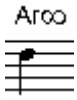
Accento (ACCENTO) o Accento forte (ACCENTOFORTE) 

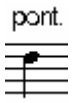
Staccato (STACCATO) o Punto sopra (PUNTOSOPRA) 

Martellato dolce (MARTDOLCE) 

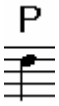
Punto allungato (PUNTOALLUNGATO) 

Martellato (MARTELLATO) 

Arco (ARCO) 

Ponticello (PONTICELLO) 

Tastiera (TASTIERA) 

Punta (PUNTA) 

Tallone (TALLONE) 

Bow up (BOWUP) 

Bow down (BOWDOWN) 

Pizzicato (PIZZ) 

generic marker (GENERIC) and in this case the NAME

attribute indicates the name identifying it.

Table 56 — MARKER element syntax

<table>
<tr><td>Diagram</td><td colspan="3">marker</td></tr>
<tr><td>Used by</td><td colspan="3"><note> <chord></td></tr>
<tr><td rowspan="4">Attributes</td><td>Name</td><td>Type</td><td>Description</td></tr>
<tr><td>TYPE</td><td>enum</td><td>the type of marker is one of the following: PORTATO, TENUTO, SFORZATO, ACCENTOFORTE, ACCENTO, PUNTOSOPRA, STACCATO, MARTDOLCE, PUNTOALLUNGATO, MARTELLATO, ARCO, PONTICELLO, TASTIERA, PUNTA, TALLONE, BOWUP, BOWDOWN, PIZZ or GENERIC</td></tr>
<tr><td>NAME</td><td>string</td><td>for a GENERIC marker indicates the name of marker</td></tr>
<tr><td>UPDOWN</td><td>direction Type</td><td>position of the marker symbol with respect to the note, can be UP, DOWN or AUTO</td></tr>
<tr><td>Attribute Group</td><td colspan="2">dxdyAttributes</td><td>specify offsets (for the different views) of the marker position from the default one</td></tr>
<tr><td>Source</td><td colspan="3">

```
<xs:element name="marker" type="markerType"/>
<xs:complexType name="markerType">
   <xs:attribute name="TYPE" use="required">
      <xs:simpleType>
         <xs:restriction base="xs:NMTOKEN">
            <xs:enumeration value="PORTATO"/>
            <xs:enumeration value="TENUTO"/>
            <xs:enumeration value="SFORZATO"/>
            <xs:enumeration value="ACCENTOFORTE"/>
            <xs:enumeration value="ACCENTO"/>
            <xs:enumeration value="PUNTOSOPRA"/>
            <xs:enumeration value="STACCATO"/>
            <xs:enumeration value="MARTDOLCE"/>
            <xs:enumeration value="PUNTOALLUNGATO"/>
            <xs:enumeration value="MARTELLATO"/>
            <xs:enumeration value="ARCO"/>
            <xs:enumeration value="PONTICELLO"/>
            <xs:enumeration value="TASTIERA"/>
            <xs:enumeration value="PUNTA"/>
            <xs:enumeration value="TALLONE"/>
            <xs:enumeration value="BOWUP"/>
            <xs:enumeration value="BOWDOWN"/>
            <xs:enumeration value="PIZZ"/>
            <xs:enumeration value="GENERIC"/>
         </xs:restriction>
      </xs:simpleType>
   </xs:attribute>
   <xs:attribute name="NAME" type="xs:string"/>
   <xs:attribute name="UPDOWN" type="directionType" default="AUTO"/>
   <xs:attributeGroup ref="dxdyAttributes"/>
</xs:complexType>
```

</td></tr>
</table>

8.3.37 <mute>

The MUTE element represents a mute symbol associated with a note/chord. Normally, mute symbols are placed above or below the note/chord. The following types of mute symbols are supported:

Table 57 — MUTE types

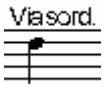
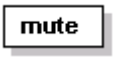
TYPES	EXAMPLE
IN	
VIA	
PLUS	
MINUS	
ON	
OFF	

Table 58 — MUTE element syntax

Diagram			
Used by	<note> <chord>		
Attributes	**Name**	**Type**	**Description**
	TYPE	enum	the type of mute, can be one of IN, VIA, PLUS, MINUS, ON, OFF
	UPDOWN	directionType	position of the mute symbol with respect to the note, can be UP, DOWN or AUTO
Attribute Group	dxdyAttributes		specify offsets (for the different views) of the mute position from the default one

Source	<xs:element name="mute" type="muteType"/> <xs:complexType name="muteType"> <xs:attribute name="TYPE" use="required"> <xs:simpleType> <xs:restriction base="xs:NMTOKEN"> <xs:enumeration value="IN"/> <xs:enumeration value="VIA"/> <xs:enumeration value="PLUS"/> <xs:enumeration value="MINUS"/> <xs:enumeration value="ON"/> <xs:enumeration value="OFF"/> </xs:restriction> </xs:simpleType> </xs:attribute> <xs:attribute name="UPDOWN" type="directionType" default="AUTO"/> <xs:attributeGroup ref="dxdyAttributes"/> </xs:complexType>

8.3.38 <harmonic>

The HARMONIC element represents a harmonic symbol associated with a note/chord. Normally, harmonic symbols are placed above or below the note/chord. The following types of harmonic symbols are supported:

Table 59 — HARMONIC types

TYPES	EXAMPLES
STRINGS	
FLUTES	

Table 60 — HARMONIC element syntax

Diagram	harmonic		
Used by	<note> <chord>		
Attributes	**Name**	**Type**	**Description**
	TYPE	enum	the type of harmonic, can be STRINGS or FLUTES
	UPDOWN	direction Type	position of the harmonic symbol with respect to the note, can be UP, DOWN or AUTO
Attribute Group	dxdyAttributes		specify offsets (for the different views) of the harmonic position from the default one
Source	<xs:element name="harmonic" type="harmonicType"/> <xs:complexType name="harmonicType">		

```
      <xs:attribute name="TYPE" use="required">
         <xs:simpleType>
            <xs:restriction base="xs:NMTOKEN">
               <xs:enumeration value="STRINGS"/>
               <xs:enumeration value="FLUTES"/>
            </xs:restriction>
         </xs:simpleType>
      </xs:attribute>
      <xs:attribute name="UPDOWN" type="directionType" default="AUTO"/>
      <xs:attributeGroup ref="dxdyAttributes"/>
   </xs:complexType>
```

8.3.39 <piano>

The PIANO element represents a piano pedal symbol associated with a note/chord. Normally piano pedal symbols are placed above or below the note/chord. The following types of piano pedal symbols are supported: Please note that for specifying piano pedal with start and stop with a dot connected with a line, a horizontal symbol has to be used.

Table 61 — PIANO Pedal types

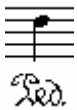
TYPES	EXAMPLES
PEDALDOWN	
PEDALUP	

Table 62 — PIANO element syntax

Diagram	piano		
Used by	<note> <chord> <rest>		
Attributes	**Name**	**Type**	**Description**
	TYPE	enum	the type of piano symbol, can be one of, PEDALDOWN, PEDALUP
	UPDOWN	direction Type	position of the piano pedal symbol with respect to the note, can be UP, DOWN or AUTO
Attribute Group	dxdyAttributes	specify offsets (for the different views) of the marker position from the default one	
Source	<xs:element name="piano" type="pianoType"/> <xs:complexType name="pianoType"> <xs:attribute name="TYPE" use="required"> <xs:simpleType> <xs:restriction base="xs:NMTOKEN">		

```
            <xs:enumeration value="PEDALDOWN"/>
            <xs:enumeration value="PEDALUP"/>
          </xs:restriction>
        </xs:simpleType>
      </xs:attribute>
      <xs:attribute name="UPDOWN" type="directionType" default="AUTO"/>
      <xs:attributeGroup ref="dxdyAttributes"/>
    </xs:complexType>
```

8.3.40 <fretboard>

The FRETBOARD element represents a fretboard symbol associated with a note/chord, normally fretboard symbols are placed above or below the note/chord.

Table 63 — FRETBOARD element syntax

Diagram			
Used by	<note> <chord>		
Attributes	**Name**	**Type**	**Description**
	NAME	string	name of the chord associated with the fretboard symbol
	NUMBER	nonNegativeInteger	number of instrument strings (e.g., 6 for guitar)
	FRETS	string	coding for frets to be pressed on strings, contains a sequence of characters one for each instrument string, each character can be x for non-played string, o for not pressed string, a number (1-9) indicating the fret to be pressed relatively to HEAD indication (e.g. with HEAD=5 1 means fret 6)
	FINGERS	string	coding for fingers pressing on string, contains a sequence of digits (1-7) one for each instrument string, each number indicates the finger, 7 for no finger
	BARRE	string	coding for barre indication, contains a couple of digits (1-NUMBER) indicating begin/end string of barre position, or it contains "00" in the case that the barre is not present
	HEAD	nonNegativeInteger	indicates the fret number from which the FRETS indicator starts.
	UPDOWN	directionType	position of the fretboard symbol with respect to the note, can be UP, DOWN or AUTO
Attribute Group	dxdyAttributes		specify offsets (for the different views) of the fretboard position from the default one
Source	<xs:element name="fretboard" type="fretboardType"/> <xs:complexType name="fretboardType"> <xs:attribute name="NAME" type="xs:string" use="required"/>		

```
     <xs:attribute name="NUMBER" type="xs:nonNegativeInteger"
use="required"/>
     <xs:attribute name="FRETS" type="xs:string" use="required"/>
     <xs:attribute name="FINGERS" type="xs:string" use="required"/>
     <xs:attribute name="BARRE" type="xs:string" use="required"/>
     <xs:attribute name="HEAD" type="xs:nonNegativeInteger" use="required"/>
     <xs:attribute name="UPDOWN" type="directionType" default="AUTO"/>
     <xs:attributeGroup ref="dxdyAttributes"/>
   </xs:complexType>
```

8.3.40.1 Example

```
<fretboard UPDOWN="DOWN" NUMBER="6" NAME="A7" FRETS="xo2223" FINGERS="771112" BARRE="35"
HEAD="0"/>
```

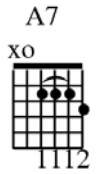


Figure 7 — Example of fretboard, A7

```
<fretboard UPDOWN="DOWN" NUMBER="6" NAME="C7" FRETS="xx1112" FINGERS="771112" BARRE="35"
HEAD="5"/>
```

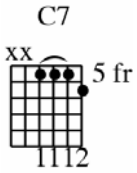


Figure 8 — Example of fretboard, C7

8.3.41 <rest>

The REST element represents a rest in a layer or in a beam. It contains:

— An optional AUGMENTATION element with the possible augmentation dots for the rest

— A sequence of zero or more FERMATA, DYNAMICTEXT, TEXT, ANNOTATION, GLASSES, PIANO and FRETBOARD elements representing symbols to be positioned above of below the rest,

— An optional USERSPACING element for user's modification to the space/distance to the next symbol in the layer.

The rest attributes contain information on the rest like:

— The ID uniquely identifying the rest within the layer,

— The DURATION of the rest (see Table 64), in cases where the duration is DGENERIC the MEASURES attribute indicates the number of measures of the rest duration. The number is typically written above the symbol by using a large font.

Table 64 — REST DURATIONS types

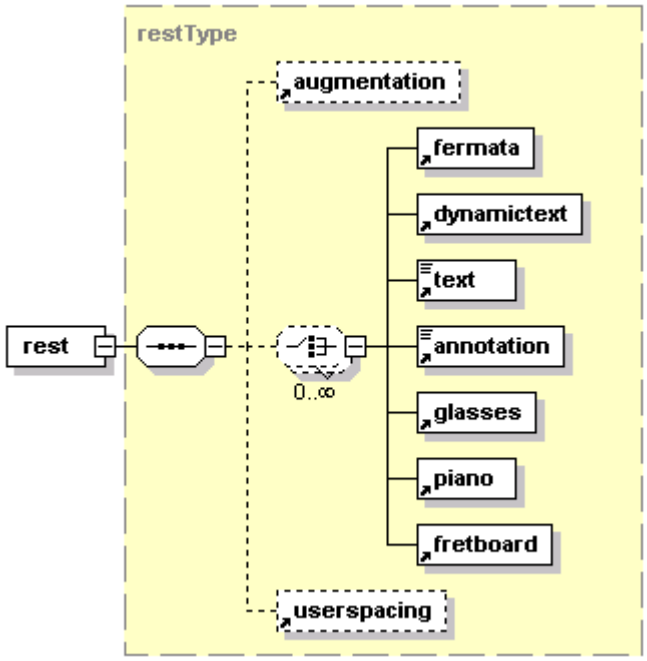
TYPES	EXAMPLE
D4M	
D2M, D2	
D1	
D1_2	
D1_4	
D1_8	
D1_16	
D1_32	
D1_64	
D1_128	
DGENERIC	

— The STAFF of the measure on which the rest has to be positioned, staff 0 is the upper one, staff 1 is the middle one or the lower one in case of two staves, staff 2 is the lower one in case of three staves.

— The HEIGHT is the position of the rest on the staff, 0 is the lower staff line, 1 is the first space, 2 is the second staff line (from bottom), 3 is the second space, etc.

— The STATUS attribute indicates if the rest is visible or not and if the rest has to have a visual duration or not (see Section 8.3.23 for a complete description).

Table 65 — REST element syntax

Diagram			
Children	<augmentation> <fermata> <dynamictext> <text> <annotation> <glasses> <piano> <fretboard> <userspacing>		
Used by	<layer> <beam>		
Attributes	**Name**	**Type**	**Description**
	ID	positiveInteger	a unique rest identifier within the layer
	DURATION	durationType	the duration of the rest (e.g. "D1_4" for a quarter rest, "D1_8" for a eight rest, etc.) (see subclause 8.2.1.4)
	MESAURES	nonNegativeInteger	number of measure of rest in case duration is DGENERIC
	SIZE	sizeType	the size of the rest (e.g. "SMALL" or "NORMAL")
	STATUS	statusType	the status of the rest ("NORMAL", "GRACED", "HIDDEN" or "GHOSTED") (see subclause 8.3.23)
	STAFF	nonNegativeInteger	number of staff within the measure where the rest has to be put, can be 0,1 or 2
	HEIGHT	integer	position of the rest on the staff
	LINK	string	a link associated with the note
	COLOR	colorType	the colour of the note

Source	<xs:element name="rest" type="restType"/> <xs:complexType name="restType"> <xs:sequence> <xs:element ref="augmentation" minOccurs="0"/> <xs:choice minOccurs="0" maxOccurs="unbounded"> <xs:element ref="fermata"/> <xs:element ref="dynamictext"/> <xs:element ref="text"/> <xs:element ref="annotation"/> <xs:element ref="glasses"/> <xs:element ref="piano"/> <xs:element ref="fretboard"/> </xs:choice> <xs:element ref="userspacing" minOccurs="0"/> </xs:sequence> <xs:attribute name="ID" type="xs:positiveInteger" use="required"/> <xs:attribute name="DURATION" type="durationType" use="required"/> <xs:attribute name="MEASURES" type="xs:nonNegativeInteger"/> <xs:attribute name="HEIGHT" type="xs:integer"/> <xs:attribute name="STAFF" type="xs:nonNegativeInteger" default="0"/> <xs:attribute name="SIZE" type="sizeType" default="NORMAL"/> <xs:attribute name="STATUS" type="statusType" default="NORMAL"/> <xs:attribute name="LINK" type="xs:string"/> <xs:attribute name="COLOR" type="colorType"/> </xs:complexType>

8.3.42 <anchorage>

The ANCHORAGE element represents an attachment point within the layer. It contains

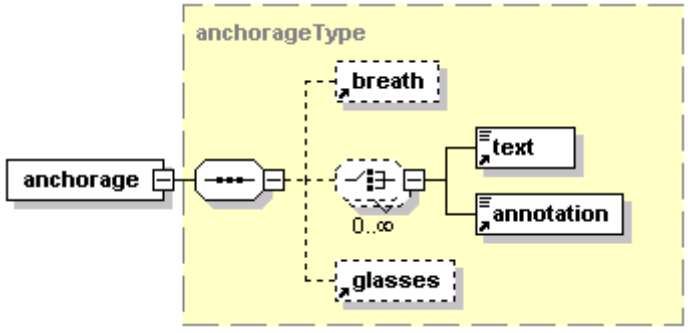
— An optional BREATH element for a breath mark

— A sequence of zero or more TEXT and ANNOTATION elements

— An optional GLASSES element for a glasses symbol

The visual position of the anchorage (SIZE attribute) is determined as a percentage of the space left on the right of the preceding note, chord or rest.

Table 66 — ANCHORAGE element syntax

Diagram			
Children	<breath> <text> <annotation> <glasses>		
Used by	<layer> <beam>		
Attributes	**Name**	**Type**	**Description**

	ID	positiveInteger	A number uniquely identifying the anchorage within the layer
	SIZE	integer	is the position of the anchorage as a percentage of the space left on the right of the preceding non-anchorage symbol
	STAFF	nonNegativeInteger	The staff where the anchorage is positioned
	LINK	string	a link associated with the anchorage
Source	<xs:element name="anchorage" type="anchorageType"/> <xs:complexType name="anchorageType"> <xs:sequence> <xs:element ref="breath" minOccurs="0"/> <xs:choice minOccurs="0" maxOccurs="unbounded"> <xs:element ref="text"/> <xs:element ref="annotation"/> </xs:choice> <xs:element ref="glasses" minOccurs="0"/> </xs:sequence> <xs:attribute name="ID" type="xs:positiveInteger" use="required"/> <xs:attribute name="SIZE" type="xs:integer"/> <xs:attribute name="STAFF" type="xs:nonNegativeInteger" default="0"/> <xs:attribute name="LINK" type="xs:string"/> </xs:complexType>		

8.3.43 <breath>

The BREATH element represents the breath sign (’).

Table 67 — BREATH element syntax

Diagram	breath	
Used by	<anchorage>	
Attribute Group	dxdyAttributes	specify offsets (for the different views) of the breath position from the default one
Source	<xs:element name="breath" type="breathType"/> <xs:complexType name="breathType"> <xs:attributeGroup ref="dxdyAttributes"/> </xs:complexType>	

8.3.44 <repetition>

The REPETITION element represents a repetition sign written in the layer of symbols. It contains an optional USERSPACING element for user’s changes to the default spacing. The following repetition signs are supported:

Table 68 — REPETITION types

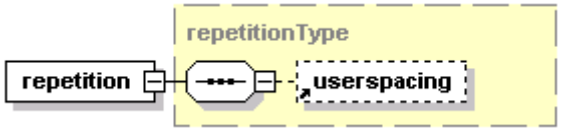
TYPES	EXAMPLES
MEASURE	

TWOMEASURE 

BEAT 

TWOBEAT 

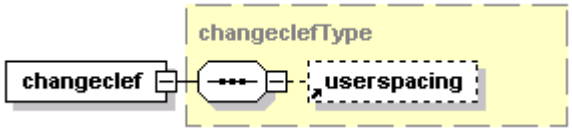
Table 69 — REPETITION element syntax

Diagram			
Children	<userspacing>		
Used by	<layer>		
Attributes	**Name**	**Type**	**Description**
	ID	positiveInteger	A number uniquely identifying the repetition within the layer
	TYPE	enum	The type of repetition, it can be MEASURE, TWOMEASURE, BEAT or TWOBEAT
	MEASURES	nonNegativeInteger	When type is MEASURE, it contains the number of measures to repeat
	STAFF	nonNegativeInteger	The staff where the repetition sign has to be positioned
Source	<xs:element name="repetition" type="repetitionType"/> <xs:complexType name="repetitionType"> <xs:sequence> <xs:element ref="userspacing" minOccurs="0"/> </xs:sequence> <xs:attribute name="ID" type="xs:nonNegativeInteger" use="required"/> <xs:attribute name="TYPE" use="required"> <xs:simpleType> <xs:restriction base="xs:NMTOKEN"> <xs:enumeration value="MEASURE"/> <xs:enumeration value="TWOMEASURE"/> <xs:enumeration value="BEAT"/> <xs:enumeration value="TWOBEAT"/> </xs:restriction> </xs:simpleType> </xs:attribute> <xs:attribute name="MEASURES" type="xs:nonNegativeInteger"/> <xs:attribute name="STAFF" type="xs:nonNegativeInteger" default="0"/> </xs:complexType>		

8.3.45 <changeclef>

The CHANGECLEF element represents a change of clef within the layer. It contains an optional USERSPACING element for user's modification to the default spacing.

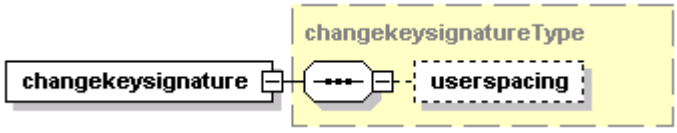
Table 70 — CHANGECLEF element syntax

Diagram			
Children	<userspacing>		
Used by	<layer> <beam>		
Attributes	**Name**	**Type**	**Description**
	ID	positiveInteger	A number uniquely identifying the clef change within the layer
	TYPE	clefTypesType	The clef type (see subclause 8.3.16 for supported clef types)
	STAFF	nonNegativeInteger	The staff where the clef change is positioned (0,1 or 2)
	SIZE	sizeType	The size of clef symbol, can be SMALL or NORMAL
	LINK	string	a link associated with the clef change
Source	<xs:element name="changeclef" type="changeclefType"/> <xs:complexType name="changeclefType"> <xs:sequence> <xs:element ref="userspacing" minOccurs="0"/> </xs:sequence> <xs:attribute name="ID" type="xs:positiveInteger" use="required"/> <xs:attribute name="TYPE" type="clefTypesType" use="required"/> <xs:attribute name="STAFF" type="xs:nonNegativeInteger" default="0"/> <xs:attribute name="SIZE" type="sizeType" default="SMALL"/> <xs:attribute name="LINK" type="xs:string"/> </xs:complexType>		

8.3.46 <changekeysignature>

The CHANGEKEYSIGNATURE element represents a change of keysignature within the layer. It contains an optional USERSPACING element for user's modification to the default spacing.

Table 71 — CHANGEKEYSIGNATURE element syntax

Diagram	

Children	<userspacing>		
Used by	<layer>		
Attributes	**Name**	**Type**	**Description**
	ID	positiveInteger	A number uniquely identifying the change in the layer
	TYPE	keysignatureTypesType	The keysignature type (see 8.3.17 for supported keysignatures)
	STAFF	nonNegativeInteger	The staff where the keysignature change is positioned (0,1 or 2)
	LINK	string	a link associated with the keysignature change
Source	<xs:element name="changekeysignature" type="changekeysignatureType"/> <xs:complexType name="changekeysignatureType"> <xs:sequence> <xs:element ref="userspacing" minOccurs="0"/> </xs:sequence> <xs:attribute name="ID" type="xs:positiveInteger" use="required"/> <xs:attribute name="TYPE" type="keysignatureTypesType" use="required"/> <xs:attribute name="STAFF" type="xs:nonNegativeInteger" default="0"/> <xs:attribute name="LINK" type="xs:string"/> </xs:complexType>		

8.3.47 <beam>

The BEAM element represents a sequence of beamed notes. It contains a sequence of more than 2 NOTE, CHORD, REST, ANCHORAGE and CHANGECLEF elements. The note and chords must have a duration which is less than or equal to 1/8 ("D1_8"). For beams crossing barlines, please see the horizontal symbols.

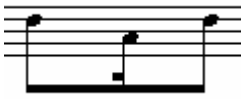
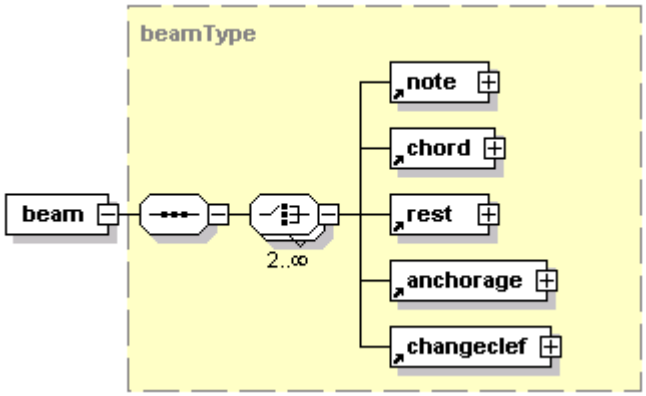


Figure 9 — Example of beam

Table 72 — BEAM element syntax

<table>
<tr><td>Diagram</td><td colspan="3"> 
</td></tr>
<tr><td>Children</td><td colspan="3"><note> <chord> <rest> <anchorage> <changeclef></td></tr>
<tr><td>Used by</td><td colspan="3"><layer></td></tr>
<tr><td rowspan="5">Attributes</td><td>Name</td><td>Type</td><td>Description</td></tr>
<tr><td>ID</td><td>positiveInteger</td><td>a number uniquely identifying the beam in the layer</td></tr>
<tr><td>STEMS</td><td>enum</td><td>indicates the stem direction, it can be UP, DOWN, STAFF12, STAFF23 and AUTO. STAFF12 and STAFF23 applies to beam having notes on different staves, STAFF12 indicates that the beam has to be positioned between staff 0 and 1 and STAFF23 between staff 1 and 2</td></tr>
<tr><td>STATUS</td><td>statusType</td><td>the status of all the notes/chords/rests contained in the beam</td></tr>
<tr><td>SIZE</td><td>sizeType</td><td>the visual size of the beam, it can be SMALL or NORMAL, it applies to all the elements inside the beam</td></tr>
<tr><td>Source</td><td colspan="3"><pre>
<xs:element name="beam" type="beamType"/>
<xs:complexType name="beamType">
  <xs:sequence>
    <xs:choice minOccurs="2" maxOccurs="unbounded">
      <xs:element ref="note"/>
      <xs:element ref="chord"/>
      <xs:element ref="rest"/>
      <xs:element ref="anchorage"/>
    </xs:choice>
  </xs:sequence>
  <xs:attribute name="ID" type="xs:positiveInteger" use="required"/>
  <xs:attribute name="STEMS" default="AUTO">
    <xs:simpleType>
      <xs:restriction base="xs:NMTOKEN">
        <xs:enumeration value="UP"/>
        <xs:enumeration value="DOWN"/>
        <xs:enumeration value="STAFF12"/>
        <xs:enumeration value="STAFF23"/>
        <xs:enumeration value="AUTO"/>
      </xs:restriction>
    </xs:simpleType>
  </xs:attribute>
  <xs:attribute name="STATUS" type="statusType" default="NORMAL"/>
  <xs:attribute name="SIZE" type="sizeType" default="NORMAL"/>
</xs:complexType>
</pre></td></tr>
</table>

8.3.48 <chord>

The CHORD element represents a chord symbol. It contains:

— A sequence of more than 2 CHORDNOTE elements representing the noteheads of the chord.

— An optional ARPEGGIO element for the arpeggio symbol.

— An optional AUGMENTATION element with the possible augmentation dots for all the note heads

— A sequence of zero or more FERMATA, DYNAMICTEXT, TEXT, ANNOTATION, GLASSES, FINGERING, STRINGS, ORNAMENT, KOREAN-ORNAMENT, MARKER, MUTE, HARMONIC, PIANO and FRETBOARD elements representing symbols to be positioned above or below the chord,

— A USERSPACING element for user's modification of the distance to the following symbol in the layer.

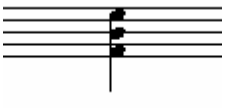
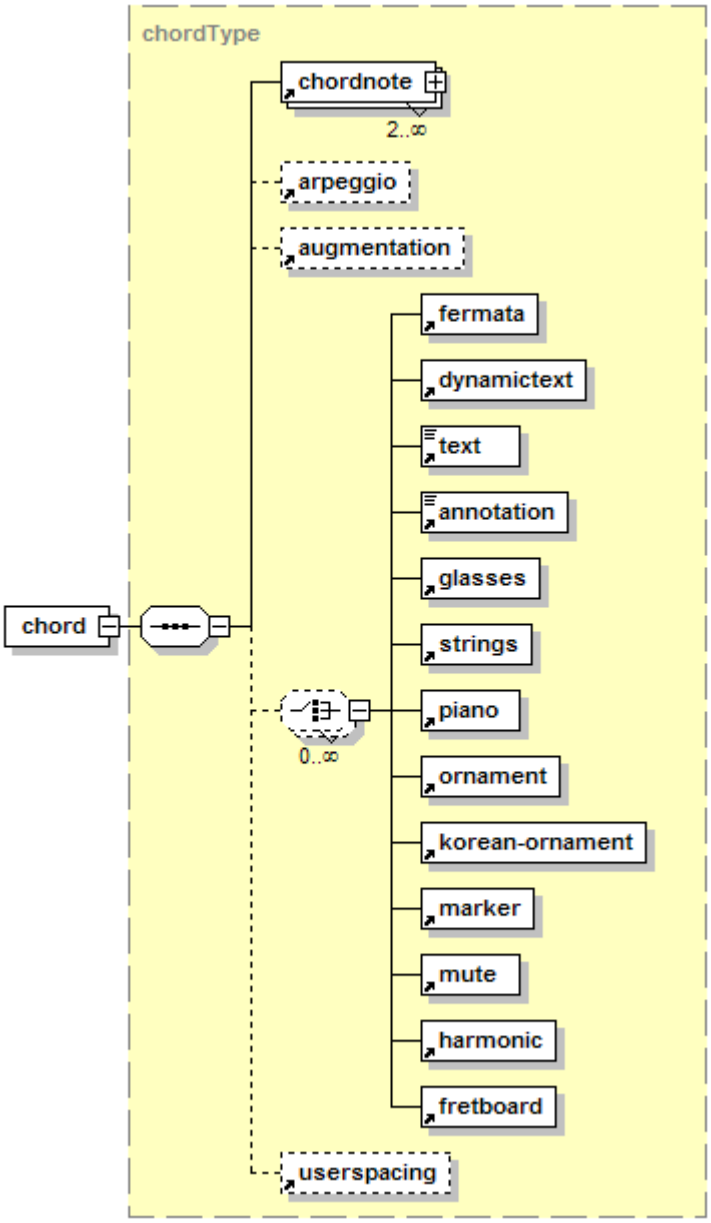


Figure 10 — Example of chord

Table 73 — CHORD element syntax

<table>
<tr><td>Diagram</td><td colspan="3"> 
</td></tr>
<tr><td>Children</td><td colspan="3"><chordnote> <arpeggio> <augmentation> <fermata> <dynamictext> <text> <annotation> <glasses> <strings> <piano> <ornament> <korean-ornament> <marker> <mute> <harmonic> <fretboard> <usespacing></td></tr>
<tr><td>Used by</td><td colspan="3"><layer> <beam></td></tr>
<tr><td rowspan="4">Attributes</td><td>Name</td><td>Type</td><td>Description</td></tr>
<tr><td>ID</td><td>positiveInteger</td><td>A number uniquely identifying the chord within the layer</td></tr>
<tr><td>DURATION</td><td>durationType</td><td>The duration of the chord</td></tr>
<tr><td>STEM</td><td>directionType</td><td>Direction of the chord stem, can be UP, DOWN or AUTO</td></tr>
</table>

	ACCIACCATURA	boolType	Indicates that the chord is an acciaccatura, possible only when DURATION is "D1_8" (an 8th note)
	SIZE	sizeType	The size of the chord (SMALL or NORMAL)
	STATUS	statusType	Indicates the chord status exactly as for notes (see 8.3.23)
	LINK	string	A link associated with the chord
	FINGERING	directionType	Indicates where to draw the fingering indications of the chord's notes (UP, DOWN or AUTO)

Source

```
<xs:element name="chord" type="chordType"/>
<xs:complexType name="chordType">
   <xs:sequence>
      <xs:element ref="chordnote" minOccurs="2" maxOccurs="unbounded"/>
      <xs:element ref="arpeggio" minOccurs="0"/>
      <xs:element ref="augmentation" minOccurs="0"/>
      <xs:choice minOccurs="0" maxOccurs="unbounded">
         <xs:element ref="fermata"/>
         <xs:element ref="dynamictext"/>
         <xs:element ref="text"/>
         <xs:element ref="annotation"/>
         <xs:element ref="glasses"/>
         <xs:element ref="strings"/>
         <xs:element ref="piano"/>
         <xs:element ref="ornament"/>
         <xs:element ref="korean-ornament"/>
         <xs:element ref="marker"/>
         <xs:element ref="mute"/>
         <xs:element ref="harmonic"/>
         <xs:element ref="fretboard"/>
      </xs:choice>
      <xs:element ref="userspacing" minOccurs="0"/>
   </xs:sequence>
   <xs:attribute name="ID" type="xs:positiveInteger" use="required"/>
   <xs:attribute name="DURATION" type="durationType" use="required"/>
   <xs:attribute name="STEM" type="directionType" default="AUTO"/>
   <xs:attribute name="ACCIACCATURA" type="boolType" default="FALSE"/>
   <xs:attribute name="SIZE" type="sizeType" default="NORMAL"/>
   <xs:attribute name="STATUS" type="statusType" default="NORMAL"/>
   <xs:attribute name="LINK" type="xs:string"/>
   <xs:attribute name="FINGERING" type="directionType" default="AUTO"/>
</xs:complexType>
```

8.3.49 <chordnote>

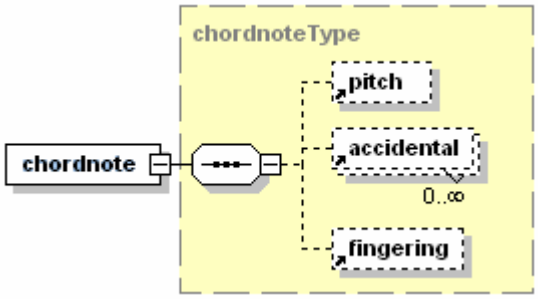
A CHORDNOTE element represents a notehead within a chord. It contains:

— An optional PITCH element containing the musical pitch of the notehead.

— Zero or more ACCIDENTAL elements for the accidentals related to the notehead.

— An optional FINGERING element containing the fingering indication for the specific notehead.

Table 74 — CHORDNOTE element syntax

<table>
<tr><td>Diagram</td><td colspan="3"> 
</td></tr>
<tr><td>Children</td><td colspan="3"><pitch> <accidental> <fingering></td></tr>
<tr><td>Used by</td><td colspan="3"><chord></td></tr>
<tr><td rowspan="7">Attributes</td><td>Name</td><td>Type</td><td>Description</td></tr>
<tr><td>ID</td><td>positiveInteger</td><td>A number uniquely identifying the chord note within the chord</td></tr>
<tr><td>STAFF</td><td>nonNegativeInteger</td><td>The staff number the note belongs to (can be 0,1 or 2)</td></tr>
<tr><td>HEIGHT</td><td>integer</td><td>The height of the note with respect to the staff (see 8.3.23 for details)</td></tr>
<tr><td>HEAD</td><td>noteHeadTypesType</td><td>The type of note head to be used (e.g., "CLASSIC", "X", etc.) (see subclause 8.2.1.11 for the complete list)</td></tr>
<tr><td>HEAD_ALPHANUM</td><td>string</td><td>when HEAD is "ALPHANUM" or "ALPHANUM_SQUARE" or "ALPHANUM_REVERSE" it contains the letter/number to be written in the note head.</td></tr>
<tr><td>COLOR</td><td>colorType</td><td>The color to be used for the note head</td></tr>
<tr><td>Source</td><td colspan="3"><pre>
<xs:element name="chordnote" type="chordnoteType"/>
<xs:complexType name="chordnoteType">
  <xs:sequence>
    <xs:element ref="pitch" minOccurs="0"/>
    <xs:element ref="accidental" minOccurs="0" maxOccurs="unbounded"/>
    <xs:element ref="fingering" minOccurs="0"/>
  </xs:sequence>
  <xs:attribute name="ID" type="xs:positiveInteger" use="required"/>
  <xs:attribute name="STAFF" type="xs:nonNegativeInteger" default="0"/>
  <xs:attribute name="HEIGHT" type="xs:integer" use="required"/>
  <xs:attribute name="HEAD" type="noteHeadTypesType" default="CLASSIC"/>
  <xs:attribute name="HEAD_ALPHANUM" type="xs:string"/>
  <xs:attribute name="COLOR" type="colorType"/>
</xs:complexType>
</pre></td></tr>
</table>

8.3.50 <arpeggio>

An ARPEGGIO element is used to indicate an arpeggio symbol to be associated with the chord. When the arpeggio is associated with chords placed on differen staffs (for example in horgans) the arpeggio is drawn crossing all of them.

Table 75 — ARPEGGIO types

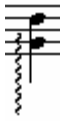
TYPES	EXAMPLE
UP, a wavy vertical line with an arrow head on top	
DOWN, a wavy vertical line with an arrow head on bottom	
NONE, a wavy vertical line without an arrow head	

Table 76 — ARPEGGIO element syntax

Diagram	arpeggio		
Used by	<chord>		
Attributes	**Name**	**Type**	**Description**
	DIRECTION	enum	Indicates the direction of the arpeggio, it can be UP, DOWN or NONE, this should not be confused with directionType
Attributes Group	dxdyAttributes		specify offsets (for the different views) of the arpeggio position from the default one
Source	<xs:element name="arpeggio" type="arpeggioType"/> <xs:complexType name="arpeggioType"> <xs:attribute name="DIRECTION" default="NONE"> <xs:simpleType> <xs:restriction base="xs:NMTOKEN"> <xs:enumeration value="UP"/> <xs:enumeration value="DOWN"/> <xs:enumeration value="NONE"/> </xs:restriction> </xs:simpleType> </xs:attribute> <xs:attributeGroup ref="dxdyAttributes"/> </xs:complexType>		

8.3.51 <barline>

A BARLINE element represents a barline symbol used to indicate the end of a measure. The types of barlines supported are:

Table 77 — BARLINE types

TYPES	EXAMPLE
SINGLE, a single vertical barline joining all the staff lines	
DOUBLE, a double barline usually indicating a change of clef or keysignature	
START_REFRAIN, a barline indicating the the following measure is the start of a refrain	
END_REFRAIN, a barline indicating that the current measure is the end of a refrain	
START_END_REFRAIN, a barline indicating that the current measure is the end of a refrain and the following measure is a start of another refrain	
END, a barline indicating the end of the score	
INVISIBLE, a non visible barline useful for visually representing unmeasured music.	

Table 78 — BARLINE element syntax

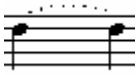
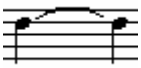
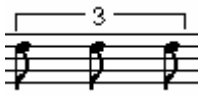
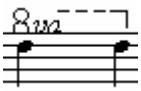
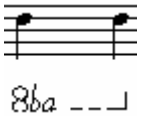
Diagram	barline		
Used by	<measure>		
Attributes	**Name**	**Type**	**Description**
	TYPE	enum	Indicates the barline type, it can be (SINGLE, DOUBLE, START_REFRAIN, END_REFRAIN, START_END_REFRAIN, END or INVISIBLE)
	LINETYPE	enum	Indicates the type of line to be used for drawing SINGLE barlines, it can be SOLID, DOT or DASH.
	REPEAT	boolType	Indicates that a repeat measure symbol has to be drawn on the barline
	FERMATA	boolType	Indicates that a fermata symbol has to be drawn on the barline
Source	<xs:element name="barline" type="barlineType"/> <xs:complexType name="barlineType"> <xs:attribute name="TYPE" use="required"> <xs:simpleType> <xs:restriction base="xs:NMTOKEN">		

```
            <xs:enumeration value="SINGLE"/>
            <xs:enumeration value="DOUBLE"/>
            <xs:enumeration value="START_REFRAIN"/>
            <xs:enumeration value="END_REFRAIN"/>
            <xs:enumeration value="START_END_REFRAIN"/>
            <xs:enumeration value="END"/>
            <xs:enumeration value="INVISIBLE"/>
         </xs:restriction>
      </xs:simpleType>
   </xs:attribute>
   <xs:attribute name="LINETYPE" default="SOLID">
      <xs:simpleType>
         <xs:restriction base="xs:NMTOKEN">
            <xs:enumeration value="SOLID"/>
            <xs:enumeration value="DOT"/>
            <xs:enumeration value="DASH"/>
         </xs:restriction>
      </xs:simpleType>
   </xs:attribute>
   <xs:attribute name="REPEAT" type="boolType" default="FALSE"/>
   <xs:attribute name="FERMATA" type="boolType" default="FALSE"/>
</xs:complexType>
```

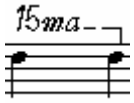
8.3.52 <horizontal>

A HORIZONTAL element represents any symbol spanning over many musical figures such as slurs, crescendo/diminuendo symbols, tuples, pedal indications, octave changes etc. It contains two ADDRESS elements indicating the start and end of the symbol. The types of horizontal symbols supported are:

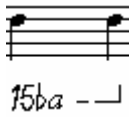
Table 79 — HORIZONTAL types

TYPES	EXAMPLE
SLUR, for a slur beginning on a figure and ending on another one, it uses the KNOT1 and KNOT2 attributes to control the slur bowing	
SLURDOT, SLURDASH for dotted and dashed slurs	
TIE, for a tie symbol connecting two notes with the same height	
TUPLE, for tuples, it uses the TUPLELINE and TUPLENUMBER attributes to control the tuple appearance (with or without line) and for the number associated with the tuple (e.g. 3 for a triplet)	
OCTVA, for an upper octave change for the notes it spans across	
OCTBA, for a lower octave change for the notes it spans across	

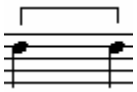
QUINDMA, for a upper quindicesima change for notes it spans across



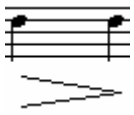
QUINDBA, for a lower quindicesima change for notes it spans across



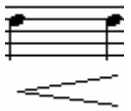
BEND, BENDDOT, BENDDASH, for a normal, dotted or dashed bend over notes



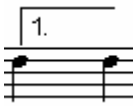
DIMINUENDO, DIMDOT, DIMDASH for a normal, dotted or dashed diminuendo sign spanning on a sequence of musical symbols



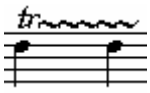
CRESCENDO, CREDOT, CREDASH for a normal, dotted or dashed crescendo sign spanning on a sequence of musical symbols



CHANGEREF, for an indication of figures to be played in different refrains, it uses the REFNUMBER attribute to indicate the number of refrain in which the figures have to be played

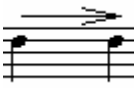


TRILL, for a trill sign (tr) extending with a ripple line to the next figure

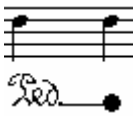


WAVE, for a wavy line above or below a sequence of musical figures

ARROW, for an arrow above or below a sequence of musical symbols



PIANOPEDAL, for a pedal indication marking the figure where the pedal has to be pressed and the ending figure where the pedal has to be released



BEAM. For specifying a beam crossing barlines

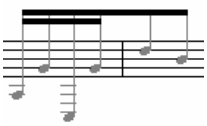
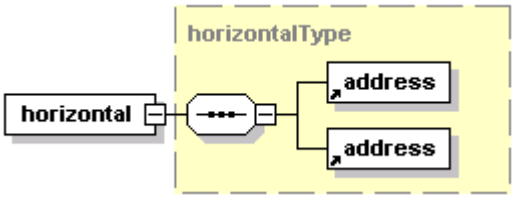


Table 80 — HORIZONTAL element syntax

Diagram			
Children	<address>		
Used by	<score>		
Attributes	**Name**	**Type**	**Description**
	ID	positiveInteger	A number uniquely identifying the horizontal symbol within the score
	TYPE	enum	Indicates the type of horizontal symbol
	UPDOWN	directionType	Indicates if the symbol has to be drawn above (UP) or below (DOWN) the musical figures or to be automatically determined (AUTO)
	LINK	string	A link associated with the musical symbol
	KNOT1	decimal	Indicates the behaviour of the slur at the starting point (only for type SLUR, SLURDOT, SLURDASH) as described in the following.
	KNOT2	decimal	Indicates the behaviour of the slur at the ending point (only for type SLUR, SLURDOT, SLURDASH) as described in the following.
	TUPLENUMBER	nonNegativeInteger	Indicates the number associated with the tuple (only for type TUPLE)
	TUPLELINE	boolType	Indicates if a line marking the tuple has to be drawn or not (only for type TUPLE)
	REFNUMBER	nonNegativeInteger	The number of refrain where the notes have to be played (only for type CHANGEREF) (e.g. 1 means that the notes has to be played only the first time)
	COLOR	colorType	the colour of the horizontal
	BEAM.STEMS	enumeration	Can be one of (UP, DOWN, STAFF12, STAFF23, AUTO) indicating the position of beam bar. (only for type BEAM)
	BEAM.STATUS	statusType	Indicates the beam status exactly as for notes (see 8.3.23) (only for type BEAM)
	BEAM.SIZE	sizeType	Can be one of (NORMAL, SMALL) indicating the size of beam. (only for type BEAM)

Attribute Groups	dxdyAttributes	specify offsets (for the different views) of the starting point position from the default one
	dxdyrAttributes	specify offsets (for the different views) of the ending point position from the default one
Source		

```
   <xs:element name="horizontal" type="horizontalType"/>
   <xs:complexType name="horizontalType">
      <xs:sequence>
         <xs:element ref="address"/>
         <xs:element ref="address"/>
      </xs:sequence>
      <xs:attribute name="ID" type="xs:positiveInteger" use="required"/>
      <xs:attribute name="TYPE" use="required">
         <xs:simpleType>
            <xs:restriction base="xs:NMTOKEN">
               <xs:enumeration value="SLUR"/>
               <xs:enumeration value="SLURDOT"/>
               <xs:enumeration value="SLURDASH"/>
               <xs:enumeration value="TIE"/>
               <xs:enumeration value="TUPLE"/>
               <xs:enumeration value="OCTVA"/>
               <xs:enumeration value="OCTBA"/>
               <xs:enumeration value="QUINDMA"/>
               <xs:enumeration value="QUINDBA"/>
               <xs:enumeration value="BEND"/>
               <xs:enumeration value="BENDDOT"/>
               <xs:enumeration value="BENDDASH"/>
               <xs:enumeration value="DIMINUENDO"/>
               <xs:enumeration value="DIMDOT"/>
               <xs:enumeration value="DIMDASH"/>
               <xs:enumeration value="CRESCENDO"/>
               <xs:enumeration value="CREDOT"/>
               <xs:enumeration value="CREDASH"/>
               <xs:enumeration value="CHANGEREF"/>
               <xs:enumeration value="TRILL"/>
               <xs:enumeration value="WAVE"/>
               <xs:enumeration value="ARROW"/>
               <xs:enumeration value="PIANOPEDAL"/>
               <xs:enumeration value="BEAM"/>
            </xs:restriction>
         </xs:simpleType>
      </xs:attribute>
      <xs:attribute name="UPDOWN" type="directionType" default="AUTO"/>
      <xs:attribute name="LINK" type="xs:string"/>
      <xs:attribute name="KNOT1" type="xs:decimal"/>
      <xs:attribute name="KNOT2" type="xs:decimal"/>
      <xs:attribute name="TUPLELINE" type="boolType" default="TRUE"/>
      <xs:attribute name="TUPLENUMBER" type="xs:nonNegativeInteger"
default="3"/>
      <xs:attribute name="REFNUMBER" type="xs:nonNegativeInteger" default="1"/>
      <xs:attribute name="BEAM.STEMS" default="AUTO">
        <xs:simpleType>
            <xs:restriction base="xs:NMTOKEN">
                <xs:enumeration value="UP"/>
                <xs:enumeration value="DOWN"/>
                <xs:enumeration value="STAFF12"/>
                <xs:enumeration value="STAFF23"/>
                <xs:enumeration value="AUTO"/>
            </xs:restriction>
        </xs:simpleType>
      </xs:attribute>
      <xs:attribute name="BEAM.STATUS" type="statusType" default="NORMAL" />
      <xs:attribute name="COLOR" type="colorType" />
       <xs:attribute name="BEAM.SIZE" type="sizeType" default=" NORMAL " />
      <xs:attributeGroup ref="dxdyAttributes"/>
```

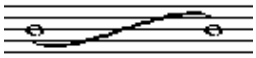
```
      <xs:attributeGroup ref="dxdyrAttributes"/>
    </xs:complexType>
```

The slur direction and shape can be changed by altering the UPDOWN value, when it is up the slur is shaped as depicted in the following figure:

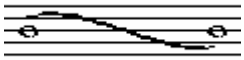
Knot1, Knot2 may have a decimal value from 100.00 to -100.00.

The default value is 1.0 for both Knot1 and Knot2 when the slur is created. If the value of -1.0, -1.0 is imposed, this change has the same effect as changing the UPDOWN value from up to down or vice versa depending on the initial value. In this way, changing the signature of one of those parameters the following results are obtained.

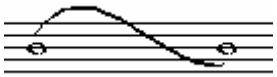
Case in which Knot1=-1.0 and knot2=1.0 with UPDOWN=UP



Case in which Knot1=1.0 and knot2=-1.0 with UPDOWN=UP



The values of Knot1, and Knot2 can be increased to change the shape of the curvature. For example, with knot1=2,5 and Knot2=-1,3:



The same solution and values can be used for defining the behaviour of the slur that connects two notes placed on different staffs of a piano part. The curvature of the slur may be calculated by using Bspline model.

8.3.53 <address>

An ADDRESS element specifies an address to a musical figure inside the score.

Table 81 — ADDRESS element syntax

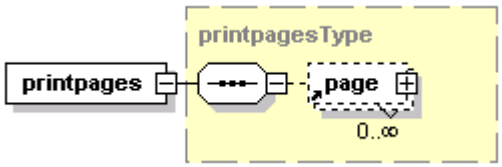
Diagram	address		
Used by	<horizontal>		
Attributes	**Name**	**Type**	**Description**
	MEASURE	positiveINteger	The ID of the measure that the figure is inside

	LAYER	positiveInteger	The number of the layer (in the measure) where the figure is
	FIGURE	positiveInteger	The ID of the figure, if the figure is inside a beam or a chord, this is the id of the beam or of the chord
	CHORD.OR.BEAM	nonNegativeInteger	the id of the figure inside the beam or chord only in the case where the FIGURE attribute refers to a beam or chord
	CHORD.IN.BEAM	nonNegativeInteger	The id of the note inside a chord inside a beam
Source	<xs:element name="address" type="addressType"/> <xs:complexType name="addressType"> <xs:attribute name="MEASURE" type="xs:positiveInteger" use="required"/> <xs:attribute name="LAYER" type="xs:positiveInteger" use="required"/> <xs:attribute name="FIGURE" type="xs:positiveInteger" use="required"/> <xs:attribute name="CHORD.OR.BEAM" type="xs:nonNegativeInteger" use="required"/> <xs:attribute name="CHORD.IN.BEAM" type="xs:nonNegativeInteger" use="required"/> </xs:complexType>		

8.3.54 <printpages>

A PRINTPAGES element contains additional information for the print view. It contains a sequence of zero or more PAGE element with the boxes (text or images) to be shown in the specific pages.

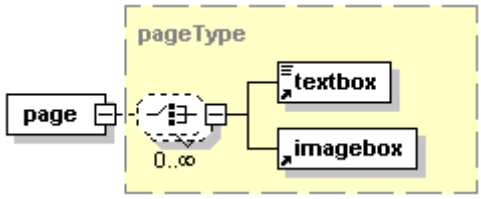
Table 82 — PRINTPAGES element syntax

Diagram	
Children	<page>
Used by	<SWF_Part>
Source	<xs:element name="printpages" type="printpagesType"/> <xs:complexType name="printpagesType"> <xs:sequence> <xs:element ref="page" minOccurs="0" maxOccurs="unbounded"/> </xs:sequence> </xs:complexType>

8.3.55 <page>

A PAGE element contains the TEXTBOX and IMAGEBOX elements to be shown in the page to be printed.

Table 83 — PAGE element syntax

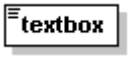
Diagram			
Children	<textbox> <imagebox>		
Used by	<printpages>		
Attributes	**Name**	**Type**	**Description**
	NUMBER	nonNegativeNumber	The number of the page where the boxes are to be shown

Source	`<xs:element name="page" type="pageType"/>`
	`<xs:complexType name="pageType">`
	`<xs:choice minOccurs="0" maxOccurs="unbounded">`
	`<xs:element ref="textbox"/>`
	`<xs:element ref="imagebox"/>`
	`</xs:choice>`
	`<xs:attribute name="NUMBER" type="xs:nonNegativeInteger" use="required"/>`
	`</xs:complexType>`		

8.3.56 <textbox>

A TEXTBOX element contains a text to be shown in the page. The text is in the XML content.

Table 84 — TEXTBOX element syntax

Diagram			
Used by	<page>		
Attributes	**Name**	**Type**	**Description**
	X	decimal	X (horizontal) position of the text on the page
	Y	decimal	Y (vertical) position of the text on the page (origin is in the upper left corner)
	ALIGN	enum	The alignment to be used for the text with respect to the (X,Y) point, it can be LEFT, CENTER or RIGHT
	FONTTYPE	string	The name of the font to be used for text
	FONTSTYLE	string	The style of the font to be used

	FONTSIZE	integer	The font size to be used measured in points (e.g., 20)
Source	<xs:element name="textbox" type="textboxType"/> <xs:complexType name="textboxType"> <xs:simpleContent> <xs:extension base="xs:string"> <xs:attribute name="X" type="xs:decimal" use="required"/> <xs:attribute name="Y" type="xs:decimal" use="required"/> <xs:attribute name="ALIGN" default="LEFT"> <xs:simpleType> <xs:restriction base="xs:NMTOKEN"> <xs:enumeration value="LEFT"/> <xs:enumeration value="CENTER"/> <xs:enumeration value="RIGHT"/> </xs:restriction> </xs:simpleType> </xs:attribute> <xs:attribute name="FONTTYPE" type="xs:string" default="DEFAULT"/> <xs:attribute name="FONTSTYLE" type="xs:string" default="NORMAL_NORMAL"/> <xs:attribute name="FONTSIZE" type="xs:integer" default="10"/> </xs:extension> </xs:simpleContent> </xs:complexType>		

8.3.57 <imagebox>

An IMAGEBOX element contains an image to be shown in the page.

Table 85 — IMAGEBOX element syntax

Diagram			
Used by	<page>		
Attributes	**Name**	**Type**	**Description**
	X	decimal	X (horizontal) position of the image on the page
	Y	decimal	Y (vertical) position of the image on the page, origin is in the upper left corner
	W	nonNegativeInteger	Width of the image box
	H	nonNegativeInteger	Height of the image box
	IMAGEFILE	string	Name of the file containing the image
Source	<xs:element name="imagebox" type="imageboxType"/> <xs:complexType name="imageboxType"> <xs:attribute name="X" type="xs:decimal" use="required"/> <xs:attribute name="Y" type="xs:decimal" use="required"/> <xs:attribute name="W" type="xs:nonNegativeInteger" use="required"/> <xs:attribute name="H" type="xs:nonNegativeInteger" use="required"/> <xs:attribute name="IMAGEFILE" type="xs:string" use="required"/> </xs:complexType>		

8.4 Main Score (SMXF_Main)

8.4.1 <SMXF_Main>

The SMXF_MAIN score contains information on the main score. It contains:

— An optional IDENTIFICATION element containing identification information (e.g. ISMN, ISRC, etc.),

— An optional sequence of zero or more CLASSIFICATION elements with multilingual metadata about the main score (e.g. title, genre, author, etc.),

— An optional sequence of zero or more PREFERENCES elements containing decoding preferences for the decoder,

— An sequence of zero or more PARTREF elements referring to the parts (using the ID) that the main score has to contain

— An optional MIDIINFO element for information on how to produce MIDI data from the score,

— An optional FORMATPAGECOMPUTERVIEW element with information on how to display the score in computer view,

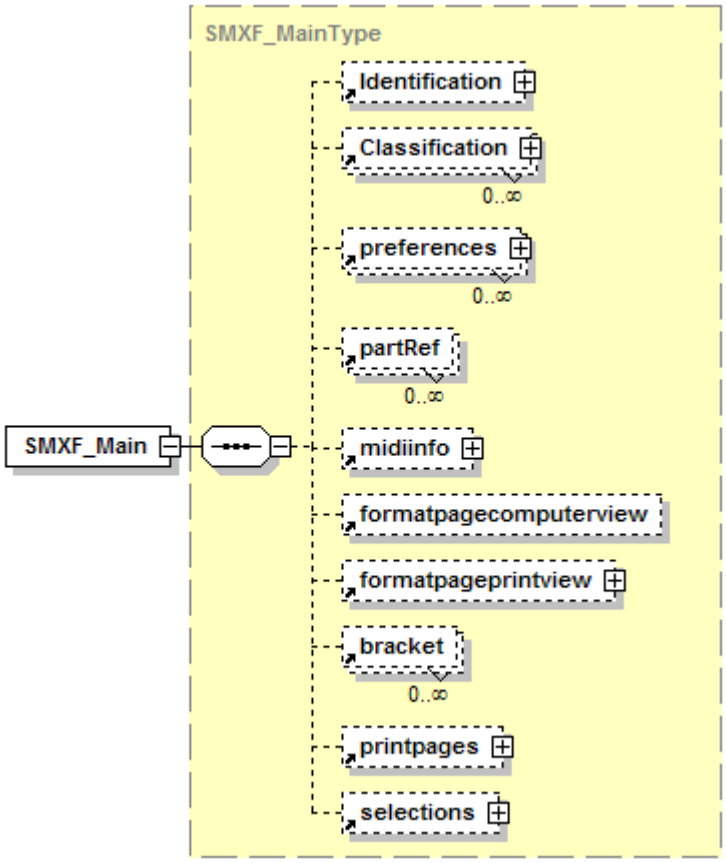
— An optional FORMATPAGEPRINTVIEW element with information about how to print the score,

— An optional sequence of zero or more BRACKET elements for brackets connecting two scores,

— An optional PRINTPAGES element containing additional text and images to be used when printing the score,

— An optional set of SELECTIONS element containing identifications of segment of music,

Table 86 — SMXF_MAIN element syntax

Diagram	
Children	<Identification> <Classification> <preferences> <midiinfo> <formatpagecomputerview> <formatpageprintview> <bracket> <printpages> <selections>
Used by	

Source	`<xs:element name="SMXF_Main" type="SMXF_MainType"/>`
	`<xs:complexType name="SMXF_MainType">`
	`<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="midiinfo" minOccurs="0"/>`
	`<xs:element ref="formatpagecomputerview" minOccurs="0"/>`
	`<xs:element ref="formatpageprintview" minOccurs="0"/>`
	`<xs:element ref="bracket" minOccurs="0" maxOccurs="unbounded"/>`
	`<xs:element ref="printpages" minOccurs="0"/>`
	`<xs:element ref="selections" minOccurs="0"/>`
	`<xs:choice>`
	`<xs:element name="partRef" minOccurs="0" maxOccurs="unbounded">`
	`<xs:complexType>`
	`<xs:attribute name="scoreID" type="xs:positiveInteger" use="required"/>`
	`</xs:complexType>`
	`</xs:element>`
	`<xs:element ref="SMXF_Part" minOccurs="0" maxOccurs="unbounded"/>`
	`</xs:choice>`
	`</xs:sequence>`
	`</xs:complexType>`

8.4.2 <Identification>

See subclause 8.3.2

8.4.3 <Classification>

See subclause 8.4.3

8.4.4 <preferences>

See subclause 8.3.4

8.4.5 <partRef>

A PARTREF element contains the reference to the parts that should be presented in the main score. It refers to the part using its ID.

Table 87— PARTREF element syntax

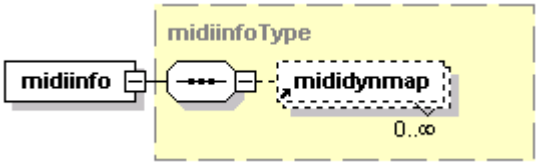
Diagram			
Used by	<SMXF_Main>		
Attributes	**Name**	**Type**	**Description**
	scoreID	positiveINteger	the numeric ID of the part it refers to.

Source	`<xs:element name="partRef" type="partRefType"/>`
	`<xs:complexType name="partRefType">`
	`<xs:attribute name="SCOREID" type="xs:positiveInteger" use="required"/>`
	`</xs:complexType>`		

8.4.6 <midiinfo>

A MIDIINFO element contains information on MIDI generation for all the parts of the main score. It contains a sequence of zero or more MIDIDYNMAP elements for the mapping of dynamic symbols (e.g., p, pp, ff, fff, etc.)

Table 88 — MIDIINFO element syntax

Diagram	
Children	<mididynmap>
Used by	<SMXF_Main>

Source	`<xs:element name="midiinfo" type="midiinfoType"/>`
	`<xs:complexType name="midiinfoType">`
	`<xs:sequence>`

```
        <xs:element ref="mididynmap" minOccurs="0" maxOccurs="unbounded"/>
      </xs:sequence>
    </xs:complexType>
```

8.4.7 <mididynmap>

A MIDIDYNMAP contains the mapping of a dynamic symbol.

Table 89 — MIDIDYNMAP element syntax

Diagram			
Used by	<midiinfo>		
Attributes	**Name**	**Type**	**Description**
	DYN	enum	Identifies the dynamic symbol, can be one of PPPP, PPP, PP, P, MP, MF, F, FF, FFF, FFFF
	VOLUME	nonNegativeInteger	The volume associated with the dynamic symbol (0 -255)
Source			

```
   <xs:element name="mididynmap" type="mididynmapType"/>
   <xs:complexType name="mididynmapType">
      <xs:attribute name="DYN" use="required">
         <xs:simpleType>
            <xs:restriction base="xs:NMTOKEN">
               <xs:enumeration value="PPPP"/>
               <xs:enumeration value="PPP"/>
               <xs:enumeration value="PP"/>
               <xs:enumeration value="P"/>
               <xs:enumeration value="MP"/>
               <xs:enumeration value="MF"/>
               <xs:enumeration value="F"/>
               <xs:enumeration value="FF"/>
               <xs:enumeration value="FFF"/>
               <xs:enumeration value="FFFF"/>
            </xs:restriction>
         </xs:simpleType>
      </xs:attribute>
      <xs:attribute name="VOLUME" type="xs:nonNegativeInteger"
use="required"/>
    </xs:complexType>
```

8.4.8 <formatpagecomputerview>

See subclause 8.3.8

8.4.9 <formatpageprintview>

See subclause 8.3.9

8.4.10 <bracket>

A BRACKET element represents a bracket between two scores. Two types of brackets are supported:

Table 90 — BRACKET types

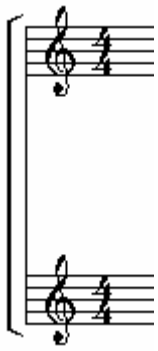
TYPES	EXAMPLE
BRACKET for a square bracket	
BRACE, for a brace bracket	

Table 91 — BRACKET element syntax

Diagram	bracket		
Used by	<SMXF_Main>		
Attributes	**Name**	**Type**	**Description**
	ID	positiveInteger	A number uniquely identifying the bracket in the main score
	TYPE	enum	The type of bracket, it can be BRACKET or BRACE
	STARTPARTID	positiveInteger	Indicates the part where the bracket starts
	ENDPARTID	positiveInteger	Indicates the part where the bracket ends
Source	<xs:element name="bracket" type="bracketType"/> <xs:complexType name="bracketType"> <xs:attribute name="ID" type="xs:positiveInteger" use="required"/> <xs:attribute name="TYPE" use="required"> <xs:simpleType> <xs:restriction base="xs:NMTOKEN"> <xs:enumeration value="BRACKET"/> <xs:enumeration value="BRACE"/> </xs:restriction> </xs:simpleType>		

	</xs:attribute> <xs:attribute name="STARTPARTID" type="xs:positiveInteger" use="required"/> <xs:attribute name="ENDPARTID" type="xs:positiveInteger" use="required"/> </xs:complexType>

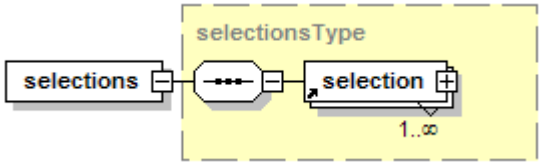
8.4.11 <printpages>

See subclause 8.3.54

8.4.12 <selections>

The SELECTIONS element contains the selections defined for the score.

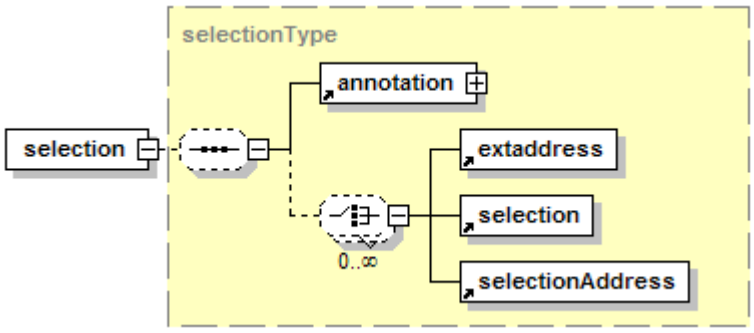
Table 92 — SELECTIONS element syntax

Diagram	
Used by	<SMXF_Main>
Source	<xs:element name="selections" type="selectionsType"/> <xs:complexType name="selectionsType"> <xs:sequence> <xs:element ref="selection" maxOccurs="unbounded"/> </xs:sequence> </xs:complexType>

8.4.13 <selection>

The SELECTION element contains an ANNOTATION element followed by any number and in any order ADDRESS, SELECTION and SELECTIONADDRESS elements.

Table 93 — SELECTIONS element syntax

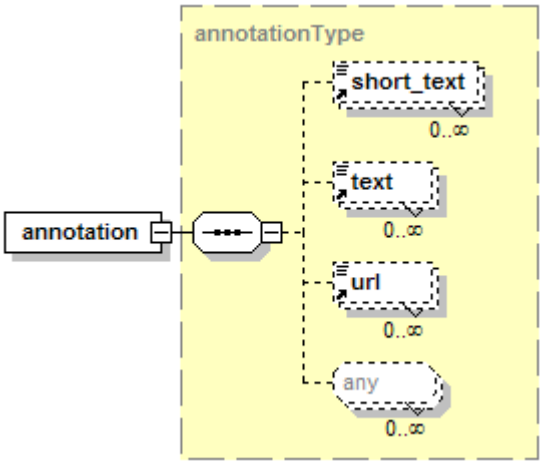
Diagram			
Used by	<selections> <selection>		
Attributes	**Name**	**Type**	**Description**

	ID	string	a string uniquely identifying the selection within the whole score
	name	string	an optional descriptive name
Source	<xs:element name="selection" type="selectionType"/> <xs:complexType name="selectionType"> <xs:sequence minOccurs="0"> <xs:element ref="annotation"/> <xs:choice minOccurs="0" maxOccurs="unbounded"> <xs:element ref="extaddress"/> <xs:element ref="selection"/> <xs:element ref="selectionAddress"/> </xs:choice> </xs:sequence> <xs:attribute name="id" type="xs:string" use="required"/> <xs:attribute name="name" type="xs:string"/> </xs:complexType>		

8.4.14 <annotation>

The ANNOTATION element represents the annotation defined for the selection, it can contain: any number of SHORT_TEXT elements for short descriptions (in many languages), any number of TEXT elements for textual descriptions (in any language), any number of URL elements for multimedia annotations and any number of any other elements (e.g., MPEG-7 descriptors).

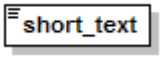
Table 94 — ANNOTATION element syntax

Diagram	
Used by	<selection>
Source	<xs:element name="annotation" type="annotationType"/> <xs:complexType name="annotationType"> <xs:sequence> <xs:element ref="short_text" minOccurs="0" maxOccurs="unbounded"/> <xs:element ref="text" minOccurs="0" maxOccurs="unbounded"/> <xs:element ref="url" minOccurs="0" maxOccurs="unbounded"/> <xs:any processContents="lax" minOccurs="0" maxOccurs="unbounded"/> </xs:sequence> </xs:complexType>

8.4.15 <short_text>

The SHORT_TEXT element contains the short form of the description.

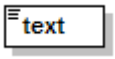
Table 95 — SHORT_TEXT element syntax

Diagram			
Used by	<annotation>		
Attributes	**Name**	**Type**	**Description**
	lang	language	language used for the description
Source	<xs:element name="short_text"> <xs:complexType> <xs:simpleContent> <xs:extension base="xs:string"> <xs:attribute name="lang" type="xs:language"/> </xs:extension> </xs:simpleContent> </xs:complexType> </xs:element>		

8.4.16 <text>

The TEXT element contains the annotation description.

Table 96 — TEXT element syntax

Diagram			
Used by	<annotation>		
Attributes	**Name**	**Type**	**Description**
	lang	language	language used for the description
Source	<xs:element name="text"> <xs:complexType> <xs:simpleContent> <xs:extension base="xs:string"> <xs:attribute name="lang" type="xs:language"/> </xs:extension> </xs:simpleContent> </xs:complexType> </xs:element>		

8.4.17 <url>

The URL element is used for multimedia annotations.

Table 97 — URL element syntax

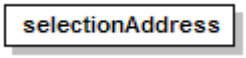
Diagram	
Used by	<annotation>

Source	`<xs:element name="url">`
	`<xs:complexType>`
	`<xs:simpleContent>`
	`<xs:extension base="xs:string">`
	`</xs:extension>`
	`</xs:simpleContent>`
	`</xs:complexType>`
	`</xs:element>`

8.4.18 <selectionAddress>

The SELECTIONADDRESS element contains a reference to another selection. The references have to avoid loops.

Table 98 — SELECTIONADDRESS element syntax

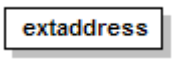
Diagram			
Used by	<selection>		
Attributes	**Name**	**Type**	**Description**
	ref	string	a reference to a selection id

Source	`<xs:element name="selectionAddress" type="selectionAddressType"/>`
	`<xs:complexType name="selectionAddressType">`
	`<xs:attribute name="ref" type="xs:string" use="required"/>`
	`</xs:complexType>`		

8.4.19 <extaddress>

The EXTADDRESS element contains a reference to a figure in a score. It extends the ADDRESS element (see subclause 8.3.53) with a reference to the part where the figure is.

Table 99 — EXTADDRESS element syntax

Diagram			
Used by	<selection>		
Attributes	**Name**	**Type**	**Description**
	PART	positiveInteger	A number uniquely identifying the part (ID of the part)

	MEASURE	positiveIntegere	see subclause 8.3.53
	LAYER	positiveInteger	see subclause 8.3.53
	FIGURE	positiveInteger	see subclause 8.3.53
	CHORD.OR.BEAM	nonNegativeInteger	see subclause 8.3.53
	CHORD.IN.BEAM	nonNegativeInteger	see subclause 8.3.53
Source	<xs:element name="extaddress" type="extaddressType"/> <xs:complexType name="addressType"> <xs:attribute name="PART" type="xs:positiveInteger" use="required"/> <xs:attribute name="MEASURE" type="xs:positiveInteger" use="required"/> <xs:attribute name="LAYER" type="xs:positiveInteger" use="required"/> <xs:attribute name="FIGURE" type="xs:positiveInteger" use="required"/> <xs:attribute name="CHORD.OR.BEAM" type="xs:nonNegativeInteger"/> <xs:attribute name="CHORD.IN.BEAM" type="xs:nonNegativeInteger"/> </xs:complexType>		

8.5 Lyrics (SMXF Lyric)

The lyric text is modelled as a sequence of syllables. Each syllable starts on a certain note and it may be extended to a following one (not necessarily in the same measure). The syllables are to be drawn aligned with the starting figure, all on the same horizontal line, except for refrains where different text is reported in different lines within the same melody. In order to associate lyrics to music notation, two different models can be used: (i) each note presents a relationship to one or more syllables, (ii) any associated syllable presents a reference (symbolic or absolute) to the music notation symbol. The first solution is the best solution if only one lyric text is associated with the music score. When more lyrics are associated with the same music score, the first solution becomes too complex, since each figure has to refer to all the syllables of the several lyrics. In these cases, the second solution can be better since it allows the inclusion of new lyrics even without modifying the music notation.

Similar to other symbols, lyric text is handled as horizontal symbols. The *Part* object refers to a list of *Syllable* objects and each syllable has a reference to the starting and ending *Figure* object. The order of the syllables in the list follows the lyric text, so that the text can be reconstructed by following the list.

Syllables are separated by using:

— an empty space when the two consecutive syllables belong to different words;

— a hyphenation mark when the syllables belong to the same word;

— a continuous line if the ending syllable of a word has to be extended on more than one note.

An attribute of the SYLLABLE element (*SEP*, in the following) is used to indicate which kind of separator has to be used: ‘ ’ for empty space, ‘n’ for new line, ‘/’ for hyphenation mark, ‘_’ for syllable extension at the end of a word and ‘-’ for syllable extension inside a word.

The relation between separators and start/end figures has to be analysed in a more detailed manner. The *start* figure reference is always to the *Figure* under which the syllable has to be positioned; on the other hand, when it comes to the *end* figure, what follows can be applied:

— if the syllable is a single word or is the ending syllable of a word and it is not extended (*SEP* = ‘ ’ or *SEP* = ‘n’), then the *end* figure is not set (meaning NULL);

— if the syllable is not the last one and it is not extended (*SEP* = '/'), then the *end* figure is set to the figure where the next syllable is positioned;

— if the syllable is not the last one and it is extended (*SEP* = '-'), then the *end* figure is set to the figure where the next syllable is positioned. The symbol '-' in the text word can be written by using '\-';

— if the syllable is the ending one and it is extended (*SEP* = '_'), then the *end* figure is set to the figure under which the extension line has to be drawn, generally it is the previous figure of the next syllable.

When two consecutive syllables of different words (one starting and the other ending with a vowel) have to be sung on the same note, as in Figure 11, the special character '+' is used to represent the slur in the syllable text. Therefore, the two highlighted 'syllables' are represented through texts "ra+in" and "me+a" in the *Syllable* objects. The drawing/printing engine replaces the + character with a slur when displaying or printing the music.

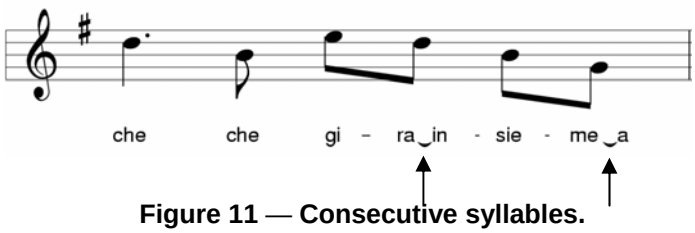


Figure 11 — Consecutive syllables.

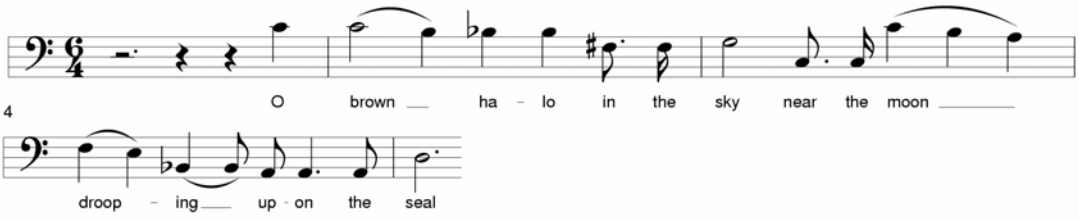
The different lines of lyrics are managed using an attribute (*LINE*) of the *syllable* pointing out on which line the syllable has to be placed.

Another possibility is to have different lyrics to the same staff. This is possible when the staff presents more voices (for instance the voice of Soprano and that of Tenor), each voice may have its own lyrics. In this way, it is possible to have two or more parts for singers on the same staff with their related music. It is also possible to have different lyrics associated to the same voice as frequently occurs in sacral music that provides the same lyrics in different languages under the same staff.

In addition, a real multilingual lyrics representation is supported since different syllable sequences can be 'plugged' on a *score*, depending on the language.

A specific language is adopted by the users to enter lyric text. This language is interpreted in the editor and transformed in the lyric model, which can be seen in the music viewer. The example reported in Figure 12 has been produced by using the lyric editor, and presents both English and German lyrics. Please note the different arrangement of slurs and ties.

```
O brown_ ha/lo in the sky near the moon__ droop-ing_ up/on the seal
```




```
Du blas/ser Schein_ am_ Himm/el, der Mond__ sinkt___ hi/nab ins Merr!
```

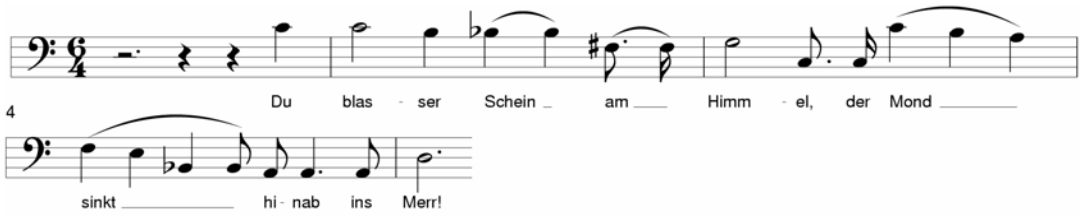


Figure 12 — Example of multilingual lyrics, English and German versions.

As shown in the examples above, the lyrics text is *augmented* with some special characters: '/', '_' and '-' and also the '@' and '+' characters are possible, as it will be shown later in the complete example. Please note that they are extremely useful in some languages while in others their usage is marginal. This language can be parsed to assign each syllable to a note, starting from the first note. The blank character, the carriage return and the '/', '_', '-', '@' characters are considered as syllable separators, whereas the '+' character is not. When such as symbols are part of the lyrics to be shown in the score, they have to written as '\/', '_', '\-', '\@'.

Particular situations occur when syllable extensions have to be entered. For this reason separators such as '-' and '_' are used at the end of the syllable to state that it is extended; the separator can be repeated to state the number of notes on which the syllable is extended. For example the "moon" syllable in the English lyrics is followed by two '_' separators, meaning the syllable is extended to the two following notes. The same thing happens with the "sinkt" syllable in the German lyrics, where it is followed by three '_' separators to extend the syllable over three notes.

In some specific circumstances avoiding any syllable assignment is necessary; for this reason the '@' separator has been introduced to skip one note during the assignment of the syllables.

As syllable separators, the lyric text includes spaces, returns and tabs used to format the lyrics. The idea is to grant the user the option to view the lyric text just as a poem, thus hiding the special separators but viewing the text correctly formatted with spaces and carriage returns. To achieve this, the model has to also store this kind of information which is useless for lyric representation in the score, but becomes useful when viewing the lyric text as a poem.

The solution adopted is to use a *text* element to store such kind of information. This special *Syllable* object is skipped during the syllable visualization on the score, not being associated with a figure. Besides, some other textual information like the title, the author, the date of composition etc can be found in the lyrics, thus the '{' and '}' characters have been used to mark the beginning and the end of a comment section (which is stored in the model as it is, while it is not associated with the score). The following is an example:

```
{<H1>}{Canto della Terra}{</H1>}
{lyrics by Lucio Quarantotto}

Si lo so a/mo/re che io+e te
for/se stia/mo+in/sie/me so/lo qual/che+i/stan/te
…
```

one *Syllable* object contains this text

```
{1999}
```

That is viewed in the lyric editor by hiding the special separators as:

```
Canto della Terra
lyrics by Lucio Quarantotto

Si lo so amore che io e te
forse stiamo insieme solo qualche istante
…
1999
```

The "{<" and ">}" sequences are treated in a special way; they are used to embed HTML formatting commands in the text. When viewing the lyrics by hiding the special operators, the characters between these two markers are completely hidden, thus removing the HTML commands. These sequences are not removed anymore when exporting the lyrics to HTML.

What follows is clarification on how blank spaces are treated within the model. The first blank character of a sequence of blanks is stored in the *SEP* attribute and the following black characters stored in a comment syllable.

The management of refrains is a complex task. The main constraint is that the entered lyric text has to be in reading order, which means the syllables of the first line are entered first, then the syllables of the second line etc. A way to mark the beginning of a refrain is needed and the '[' character was chosen to point out that the note associated with the following syllable has to be considered as the refrain start. The character '%' is used like a RETURN, the assignment returns back to the previous refrain start, thus incrementing the current line. Finally the ']' character is used to end the refrain and decrement the current line.

For example the sequence **"A [B % C] D"** (where A, B, C, D are syllable sequences of any complexity) produce something structured like:

1. A B D

2. C

where 1. and 2. represent the lyric line where the syllables are positioned under the music score staff. The sequence **"A [B % C %D % E] F"** produces a lyric structured as follows under the score:

1. A B F

2. C

3. D

4. E

The above introduced operators can be nested as in **"A [B [C % D] E % F [G % H] I] J"**, thus producing the following complex structure:

1. A B C E J

2. D

3. F G I

4. H

To avoid inconsistencies the number of notes used in the assignment of each refrain should be the same, for example in sequence “A [B % C] D”, the syllable sequences B and C have to use the same number of notes. A way to avoid multiple assignments due to different number of used notes consists of storing the number of notes assigned and the next usable note when ‘%’ is found, and in restoring the assigning position with the maximum number of used notes when the end ‘]’ is found.

8.5.1 <SMXF_Lyric>

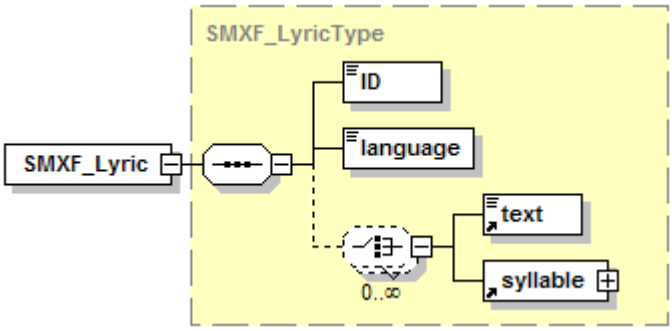
The SMXF_LYRIC element contains the lyrics in a specific language for a part. It contains:

— An ID element which identifies the lyrics,

— A LANGUAGE element with the language used for the lyric (e.g., "en", "it", etc.),

— A sequence of zero or more TEXT or SYLLABLE elements.

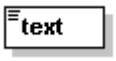
Table 100 — SMXF_LYRIC element syntax

<table>
<tr><td>Diagram</td><td> 
</td></tr>
<tr><td>Children</td><td><text> <syllable></td></tr>
<tr><td rowspan="4">Attributes</td><td>Name</td><td>Type</td><td>Description</td></tr>
<tr><td>SCOREID</td><td>string</td><td>ID of the score part to which the lyrics applies to</td></tr>
<tr><td>BASELINE</td><td>decimal</td><td>Multiple of the space (distance between two staff lines) indicating the position of the first line of lyrics</td></tr>
<tr><td>LINESDIST</td><td>decimal</td><td>Multiple of the space indicating the distance between two lines of lyrics</td></tr>
<tr><td>Source</td><td colspan="3"><pre>
<xs:element name="SMXF_Lyric" type="SMXF_LyricType"/>
<xs:complexType name="SMXF_LyricType">
  <xs:sequence>
    <xs:element name="ID" type="xs:string"/>
    <xs:element name="language" type="xs:string"/>
    <xs:choice minOccurs="0" maxOccurs="unbounded">
      <xs:element ref="text"/>
      <xs:element ref="syllable"/>
    </xs:choice>
  </xs:sequence>
  <xs:attribute name="SCOREID" type="xs:string" use="required"/>
  <xs:attribute name="BASELINE" type="xs:decimal" default="1.0"/>
  <xs:attribute name="LINESDIST" type="xs:decimal" default="1.0"/>
</xs:complexType>
</pre></td></tr>
</table>

8.5.2 <text>

A TEXT element is used to contain the syllable text (when inside a SYLLABLE element) or to contain text outside the lyrics itself, like the title or formatting commands (when inside a SMXF_LYRIC element).

Table 101 — TEXT element syntax

Diagram	
Used by	<SMXF_Lyric> <syllable>
Content	Text of the syllable
Source	`<xs:element name="text" type="xs:string"/>`

8.5.3 <syllable>

A SYLLABLE element contains information on a single syllable associated with a note. A syllable can be extended on multiple notes or can be associated with only one note. A syllable element contains:

— A TEXT element with the syllable text.

— A START element identifying to which note the syllable applies to.

— An optional END element identifying the note where the syllable ends.

The SEP attribute indicates the syllable separator and possibly the kind of extension:

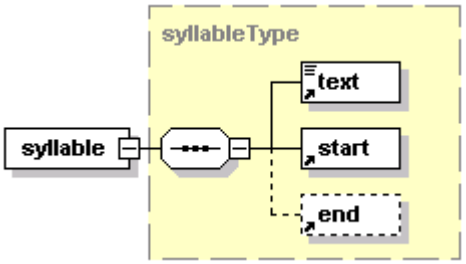
— "/", " " or "n" for a single syllable ("n" indicates a new line) and in this case the END element should not be present.

— "_" for an extension at the end of a word (the END elements indicates the ending note), the "_" is repeated as many times as the number of notes on which the syllable is extended.

— "-" for an extension inside a word (the END elements indicates the ending note) the "-" is repeated as many times as the number of notes on which the syllable is extended.

— "@" for skipping notes, it can be repeated as many times as the number of notes to skip.

Table 102 — SYLLABLE element syntax

Diagram	
Children	<text> <start> <end>

Used by	<SMXF_Lyric>		
Attributes	**Name**	**Type**	**Description**
	LINE	nonNegativeInteger	the lyric line in which the syllable is contained
	REFRAIN	string	contains the refrain codes "[", "%", "]"
	SEP	string	contains the syllable separators to the next syllable
Source	<xs:element name="syllable" type="syllableType"/> <xs:complexType name="syllableType"> <xs:sequence> <xs:element ref="text"/> <xs:element ref="start"/> <xs:element ref="end" minOccurs="0"/> </xs:sequence> <xs:attribute name="LINE" type="xs:nonNegativeInteger"/> <xs:attribute name="REFRAIN" type="xs:string"/> <xs:attribute name="SEP" type="xs:string"/> </xs:complexType>		

8.5.4 <start>

A START element refers to a figure in the score in the same way as the ADDRESS element (see section 8.3.53).

Table 103 — START element syntax

Diagram	start		
Used by	<syllable>		
Attributes	**Name**	**Type**	**Description**
	MEASURE	positiveInteger	See 8.3.53
	LAYER	positiveInteger	See 8.3.53
	FIGURE	positiveInteger	See 8.3.53
	CHORD.OR.BEAM	nonNegativeInteger	See 8.3.53
	CHORD.IN.BEAM	nonNegativeInteger	See 8.3.53

Source	<xs:element name="start" type="addressType"/> <xs:complexType name="addressType"> <xs:attribute name="MEASURE" type="xs:positiveInteger" use="required"/> <xs:attribute name="LAYER" type="xs:positiveInteger" use="required"/> <xs:attribute name="FIGURE" type="xs:positiveInteger" use="required"/> <xs:attribute name="CHORD.OR.BEAM" type="xs:nonNegativeInteger"/> <xs:attribute name="CHORD.IN.BEAM" type="xs:nonNegativeInteger"/> </xs:complexType>

8.5.5 <end>

An END element refers to a figure in the score in the same way as the ADDRESS element (see section 8.3.53).

Table 104 — END element syntax

Diagram	end		
Used by	<syllable>		
Attributes	**Name**	**Type**	**Description**
	MEASURE	positiveInteger	See 8.3.53
	LAYER	positiveInteger	See 8.3.53
	FIGURE	positiveInteger	See 8.3.53
	CHORD.OR.BEAM	nonNegativeInteger	See 8.3.53
	CHORD.IN.BEAM	nonNegativeInteger	See 8.3.53
Source	<xs:element name="end" type="addressType"/>		

9 Symbolic Music Synchronization Information (SM-SI)

9.1 Symbolic Music Synchronization Information (SM-SI) Introduction

Symbolic Music Synchronization Information allows the music score to be presented synchronously with a BIFS scene, with any other timed resource (e.g., audio, video), or even with a live event. The Synchronization Information (SyncInfo) is used to determine at each time instance which measure is currently playing and then arrange its visualization. The SyncInfo can be transmitted in two ways, all in one chunk within the access unit for the synchronization with recorded events (audio/video) or one per access unit in case of live events.

9.2 SM-SI Binary Format

The synch_file class is used to indicate to the decoder when each measure of the score has to be presented. It is specified by the following semantics:

```
/*********************************
     synch_file bitstream formats
*********************************/

class synch_file {
  bit(2) synchType;
  switch(synchType) {
    case 0b00: // is synchronization with a live event
      unsigned int(16) measureNumber;// the measure number to be shown at the time
                                     // the access unit arrives at the decoder
      break;
    case 0b01:
      bit(1) more_data = 1;

      while (more_data) {
        bit(1) jump;  // if 1 it indicates a jump to a measure,
                      // if 0 it indicates the following measure (starting from measure 1)
        if(jump)
          unsigned int(16) measureNumber;    //the measure number to jump to
        unsigned int(16) duration;       //duration of the measure in milliseconds
        bit(1) more_data;
      }
      break;
    case 0b10: reserved
    case 0b11: reserved
  }
}
```

The synchType = 0b00 is used for synchronization with real-time streaming, when the decoder receives an Access Unit with such synchronization information the indicated measure has to be executed/presented.

The synchType = 0b01 is used to provide the duration (in milliseconds) of each measure of the score and to possibly indicate the order in which the measures are played/rendered (e.g., in case of refrains) and thus in which order they should be displayed. The semantics sees the execution of measures one after the other following their ordering. The jump bit is used to indicate when the playing order has to jump to another measure overriding the natural order.

Example:

The following example indicates the execution of a score with 4 measures

Measure	1	2	3	2	3	4	1	2	3	4
Duration (ms)	1000	1000	1000	1500	1500	1500	1500	1500	1500	1500

```
jump: 0
duration: 1000 (duration of measure 1)
jump: 0
duration: 1000 (duration of measure 2)
jump: 0
duration: 1000 (duration of measure 3)
jump: 1
measureNumber: 2
duration: 1500 (duration of measure 2)
jump: 0
duration: 1500 (duration of measure 3)
jump: 0
duration: 1500 (duration of measure 4)
```

```
jump: 1
measureNumber: 1
duration: 1500 (duration of measure 1)
jump: 0
duration: 1500 (duration of measure 2)
jump: 0
duration: 1500 (duration of measure 3)
jump: 0
duration: 1500 (duration of measure 4)
```

10 Symbolic Music Formatting Language (SM-FL)

10.1 SMR Formatting Introduction

This subclause is Informative section.

The formatting engine for symbolic music representation rendering and sheet music production is divided into two main systems: the insertion and positioning engine and the justification and line-breaking module.

The Symbolic Music Formatting Language (SM-FL) is defined to allow describing the insertion point and positioning of music symbols. It specifies a rule-based formatting language and engine, which is used to describe sets of rules and conditions interpreted in real time by an inferential engine at the moment that the position of symbols has to be estimated. The SM-FL rules define formatting actions: they assign specific values to specific parameters related to the visual rendering of the music symbols. For example, a rule states that the stem is characterized by a given length. The actions to be performed are activated by the conditions based on the music context of the symbol that is under evaluation. These conditions are expressed in terms of conditional sentences and define a music scenario. For example, a music scenario could consist of a note not belonging to a chord, i.e., non-polyphonic, and with a height set within a certain range. The verification of a music scenario defined by a conditional sentence leads to the application of a certain formatting rule to the symbol under evaluation.

The SM-FL formatting engine is invoked whenever a symbol is represented, i.e., it may occur when a new symbol is added and/or when a SMF-XF is loaded and rendered..

10.2 General architecture of the formatting engine

In Figure 13, the general architecture of formatting and justification engines are shown. The functionality of each component of the architecture is outlined as follows. The rendered image of the music score is both the source and the result of the formatting and justification engines. The rendered image is used to show music scores and in case of authoring the user may interact by sending commands to insert, delete symbols and to edit the music score in general.

In an SMR tool, the music score is modelled by using an object-oriented model, including main score and part within the same model. The parts are used to build the main score. The additional information for the main score organization is contained in a separate file. Music notation symbols retain all the relationships among the notation symbols and music structure in general, disregarding the values of graphic details concerning the music formatting on the rendering device (e.g., absolute or relative coordinates).

When the SM-FL engine for the automatic formatting is activated, the process of placing a symbol in the score involves some computation of conditions and activates the corresponding rules. The conditions depend on the music context for example the stem direction of notes may depend on the note height in the staff. In the object-oriented model, there are permanent parameters, therefore the value is computed only once at the very insertion of the symbol in the score. Furthermore, some dynamic parameters are computed every time the rendered image of the score is produced. This is very important when the score is visualized in a computer window (or in any resizable window of a device); whereas it is less important when the goal of the algorithm is to arrange the score for producing a music sheet or an image.

In order to make the process of music score visualization shorter, the SM-FL formatting engine has been conceived as being made up of two parts:

- the ***Insertion Engine*** evaluates permanent parameters and it is invoked every times a new music element is added to the model, mainly in the authoring phase;
- the ***Positioning Engine*** is invoked every time the rendering of the music score has to be redrawn (for example on image resizing or on score scrolling). This allows estimation of the dynamic parameters related to symbols positioning.

The formatting engine estimates the context needed in order to assess the conditions, according to the music notation symbol involved. To perform the context evaluation, the SM-FL engine makes queries to the object-oriented model of music. The context evaluation permits the identification of rules to be applied for setting the parameters of the symbol being inserted or positioned with the appropriate values. Each activated rule points at the value to be set for the parameter under evaluation. For example, the stem height, the starting position of the slur, etc.

Conditions and rules are written in SM-FL language and are included within a unique file loaded when the decoder is invoked. The entire set of SM-FL rules and conditions can be reloaded/resent so as to permit the possibility of changing rules and reapplying them to the same music score.

Parameters related to the music visualization are computed by the decoder in real time, on the basis of the SM-FL Positioning conditions. Some formatting parameters of the music notation symbols (for example direction of the note stems) may be stored in the SM-XL and expressed in terms of simple symbolic relationships (for example, flipping up/down the stem, above and below for expressions concerning the note they are referred to). This is useful in order to cope with exceptions instead of computing them on the basis of SM-FL rules at run time. The context evaluation and the estimation of positioning parameters are based on the analysis of other music symbols in the score. Therefore, the rendering is strictly dependent on the positioning engine of the formatting engine.

The estimation of horizontal spaces among music symbols is performed by the justification and the line breaking modules (see Figure 13).

The SM-FL and the justification engines perform their task independently from one another. In the decoder the justification and the line breaking engines can be independently disabled in order to observer the difference and to see a stable music score when the music is inserted.

The parameters set by the justification engine are the spaces between the score symbols. They are not related to the formatting parameters set with SM-FL rules, since the SM-FL estimates only relative displacements with respect to the other symbols. When music symbols are horizontally spaced by the justification module and formatted by SM-FL positioning engine, the line-breaking module is capable of arranging music score in order to fill the page/view margins. The three modules set different parameters, thus contributing to the resulting visualization of the music score on the computer screen as well as on the print page.

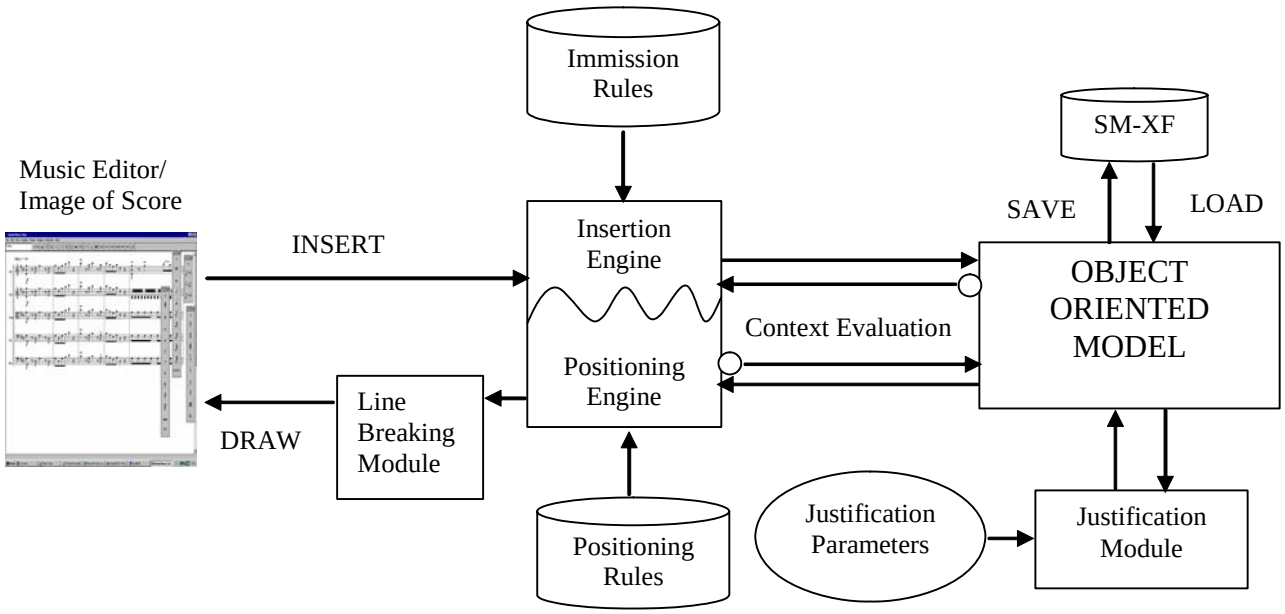


Figure 13 – general architecture of an SMR decoder formatting engine

10.3 The SMR Rendering Rule Approach

The decision process of inserting and positioning music notation symbols in music sheets must cope with a huge number of variables which characterize the music score context. This leads MPEG-4 SMR to abandon a strictly procedural organization of the formatting algorithms in favour of an inferential architecture for the formatting engine.

The SM-FL formatting engine is a *rule based system*, where the rules stating formatting actions for music notation symbols are interpreted in real-time. The formatting actions define the values to be assigned to the parameters related to the displaying and the visualization of music notation symbols. The actions to be performed are activated by conditions based on the music context of the symbol under insertion and positioning.

In order to take into account several aspects, such as: "*is in polyphony*", or "*note belonging to a chord*", a set of assertions can be described for evaluation of the music context. These assertions can be true or false depending on the context of the symbol under evaluation. According to their values, different actions can be applied in different contexts. Single assertions can be combined to link together assertions in complex conditional sentences. This is a simple mechanism to insert/define specific rules for managing any exceptions. They are always motivated and conceived as something to be activated by the specific context that brought about the exception or the typical formatting rule. The verification of a music scenario, defined by a conditional sentence, leads to the application of a given formatting rule for the symbol under evaluation.

The author can specify the conditional sentences by combining together the single assertions available and the rules to be applied within a certain context. This results in the ability of changing formatting style of the score without spending too much time with extremely tedious activities like editing for modification of music notation, symbols by symbol. This grants an authoring tool with a high flexibility, which relies on the flexibility of the SM-FL language and formatting engine.

The SM-FL language is composed of words belonging to the music background: the syntax is quite simple, so as to be intelligible for users without specific background in information technology. This allows rendering the same music score by using different music style sheets according to the publisher's and author's or final user's needs and tastes.

The ***Insertion rules*** indicate the position of a symbol with respect to the position of another symbols. The graphic elements presenting values to be imposed by the Insertion formatting engine, are reported in the following. For example:

- the stem direction of notes or chords, upward or downward;
- the stem direction for all the notes composing a beamed group;
- the relative position of markers (accents, expressions, ornaments, instrument symbols, bowing, fingering, etc.) with respect to the
  - the note (up/down) they are referred to, either considering or not the direction of the note stem (on stem, opposite to stem).
  - The staff (above/below), etc.;
- the automatic beaming of groups of notes depending on the time signature.

For example when redrawing the SMR decoder window at window resizing or score page scrolling (see clause 11, Relationship with other parts of the standard), the Positioning formatting engine evaluates the context to set dynamic parameters by applying Positioning rules. ***Positioning rules*** indicate the position of symbols with respect to the current visualization. This is expressed in terms of distance among other symbols in a rendered image of the music score. This measure is expressed by using as unit of measure the distance between two consecutive staff lines. This allows the correct visualization of the music score to be arranged regardless of the visualization magnitude.

The dynamic parameters estimated by positioning rules are, for example:

- the stem length for notes, for chords and for notes and chords belonging to beams;
- the position and the angle/slope of the lines of beamed notes or chords;
- the coordinates of the markers with respect to the note they are referred to, expressed in terms of distance between staff lines (dy) and note head width (dx);
- the position of the symbols, inside or outside the staff, on the left or on the right of the notehead, etc.;
- the coordinates of horizontal symbols such as: slurs, bend, crescendo, decrescendo, change of octave, etc.

When more than one marker (expression, articulation, accent, etc., are generically called markers) is associated with a note, the ***Priority Rule*** is used. This rule specifies the order with which the positioning rules have to be applied, in order to place symbols around the figures (notes and rests are generically called figures). Symbols with higher priority are drawn closer to the figure even if they have been added to the score after lower priority symbols. The insertion of a high priority symbols implies the re-estimation of the position of the lower priority symbols. The priority rule has the form of a list, where symbols have a decreasing order of priority.

10.3.1 Note height

For the evaluation of conditions regarding notes the height of the note inside the staff is of primary importance.. The height of the note in the formatting conditions is slightly different from the height used in the SM-XF format. Figure 12 shows the reference numbers assigned to the note heights as referred to the lines and spaces of the staff in SM-FL language. When tablatures are used the midline is always at 0.

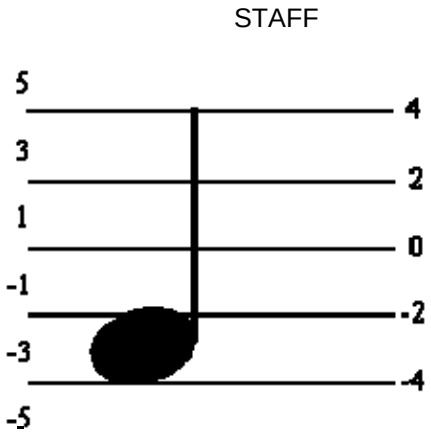


Figure 15 – The heights of the notes in SM-FL with their position on the staff

Please note that having the positions numbered on the staff gives the possibility of defining the rules according to the position of the note on the staff. Other languages and models for music notation represent the position of the note on the basis of the note pitch and thus on the semantic interpretation of the note. The two models can be interchanged by taking into account the current clef and key signature.

10.4 Syntax of rules and conditions

10.4.1 <SMFL>

The SMFL element contains the formatting information to be used for score rendering.

It contains:

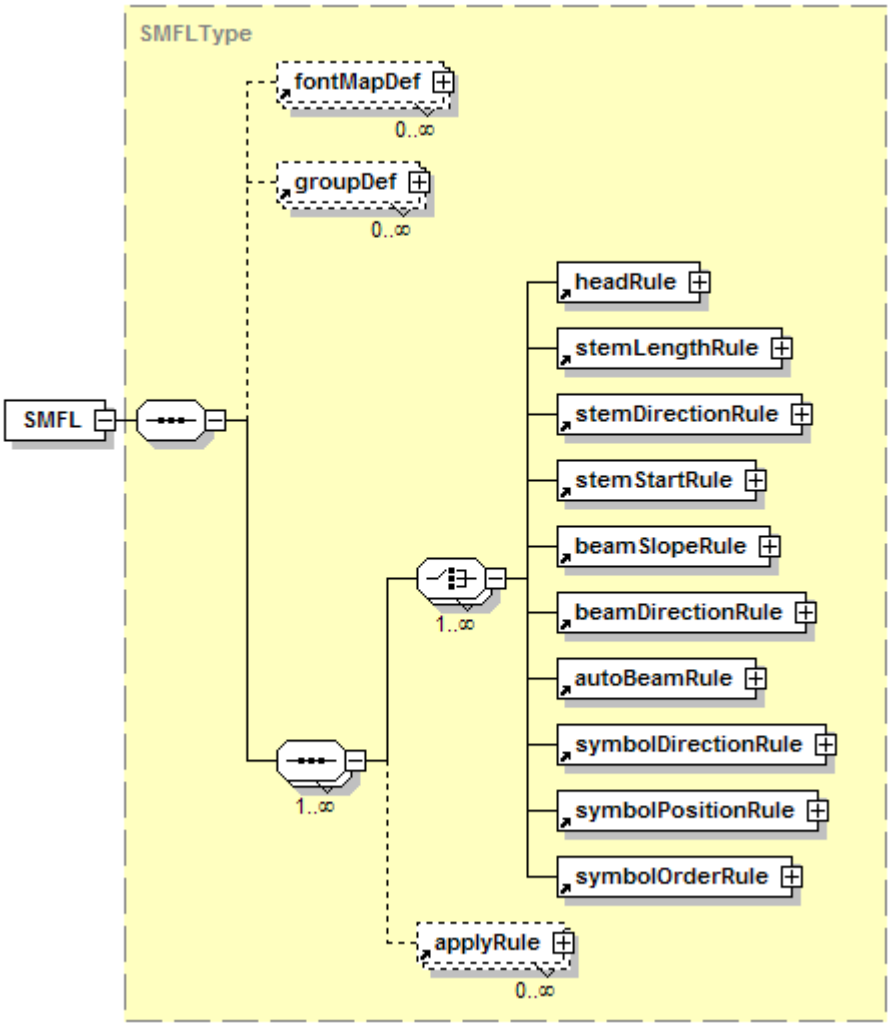
— An optional sequence of FONTMAPDEF elements containing the mappings for the fonts used in the score

— An optional sequence of GROUPDEF elements containing the custom symbols groups definition

— A sequence of rule definitions (HEADRULE, STEMLENGTHRULE, STEMDIRECTIONRULE, STEMSTARTRULE, etc… elements) followed with rule application conditions (APPLYRULE elements)

Table 105 — SMFL element syntax

Diagram



Children

Source

```
<xs:element name="SMFL" type="SMFLType"/>
<xs:complexType name="SMFLType">
   <xs:sequence>
      <xs:element ref="fontMapDef" minOccurs="0" maxOccurs="unbounded"/>
      <xs:element ref="groupDef" minOccurs="0" maxOccurs="unbounded"/>
      <xs:sequence maxOccurs="unbounded">
         <xs:choice maxOccurs="unbounded">
            <xs:element ref="headRule"/>
            <xs:element ref="stemLengthRule"/>
            <xs:element ref="stemDirectionRule"/>
            <xs:element ref="stemStartRule"/>
            <xs:element ref="beamSlopeRule"/>
            <xs:element ref="beamDirectionRule"/>
            <xs:element ref="autoBeamRule"/>
            <xs:element ref="symbolDirectionRule"/>
            <xs:element ref="symbolPositionRule"/>
            <xs:element ref="symbolOrderRule"/>
         </xs:choice>
         <xs:element ref="applyRule" minOccurs="0" maxOccurs="unbounded"/>
      </xs:sequence>
   </xs:sequence>
</xs:complexType>
```

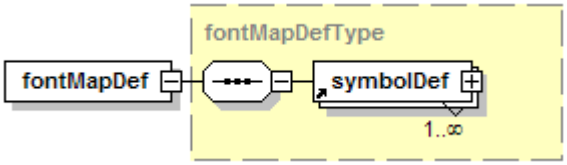
10.4.2 <fontMapDef>

The FONTMAPDEF element allows for the definition of a table for all fonts to be used in the decoder for representing symbols. It contains a sequence of SYMBOLDEF elements for the symbols to be mapped to the fonts that have to be present in the decoder or that can be replaced by font provided via beatstream. All the symbols in the table have their own:

- name according to the SM-XF and
- the UNICODE to identify the symbol in the font, and
- the symbol dimensions

Font names for predefined symbols of CWMN are "musica", "musica2", "musica3", "bigtext", "text"

Table 106 — FONTMAPDEF element syntax

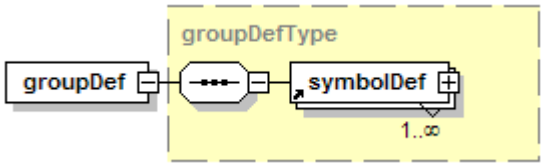
<table>
<tr><td>Diagram</td><td colspan="3"> 
</td></tr>
<tr><td>Children</td><td colspan="3"><symbolDef></td></tr>
<tr><td>Used by</td><td colspan="3"><SMFL></td></tr>
<tr><td rowspan="3">Attributes</td><td>Name</td><td>Type</td><td>Description</td></tr>
<tr><td>name</td><td>NMTOKEN</td><td>The name of the font to be defined</td></tr>
<tr><td>font</td><td>string</td><td>The name of the font to be used for the group</td></tr>
<tr><td>Source</td><td colspan="3"><pre>
<xs:element name="fontMapDef" type="fontMapDefType"/>
<xs:complexType name="fontMapDefType">
  <xs:sequence>
    <xs:element ref="symbolDef" maxOccurs="unbounded"/>
  </xs:sequence>
  <xs:attribute name="font" type="xs:string" use="optional"/>
</xs:complexType>
</pre></td></tr>
</table>

10.4.3 <groupDef>

The GROUPDEF element allows for the definition of a group of custom symbols. It contains a sequence of symbolDef elements for the symbols to be defined for the group. All the symbols in a group share the same rules for handling them (symbolDirectionRule and symbolPositionRule).

The name of the group identifies also the name of the font to be used for symbols rendering.

Table 107 — GROUPDEF element syntax

<table>
<tr><td>Diagram</td><td colspan="3"> 
</td></tr>
<tr><td>Children</td><td colspan="3"><symbolDef></td></tr>
<tr><td>Used by</td><td colspan="3"><SMFL></td></tr>
<tr><td rowspan="3">Attributes</td><td>Name</td><td>Type</td><td>Description</td></tr>
<tr><td>name</td><td>NMTOKEN</td><td>The name of the group defined</td></tr>
<tr><td>font</td><td>string</td><td>The optional name of the font to be used for the group</td></tr>
<tr><td>Source</td><td colspan="3"><pre>
<xs:element name="groupDef" type="groupDefType"/>
<xs:complexType name="groupDefType">
  <xs:sequence>
    <xs:element ref="symbolDef" maxOccurs="unbounded"/>
  </xs:sequence>
  <xs:attribute name="name" type="xs:NMTOKEN" use="required"/>
  <xs:attribute name="font" type="xs:string" use="optional"/>
</xs:complexType>
</pre></td></tr>
</table>

10.4.4 <symbolDef>

The SYMBOLDEF element is used to define a symbol in a group of symbols. It contains:

— the UNICODE code of the character representing the symbol in the font used for carrying out the symbols visualization.

— A DIMENSION element with the information regarding symbol visual extension.

Each symbol has a reference point used for alignment. The position of this reference point is provided in the DX and DY elements of the DIMENSION element. They represent the delta in pixels/points to apply to the reference point to obtain the point normally used for character printing (on the baseline)

— the TOTOP element contains the number of pixel from the reference point to the upper part of the symbol

— the TOBOTTOM element contains the number of pixel from the reference point to the lower part of the symbol

— the TOLEFT element contains the number of pixel from the reference point to the left side of the symbol

— the TORIGHT element contains the number of pixel from the reference point to the right side of the symbol

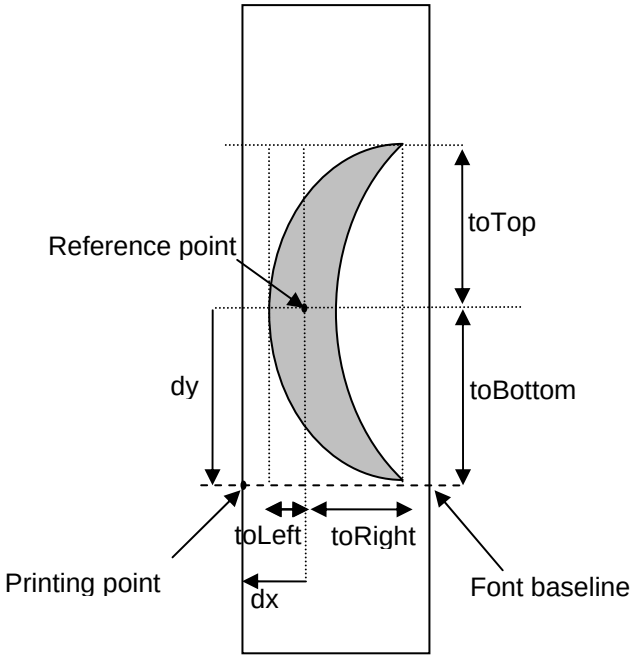
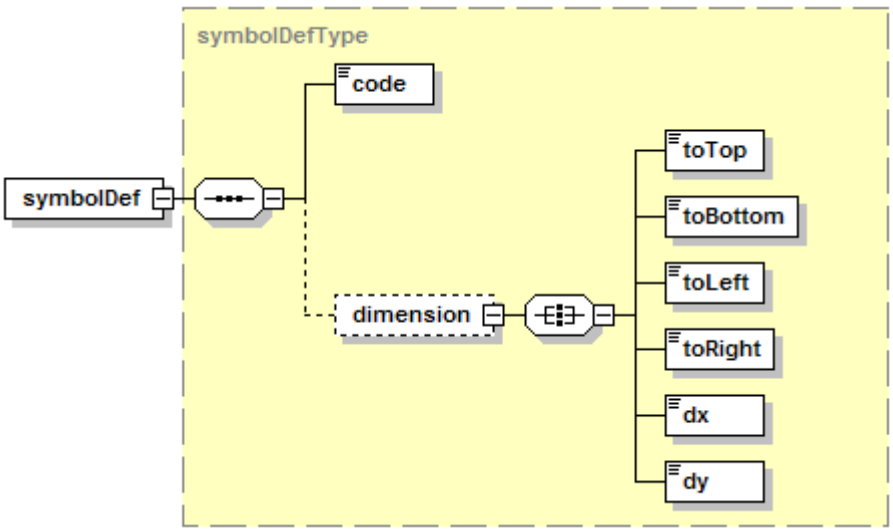


Figure 16 — DIMENSION parameters

Table 108 — SYMBOLDEF element syntax

Diagram			
Children	**Name**	**Type**	**Description**
	code	positiveInteger	The UNICODE code of the character containing the symbol visual representation
	toTop	integer	The pixel from the reference point to the upper part of the symbol

	toBottom	integer	The pixels from the reference point to the lower part of the symbol
	toLeft	integer	The pixels from the reference point to the left side of the symbol
	toRight	integer	The pixels form the reference point to the right side of the symbol
	dx	integer	The delta x in pixels from the reference point to the printing position (usually negative)
	dy	integer	The delta y in pixels from the reference point to the printing position (usually negative)
Used by	<groupDef> <fontMapDef>		
Attributes	**Name**	**Type**	**Description**
	name	NMTOKEN	The name of the symbol

Source

```
<xs:element name="symbolDef" type="symbolDefType"/>
<xs:complexType name="symbolDefType">
   <xs:sequence>
      <xs:element name="code" type="xs:positiveInteger"/>
      <xs:element name="dimension" minOccurs="0">
         <xs:complexType>
            <xs:all>
               <xs:element name="toTop" type="xs:integer"/>
               <xs:element name="toBottom" type="xs:integer"/>
               <xs:element name="toLeft" type="xs:integer"/>
               <xs:element name="toRight" type="xs:integer"/>
               <xs:element name="dx" type="xs:integer"/>
               <xs:element name="dy" type="xs:integer"/>
            </xs:all>
         </xs:complexType>
      </xs:element>
   </xs:sequence>
   <xs:attribute name="name" type="xs:NMTOKEN" use="required"/>
</xs:complexType>
```

10.4.4.1 Symbols Definition Table

For the table of symbols used from font files see the normative digital annex **SMR_font_table**. Please note that some of the symbols are created by direct drawing such as staff lines, slurs, etc.

10.4.4.2 Example

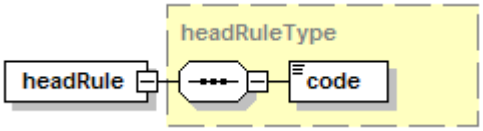
```
<groupDef name="custom1">
   <symbolDef name="sym1">
      <code>123</code>
      <dimension>
         <toTop>20</toTop>
         <toBottom>20</toBottom>
         <toLeft>20</toLeft>
         <toRight>20</toRight>
         <dx>-10</dx>
```

```
         <dy>-10</dy>
      </dimesion>
   </symbolDef>
   <symbolDef name="sym2">
      <code>124</code>
      <dimension>
         <toTop>50</toTop>
         <toBottom>20</toBottom>
         <toLeft>20</toLeft>
         <toRight>20</toRight>
         <dx>-10</dx>
         <dy>-10</dy>
      </dimesion>
   </symbolDef>
</groupDef>
```

10.4.5 <headRule>

The HEADRULE element represents the rule used when the note head type has to be set. It sets the character code to use for the note head symbol in the standard music font.

Table 109 — HEADRULE element syntax

<table>
<tr><td>Diagram</td><td colspan="3"> 
</td></tr>
<tr><td rowspan="2">Children</td><td>Name</td><td>Type</td><td>Description</td></tr>
<tr><td>Code</td><td>integer</td><td>The ASCII code to use for the note head symbol</td></tr>
<tr><td>Used by</td><td colspan="3"><SMFL></td></tr>
<tr><td rowspan="2">Attributes</td><td>Name</td><td>Type</td><td>Description</td></tr>
<tr><td>ruleId</td><td>ID</td><td>Rule identificator</td></tr>
<tr><td>Source</td><td colspan="3"><pre>
<xs:element name="headRule" type="headRuleType"/>
<xs:complexType name="headRuleType">
   <xs:sequence>
      <xs:element name="code" type="xs:integer"/>
   </xs:sequence>
   <xs:attribute name="ruleId" type="xs:ID" use="required"/>
</xs:complexType>
</pre></td></tr>
</table>

10.4.5.1 Example

```
<headRule ruleId="classicBlack">
   <code>123</code>
</headRule>
<headRule ruleId="classicWhite">
   <code>124</code>
</headRule>
<headRule ruleId="X">
   <code>156</code>
</headRule>
```

10.4.6 <stemLengthRule>

The STEMLENGTHRULE element represents the rule used to set the length of the note's stem. It provides the length of the stem as a number of spaces (distance between two staff lines) or as proportional to the note height.

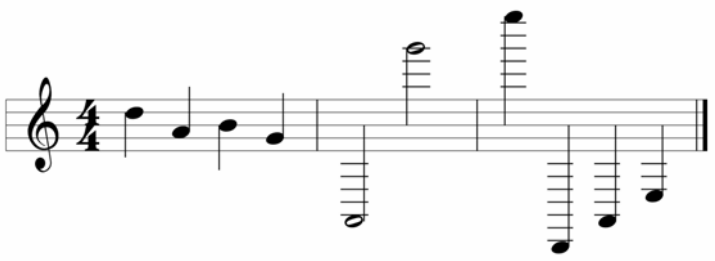
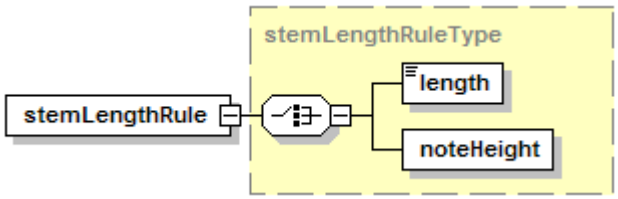


Figure 17 — STEM LENGH PARAMETER EXAMPLES

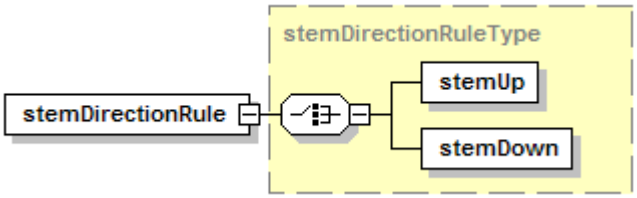
Table 110 — STEMLENGTHRULE element syntax

Diagram			
Children	**Name**	**Type**	**Description**
	length	decimal	Length of the stem in spaces
	noteHeight	none	Indicates to use a stem length proportional to the note height
Used by	<SMFL>		
Attributes	**Name**	**Type**	**Description**
	ruleId	ID	Rule identificator
Source	<xs:element name="stemLengthRule" type="stemLengthRuleType"/> <xs:complexType name="stemLengthRuleType"> <xs:choice> <xs:element name="length" type="xs:decimal"/> <xs:element name="noteHeight"/> </xs:choice> <xs:attribute name="ruleId" type="xs:ID" use="required"/> </xs:complexType>		

10.4.7 <stemDirectionRule>

The STEMDIRECTIONRULE element represents the rule used to set the direction of the note stem (up or down).

Table 111 — STEMDIRECTIONRULE element syntax

<table>
<tr><td>Diagram</td><td colspan="3"> 
</td></tr>
<tr><td rowspan="3">Children</td><td>Name</td><td>Type</td><td>Description</td></tr>
<tr><td>stemUp</td><td>none</td><td>Indicates to use a stem up</td></tr>
<tr><td>stemDown</td><td>none</td><td>Indicates to use a stem down</td></tr>
<tr><td>Used by</td><td colspan="3"><SMFL></td></tr>
<tr><td rowspan="2">Attributes</td><td>Name</td><td>Type</td><td>Description</td></tr>
<tr><td>ruleId</td><td>ID</td><td>Rule identificator</td></tr>
<tr><td>Source</td><td colspan="3"><pre>
<xs:element name="stemDirectionRule" type="stemDirectionRuleType"/>
<xs:complexType name="stemDirectionRuleType">
  <xs:choice>
    <xs:element name="stemUp"/>
    <xs:element name="stemDown"/>
  </xs:choice>
  <xs:attribute name="ruleId" type="xs:ID" use="required"/>
</xs:complexType>
</pre></td></tr>
</table>

10.4.8 <stemStartRule>

The STEMSTARTRULE element represents the rule used to set where the stem begins with respect to the note head. The ALIGNX and ALIGNY elements indicate the position from where the stem begins.

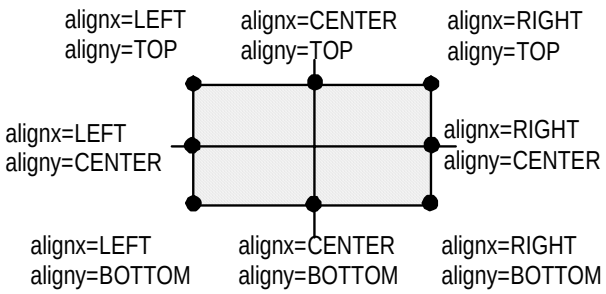
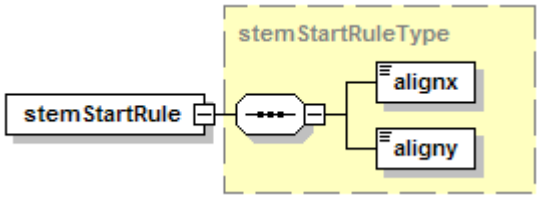


Table 112 — STEMSTARTRULE element syntax

Diagram			
Children	**Name**	**Type**	**Description**
	alignx	enum	One of LEFT, CENTER or RIGHT
	aligny	enum	One of TOP, CENTER, BOTTOM
Used by	<SMFL>		
Attributes	**Name**	**Type**	**Description**
	ruleId	ID	Rule identificator

Source

```
<xs:element name="stemStartRule" type="stemStartRuleType"/>
<xs:complexType name="stemStartRuleType">
   <xs:sequence>
      <xs:element name="alignx">
         <xs:simpleType>
            <xs:restriction base="xs:NMTOKEN">
               <xs:enumeration value="LEFT"/>
               <xs:enumeration value="CENTER"/>
               <xs:enumeration value="RIGHT"/>
            </xs:restriction>
         </xs:simpleType>
      </xs:element>
      <xs:element name="aligny">
         <xs:simpleType>
            <xs:restriction base="xs:NMTOKEN">
               <xs:enumeration value="TOP"/>
               <xs:enumeration value="CENTER"/>
               <xs:enumeration value="BOTTOM"/>
            </xs:restriction>
         </xs:simpleType>
      </xs:element>
   </xs:sequence>
   <xs:attribute name="ruleId" type="xs:ID" use="required"/>
</xs:complexType>
```

10.4.9 <beamSlopeRule>

The BEAMSLOPERULE element represents the rule used to set the slope of a beam. It can be a fixed slope or a slope within a minimum and a maximum (included). The slope is represented with a decimal number between -1 and 1 (excluded),a slope equal to 0 is horizontal a slope equal to 0.5 is at 45 degrees upward and at -0.5 is at 45 degrees downward.

In case a slope interval is specified, a slope is calculated connecting with a line the centre of note heads of the first and of the last note in the beam.

If the calculated slope is greater than 0, then

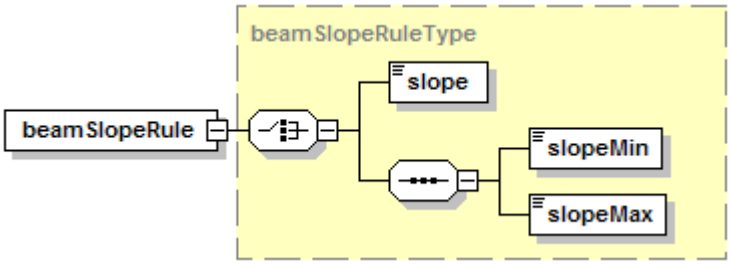
— If the calculated slope is within the interval [slopeMin, slopeMax] the calculated slope is used,

— if it is less than slopeMin the slopeMin slope is used and

— if it is over the slopeMax the maximum slope is used.

If the calculated slope is less than 0, the same procedure applies but with interval [-slopeMax, -slopeMin].

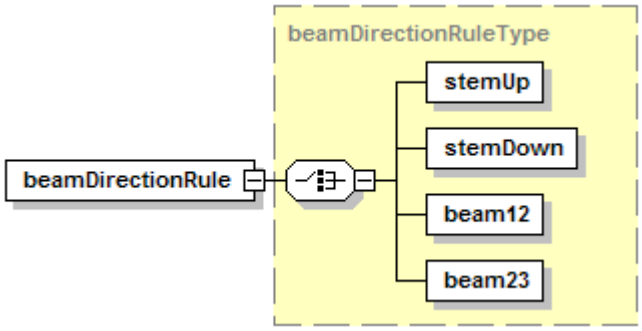
Table 113 — BEAMSLOPERULE element syntax

<table>
<tr><td>Diagram</td><td> 
</td></tr>
<tr><td rowspan="4">Children</td><td>Name</td><td>Type</td><td>Description</td></tr>
<tr><td>slope</td><td>decimal</td><td>the slope to be used for the beam</td></tr>
<tr><td>slopeMin</td><td>decimal</td><td>the minimum slope to use</td></tr>
<tr><td>slopeMax</td><td>decimal</td><td>the maximum slope to use</td></tr>
<tr><td>Used by</td><td colspan="3"><SMFL></td></tr>
<tr><td rowspan="2">Attributes</td><td>Name</td><td>Type</td><td>Description</td></tr>
<tr><td>ruleId</td><td>ID</td><td>Rule identificator</td></tr>
<tr><td>Source</td><td colspan="3"><pre>
<xs:element name="beamSlopeRule" type="beamSlopeRuleType"/>
<xs:complexType name="beamSlopeRuleType">
   <xs:choice>
      <xs:element name="slope" type="xs:decimal"/>
      <xs:sequence>
         <xs:element name="slopeMin" type="xs:decimal"/>
         <xs:element name="slopeMax" type="xs:decimal"/>
      </xs:sequence>
   </xs:choice>
   <xs:attribute name="ruleId" type="xs:ID" use="required"/>
</xs:complexType>
</pre></td></tr>
</table>

10.4.10 <beamDirectionRule>

The BEAMDIRECTIONRULE element represents the rule used to set the direction of the note stems in a beam. The stems can be up, down or in a way to position the beam between staff 1 and 2 (the upper staff and the mid/lower staff) or between staff 2 and 3 (the mid staff and the lower staff).

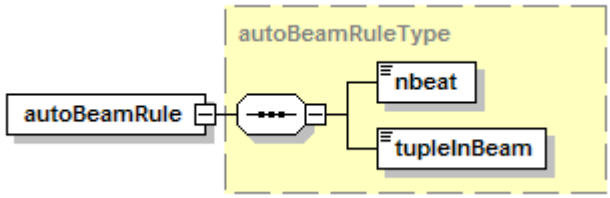
Table 114 — BEAMDIRECTIONRULE element syntax

<table>
<tr><td>Diagram</td><td colspan="3"> 
</td></tr>
<tr><td rowspan="5">Children</td><td>Name</td><td>Type</td><td>Description</td></tr>
<tr><td>stemUp</td><td>none</td><td>the beam has to have all stems up</td></tr>
<tr><td>stemDown</td><td>none</td><td>the beam has to have all stems down</td></tr>
<tr><td>beam12</td><td>none</td><td>the beam has to be between staff 1 and staff 2</td></tr>
<tr><td>beam23</td><td>none</td><td>the beam has to be between staff 2 and staff 3</td></tr>
<tr><td>Used by</td><td colspan="3"><SMFL></td></tr>
<tr><td rowspan="2">Attributes</td><td>Name</td><td>Type</td><td>Description</td></tr>
<tr><td>ruleId</td><td>ID</td><td>Rule identificator</td></tr>
<tr><td>Source</td><td colspan="3"><pre>
<xs:element name="beamDirectionRule" type="beamDirectionRuleType"/>
<xs:complexType name="beamDirectionRuleType">
  <xs:choice>
    <xs:element name="stemUp"/>
    <xs:element name="stemDown"/>
    <xs:element name="beam12"/>
    <xs:element name="beam23"/>
  </xs:choice>
  <xs:attribute name="ruleId" type="xs:ID" use="required"/>
</xs:complexType>
</pre></td></tr>
</table>

10.4.11 <autoBeamRule>

The AUTOBEAMRULE element represents the rule used for setting the automatic beaming. The rule defines the number of beats to present in the measure and if the notes in tuplets have to be beamed together regardless of the beats division or not.

Table 115 — AUTOBEAMRULE element syntax

<table>
<tr><td>Diagram</td><td colspan="3"> 
</td></tr>
<tr><td rowspan="3">Children</td><td>Name</td><td>Type</td><td>Description</td></tr>
<tr><td>nbeat</td><td>integer</td><td>the number of beats in which the measure has to be divided</td></tr>
<tr><td>tupleInBeam</td><td>boolean</td><td>indicates if the tuplets have to be beamed together or not.</td></tr>
<tr><td>Used by</td><td colspan="3"><SMFL></td></tr>
<tr><td rowspan="2">Attributes</td><td>Name</td><td>Type</td><td>Description</td></tr>
<tr><td>ruleId</td><td>ID</td><td>Rule identificator</td></tr>
<tr><td>Source</td><td colspan="3"><pre>
<xs:element name="autoBeamRule" type="autoBeamRuleType"/>
<xs:complexType name="autoBeamRuleType">
  <xs:sequence>
    <xs:element name="nbeat" type="xs:integer"/>
    <xs:element name="tupleInBeam" type="xs:boolean"/>
  </xs:sequence>
  <xs:attribute name="ruleId" type="xs:ID" use="required"/>
</xs:complexType>
</pre></td></tr>
</table>

10.4.12 <symbolDirectionRule>

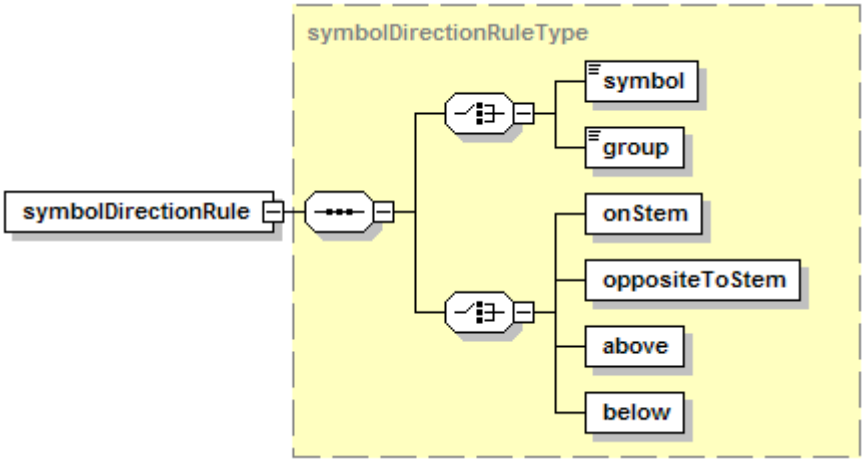
The SYMBOLDIRECTIONRULE element represents the rule used to indicate where a symbol has to be placed with respect to a note. The rule applies to a symbol or to a group of symbols. The symbol can be placed on the stem side, opposite to stem, above or below. The symbol can be a predefined symbol or a user defined symbol.

Table 116 — Predefined symbol names
(symbols names on the same row are synonym)

symbol name	description
CORONA, FERMATA	fermata symbol
GRUP, TURN	turn symbol
MORD, MORDENT	mordent symbol
TRILLO, TRILL	trill symbol
TREM, TREMOLO	tremolo symbol
STAC, STACCATO	staccato symbol
SFOR, SFORZATO	sforzato symbol
ACCE, ACCENTO	accento symbol
MARTF, MARTELLATO	martellato symbol
TEN, TENUTO	tenuto symbol
MART, MARTDOLCE	martellato dolce symbol
PUNTOALL, PUNTOALLUNGATO	punto allungato symbol
ARCATA	arcata symbol
PIZZICATO	pizzicato symbol

CORDA, STRING	string symbol
ARCATASU, BOWUP	bow up symbol
ARCATAGIU, BOWDOWN	bow down symbol
PUNTA	punta symbol
TALLONE	tallone symbol
PONTICELLO	ponticello symbol
TASTIERA	tastiera symbol
ARCO	arco symbol
DITEG, FINGER	fingering symbol
CONSORD, WITHMUTE	with mute symbol
VIASORD, WITHOUTMUTE	without mute symbol
KORN, KOREANORNAMENT	korean ornament symbol
ARM, HARM	harmonic symbol

Table 117 — SYMBOLDIRECTIONRULE element syntax

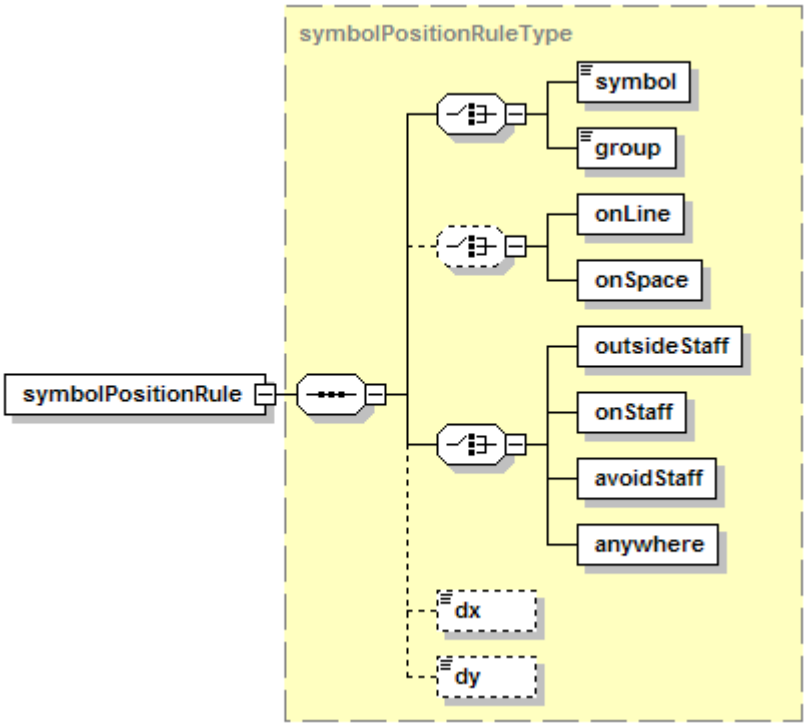
<table>
<tr><td>Diagram</td><td colspan="3"> 
</td></tr>
<tr><td rowspan="8">Children</td><td>Name</td><td>Type</td><td>Description</td></tr>
<tr><td>symbol</td><td>NMTOKEN</td><td>the symbol to which the rule applies to, it can be a predefined symbol name (see Table 116) or a user defined symbol name.</td></tr>
<tr><td>group</td><td>NMTOKEN</td><td>the group to which the rule applies to</td></tr>
<tr><td>on Stem</td><td>none</td><td>indicates to place the symbol from the side of the stem</td></tr>
<tr><td>oppositeToStem</td><td>none</td><td>indicates to place the symbol from the other side of the stem</td></tr>
<tr><td>above</td><td>none</td><td>indicates to place the symbol above the note</td></tr>
<tr><td>below</td><td>none</td><td>indicates to place the symbol below the note</td></tr>
<tr><td>Used by</td><td colspan="3"><SMFL></td></tr>
<tr><td rowspan="2">Attributes</td><td>Name</td><td>Type</td><td>Description</td></tr>
<tr><td>ruleId</td><td>ID</td><td>Rule identificator</td></tr>
</table>

Source	<xs:element name="symbolDirectionRule" type="symbolDirectionRuleType"/> <xs:complexType name="symbolDirectionRuleType"> <xs:sequence> <xs:choice> <xs:element name="symbol" type="xs:NMTOKEN"/> <xs:element name="group" type="xs:NMTOKEN"/> </xs:choice> <xs:choice> <xs:element name="onStem"/> <xs:element name="oppositeToStem"/> <xs:element name="above"/> <xs:element name="below"/> </xs:choice> </xs:sequence> <xs:attribute name="ruleId" type="xs:ID" use="required"/> </xs:complexType>

10.4.13 <symbolPositionRule>

The SYMBOLPOSITIONRULE element represents the rule used to set the position of a symbol or group of symbols in the staff. The symbol can be placed on a line or on a space; it can be placed outside the staff or on the staff or avoiding the staff or anywhere. Moreover a delta x and delta y can be used to change the default position

Table 118 — SYMBOLPOSITIONRULE element syntax

Diagram			
Children	**Name**	**Type**	**Description**
	symbol	NMTOKEN	the symbol to which the rule applies
	group	NMTOKEN	the group to which the rule applies

	onLine	none	the symbol has to be placed on the line
	onSpace	none	the symbol has to be placed on a space
	outsideStaff	none	the symbol has to be placed outside the staff
	onStaff	none	the symbol should be placed in the staff
	avoidStaff	none	the symbol should not be placed in the staff
	anywhere	none	the symbol can be placed anywhere
Used by	<SMFL>		
Attributes	**Name**	**Type**	**Description**
	ruleId	ID	Rule identificator

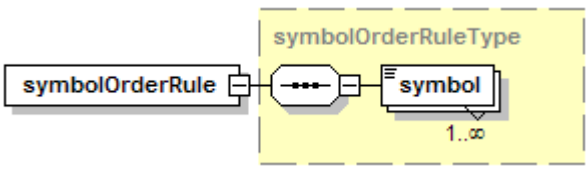
Source

```
<xs:element name="symbolPositionRule" type="symbolPositionRuleType"/>
<xs:complexType name="symbolPositionRuleType">
   <xs:sequence>
      <xs:choice>
         <xs:element name="symbol" type="xs:NMTOKEN"/>
         <xs:element name="group" type="xs:NMTOKEN"/>
      </xs:choice>
      <xs:choice minOccurs="0">
         <xs:element name="onLine"/>
         <xs:element name="onSpace"/>
      </xs:choice>
      <xs:choice>
         <xs:element name="outsideStaff"/>
         <xs:element name="onStaff"/>
         <xs:element name="avoidStaff"/>
         <xs:element name="anywhere"/>
      </xs:choice>
      <xs:element name="dx" type="xs:decimal" minOccurs="0"/>
      <xs:element name="dy" type="xs:decimal" minOccurs="0"/>
   </xs:sequence>
   <xs:attribute name="ruleId" type="xs:ID" use="required"/>
</xs:complexType>
```

10.4.14 <symbolOrderRule>

The SYMBOLORDERRULE element represents the rule used to set the order in which the symbols are positioned with respect to the note. Only one SYMBOLORDERRULE element has to be present. The symbols names can be the predefined ones and the user defined ones.

Table 119 — SYMBOLORDERRULE element syntax

Diagram	

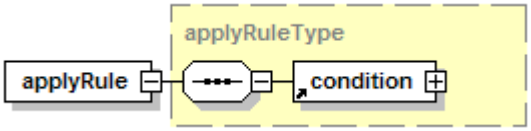
Children	**Name**	**Type**	**Description**
	symbol	NMTOKEN	the symbol name, it can be a predefined symbol name (see Table 116) or an user defined symbol name
Used by	<SMFL>		
Attributes	**Name**	**Type**	**Description**
	ruleId	ID	Rule identificator

Source	`<xs:element name="symbolOrderRule" type="symbolOrderRuleType"/>`
	`<xs:complexType name="symbolOrderRuleType">`
	`<xs:sequence>`
	`<xs:element name="symbol" type="xs:NMTOKEN" maxOccurs="unbounded"/>`
	`</xs:sequence>`
	`</xs:complexType>`		

10.4.15 <applyRule>

The APPLYRULE element is used to state the condition under which a rule has to be applied.

Table 120 — APPLYRULE element syntax

Diagram			
Children	<condition>		
Used by	<SMFL>		
Attributes	**Name**	**Type**	**Description**
	rule	IDREF	Reference to the rule to apply when the condition is verified

Source	`<xs:element name="applyRule" type="applyRuleType"/>`
	`<xs:complexType name="applyRuleType">`
	`<xs:sequence>`
	`<xs:element ref="condition"/>`
	`</xs:sequence>`
	`<xs:attribute name="rule" type="xs:IDREF" use="required"/>`
	`</xs:complexType>`		

10.4.16 <condition>

The CONDITION element is used to state all the condition terms that have to be verified in order to apply a rule. It can contain any number of sub conditions and it can contain:

— A NOTE element for conditions on notes,

— A CHORD element for conditions on chords,

— A REST element for conditions on rests,

— A SYMBOL element for conditions on symbols,

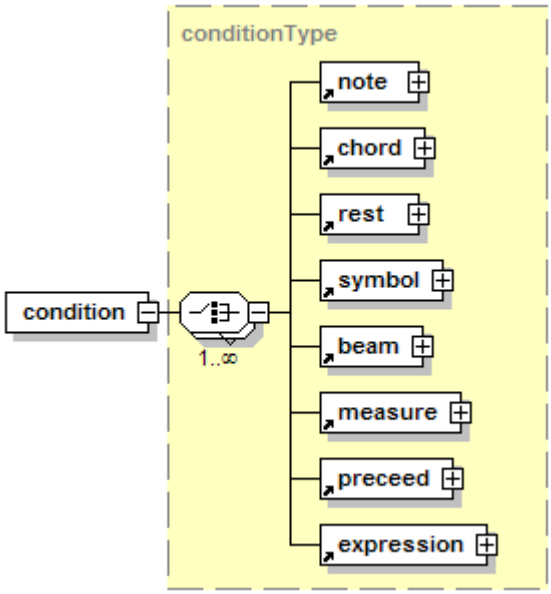
— A BEAM element for conditions on beams,

— A MEASURE element for conditions on measures,

— A PRECEED element to check ordering constraints,

— An EXPRESSION element for generic arithmetical expressions.

Table 121 — CONDITION element syntax

Diagram	
Children	<note> <chord> <rest> <symbol> <beam> <measure> <preceed> <expression>

Source	`<xs:element name="condition" type="conditionType"/>`
	`<xs:complexType name="conditionType">`
	`<xs:choice maxOccurs="unbounded">`
	`<xs:element ref="note"/>`
	`<xs:element ref="chord"/>`
	`<xs:element ref="rest"/>`
	`<xs:element ref="symbol"/>`
	`<xs:element ref="beam"/>`
	`<xs:element ref="measure"/>`
	`<xs:element ref="preceed"/>`
	`<xs:element ref="expression"/>`
	`</xs:choice>`
	`</xs:complexType>`

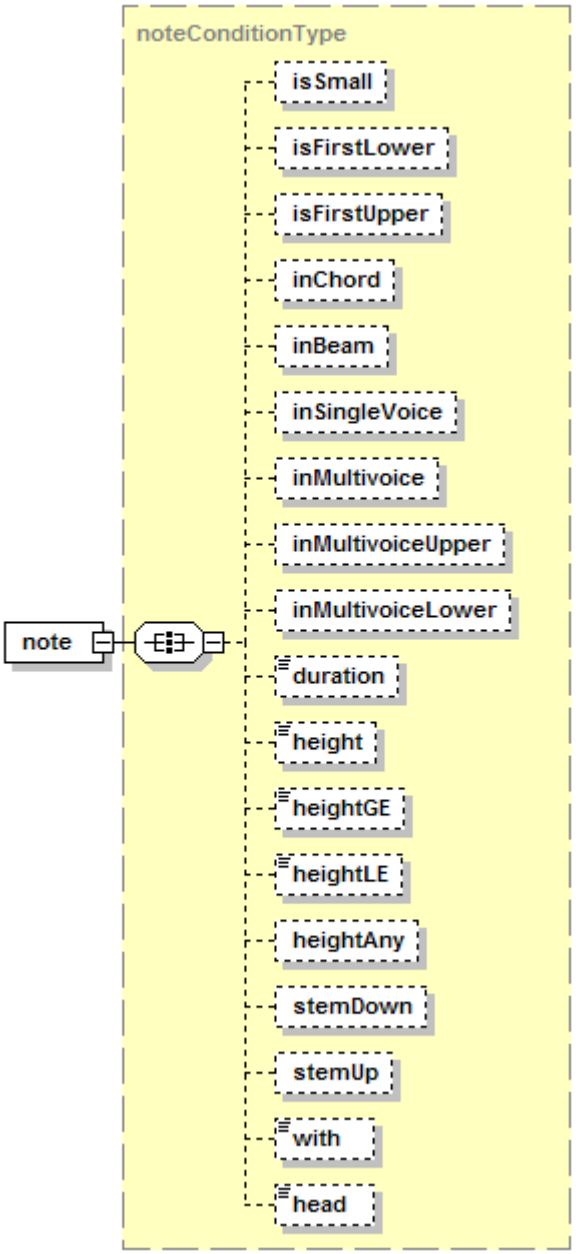
10.4.17 <note>

A NOTE element is used to express conditions on a note: it can contain elements expressing conditions on characteristics of the note that should be verified; thus the conditions expressed are all in AND.

Table 122 — head types (multiple names on the same line are synonymous)

head type
CLASSIC
ALPHANUM
ALPHANUM_SQUARE
ALPHANUM_REVERSE
CIRCLEX
X, X_HEAD
DDIESIS, DSHARP
DIAMOND
CLUSTER
DOWNTRIANG
LEFTTRIANG
RIGHTTRIANG
SQUARE
TRIANG
UPTRIANGROUND
DOWNTRIANGROUND
UPTRIANG
CROIX
MOON
PLUS, PLUS_HEAD
RHYTHMIC
THINRYHTMIC
PARENTHESIS
EMPTY
DIDAPTIC

Table 123 — NOTE element syntax

Diagram			
Children	**Name**	**Type**	**Description**
	isSmall	none	true when the note is small
	isFirstLower	none	true when the note is in a beam and the note is the first lower note
	isFirstUpper	none	true when the note is in a beam and the note is the first upper note
	inChord	none	true when the note is in a chord
	inBeam	none	true when the note is in a beam

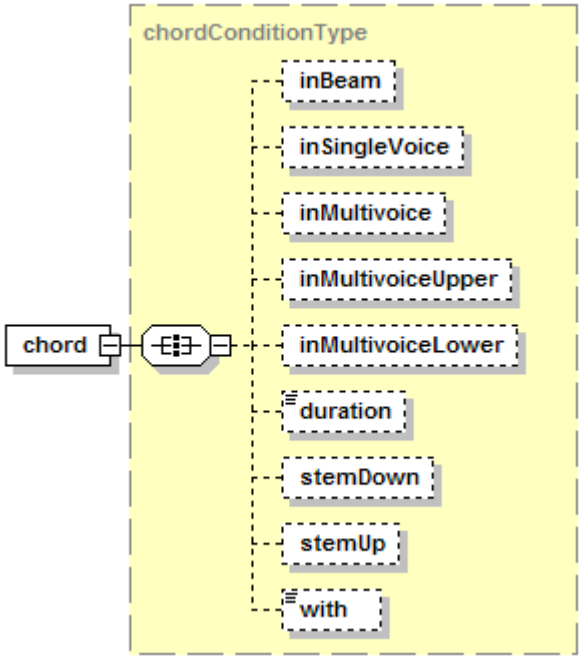
	inSingleVoice	none	true when the note is in a single voice measure
	inMultiVoice	none	true when the note is in multi voice measure
	inMultivoiceUpper	none	true when the note is in the upper voice
	inMultivoiceLower	none	true when the note is in the lower voice
	duration	enum	true when the note is of the duration specified
	height	integer	true when the note has the height specified
	heightGE	integer	true when the note has the height greater or equal to the one specified
	heightLE	integer	true when the note has a height lower or equal to the one specified
	heightAny	none	true when the note has any height
	stemDown	none	true when the note has the stem down
	stemUp	none	true when the note has the stem up
	with	NMTOKEN	true when the note has the symbol specified
	head	MNTOKEN	true when the note has the head type specified
Used by	<condition>		
Source			

```
<xs:element name="note" type="noteConditionType"/>
<xs:complexType name="noteConditionType">
   <xs:all>
      <xs:element name="isSmall" minOccurs="0"/>
      <xs:element name="isFirstLower" minOccurs="0"/>
      <xs:element name="isFirstUpper" minOccurs="0"/>
      <xs:element name="inChord" minOccurs="0"/>
      <xs:element name="inBeam" minOccurs="0"/>
      <xs:element name="inSingleVoice" minOccurs="0"/>
      <xs:element name="inMultivoice" minOccurs="0"/>
      <xs:element name="inMultivoiceUpper" minOccurs="0"/>
      <xs:element name="inMultivoiceLower" minOccurs="0"/>
      <xs:element name="duration" type="durationType" minOccurs="0"/>
      <xs:element name="height" type="xs:integer" minOccurs="0"/>
      <xs:element name="heightGE" type="xs:integer" minOccurs="0"/>
      <xs:element name="heightLE" type="xs:integer" minOccurs="0"/>
      <xs:element name="heightAny" minOccurs="0"/>
      <xs:element name="stemDown" minOccurs="0"/>
      <xs:element name="stemUp" minOccurs="0"/>
      <xs:element name="with" type="xs:NMTOKEN" minOccurs="0"/>
      <xs:element name="head" type="xs:NMTOKEN" minOccurs="0"/>
   </xs:all>
</xs:complexType>
```

10.4.18 <chord>

A CHORD element is used to express conditions on a chord: it can contain elements expressing conditions on characteristics of the chord that should be verified; thus the conditions expressed are all in AND.

Table 124 — CHORD element syntax

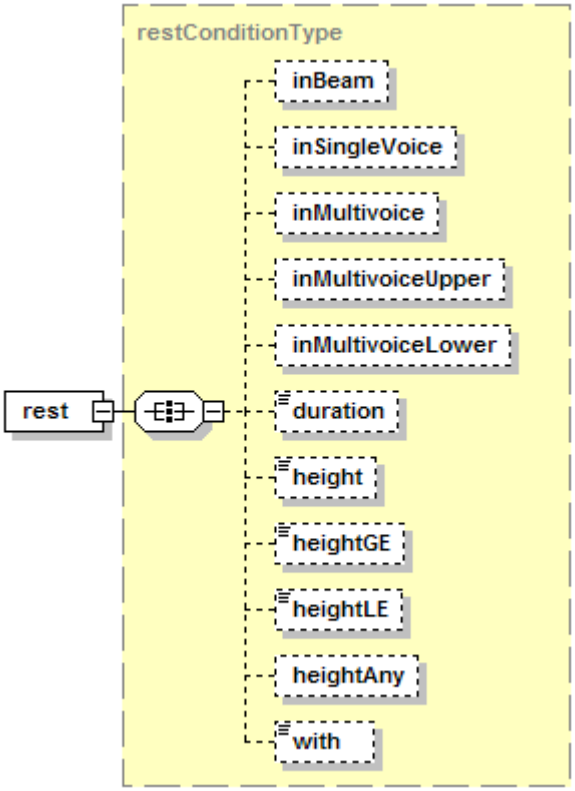
<table>
<tr><td>Diagram</td><td colspan="3"> 
</td></tr>
<tr><td rowspan="11">Children</td><td>Name</td><td>Type</td><td>Description</td></tr>
<tr><td>inBeam</td><td>none</td><td>true when the chord is in a beam</td></tr>
<tr><td>inSingleVoice</td><td>none</td><td>true when the chord is in a single voice measure</td></tr>
<tr><td>inMultivoice</td><td>none</td><td>true when the chord is in a multi voice measure</td></tr>
<tr><td>inMultivoiceUpper</td><td>none</td><td>true when the chord is in the upper multi voice</td></tr>
<tr><td>inMultivoiceLower</td><td>none</td><td>true when the chord is in the lower multi voice</td></tr>
<tr><td>duration</td><td>enum</td><td>true when the chord has the duration specified</td></tr>
<tr><td>stemDown</td><td>none</td><td>true when the chord has the stem down</td></tr>
<tr><td>stemUp</td><td>none</td><td>true when the chord has the stem up</td></tr>
<tr><td>with</td><td>NMTOKEN</td><td>true when the chord has the specified symbol</td></tr>
<tr><td>Used by</td><td colspan="3"><condition></td></tr>
<tr><td>Source</td><td colspan="3"><pre>
<xs:element name="chord" type="chordConditionType"/>
<xs:complexType name="chordConditionType">
  <xs:all>
    <xs:element name="inBeam" minOccurs="0"/>
    <xs:element name="inSingleVoice" minOccurs="0"/>
</pre></td></tr>
</table>

```
        <xs:element name="inMultivoice" minOccurs="0"/>
        <xs:element name="inMultivoiceUpper" minOccurs="0"/>
        <xs:element name="inMultivoiceLower" minOccurs="0"/>
        <xs:element name="duration" type="durationType" minOccurs="0"/>
        <xs:element name="stemDown" minOccurs="0"/>
        <xs:element name="stemUp" minOccurs="0"/>
        <xs:element name="with" type="xs:NMTOKEN" minOccurs="0"/>
    </xs:all>
  </xs:complexType>
```

10.4.19 <rest>

A REST element is used to express conditions on a rest: it can contain elements expressing conditions on characteristics of the rest that should be verified; thus the conditions expressed are all in AND.

Table 125 — REST element syntax

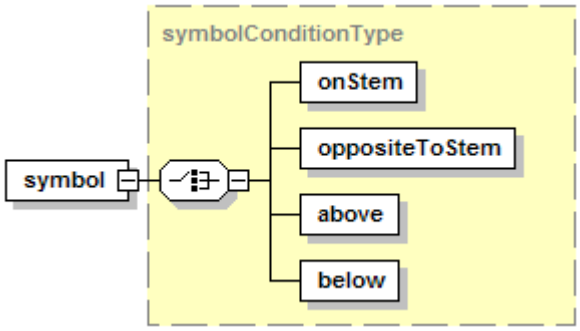
Diagram			
Children	**Name**	**Type**	**Description**
	inBeam	none	true when the rest is in a beam
	inSingleVoice	none	true when the rest is in a single voice measure
	inMultivoice	none	true when the rest is in a multi voice measure
	inMultivoiceUpper	none	true when the rest is in the upper multi voice

	inMultivoiceLower	none	true when the rest is in the lower multi voice
	duration	enum	true when the rest has the duration specified
	height	integer	true when the rest has the height specified
	heightGE	integer	true when the rest has the height greater or equal to the one specified
	heightLE	integer	true when the rest has a height lower or equal to the one specified
	heightAny	none	true when the rest has any height
	with	NMTOKEN	true when the rest has the specified symbol
Used by	<condition>		
Source	<xs:element name="rest" type="restConditionType"/> <xs:complexType name="restConditionType"> <xs:all> <xs:element name="inBeam" minOccurs="0"/> <xs:element name="inSingleVoice" minOccurs="0"/> <xs:element name="inMultivoice" minOccurs="0"/> <xs:element name="inMultivoiceUpper" minOccurs="0"/> <xs:element name="inMultivoiceLower" minOccurs="0"/> <xs:element name="duration" type="durationType" minOccurs="0"/> <xs:element name="height" type="xs:integer" minOccurs="0"/> <xs:element name="heightGE" type="xs:integer" minOccurs="0"/> <xs:element name="heightLE" type="xs:integer" minOccurs="0"/> <xs:element name="heightAny" minOccurs="0"/> <xs:element name="with" type="xs:NMTOKEN" minOccurs="0"/> </xs:all> </xs:complexType>		

10.4.20 <symbol>

A SYMBOL element is used to express conditions on a symbol: it can contain elements expressing conditions on characteristics of the symbol that should be verified; thus the conditions expressed are all in AND.

Table 126 — SYMBOL element syntax

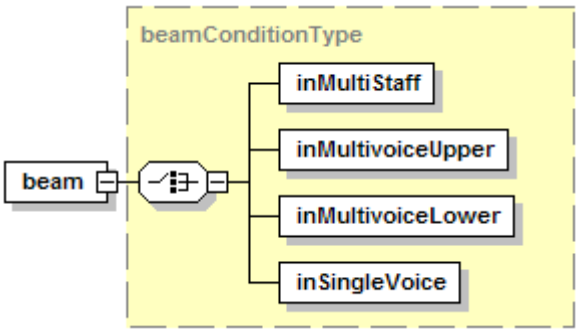
Diagram			
Children	**Name**	**Type**	**Description**

	onStem	none	true when the symbol is on the side of the stem
	oppositeToStem	none	true when the symbol is on the opposite side of the stem
	above	none	true when the symbol is above
	below	none	true when the symbol is below
Used by	<condition>		
Source	<xs:element name="symbol" type="symbolConditionType"/> <xs:complexType name="symbolConditionType"> <xs:choice> <xs:element name="onStem"/> <xs:element name="oppositeToStem"/> <xs:element name="above"/> <xs:element name="below"/> </xs:choice> </xs:complexType>		

10.4.21 <beam>

A BEAM element is used to express conditions on a beam: it can contain elements expressing conditions on characteristics of the beam that should be verified; thus the conditions expressed are all in AND.

Table 127 — BEAM element syntax

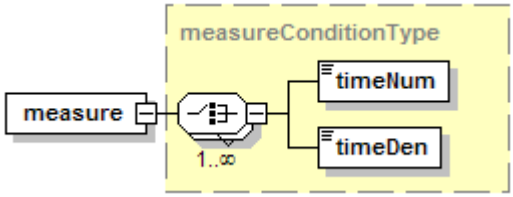
Diagram			
Children	**Name**	**Type**	**Description**
	inMultiStaff	none	true when the beam is in a multistaff measure
	inMultivoiceUpper	none	true when the beam is in the upper voice
	inMultivoiceLower	none	true when the beam is in the lower voice
	inSingleVoice	none	true when the beam is in a single voice measure
Used by	<condition>		

Source	<xs:element name="beam" type="beamConditionType"/> <xs:complexType name="beamConditionType"> <xs:choice> <xs:element name="inMultiStaff"/> <xs:element name="inMultivoiceUpper"/> <xs:element name="inMultivoiceLower"/> <xs:element name="inSingleVoice"/> </xs:choice> </xs:complexType>

10.4.22 <measure>

A MEASURE element is used to express conditions on a measure: it can contain elements expressing conditions on characteristics of the measure that should be verified; thus the conditions expressed are all in AND.

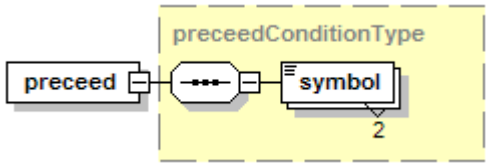
Table 128 — MEASURE element syntax

Diagram			
Attributes	**Name**	**Type**	**Description**
	timeNum	nonNegative Integer	true when the numerator of the time signature is the one specified
	timeDen	nonNegative Integer	true when the denominator of the time signature is the one specified
Used by	<condition>		
Source	<xs:element name="measure" type="measureConditionType"/> <xs:complexType name="measureConditionType"> <xs:choice maxOccurs="unbounded"> <xs:element name="timeNum" type="xs:nonNegativeInteger"/> <xs:element name="timeDen" type="xs:nonNegativeInteger"/> </xs:choice> </xs:complexType>		

10.4.23 <preceed>

A PRECEED element is used to express a condition on the order of symbols. The condition expressed is true when the first symbol provided preceeds the second one in the symbol order rule.

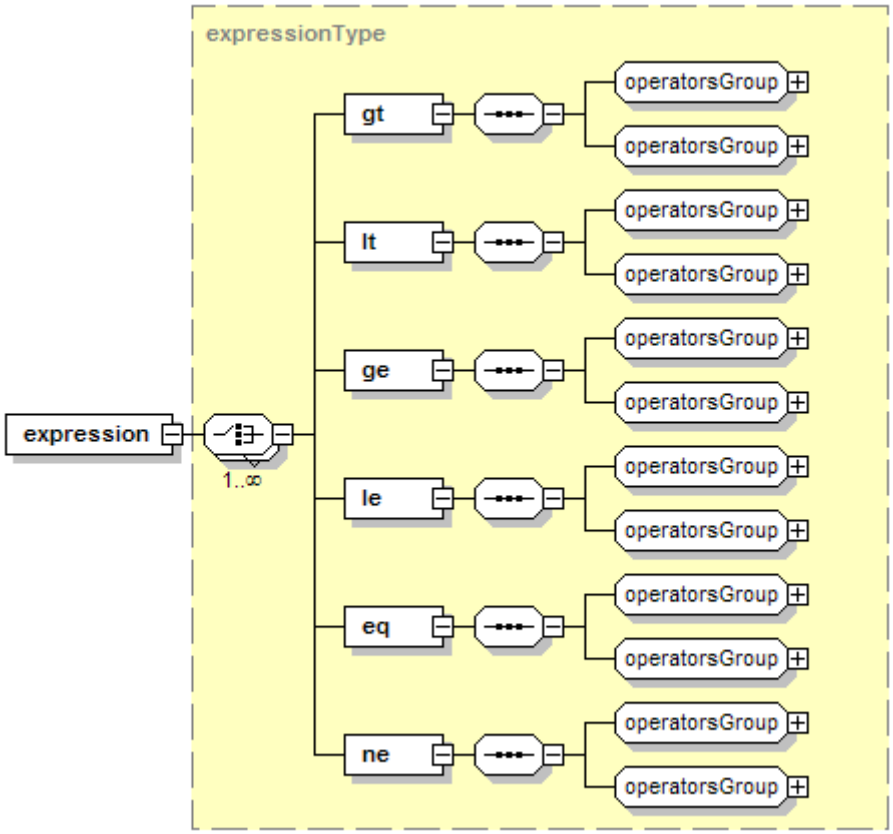
Table 129 — PRECEED element syntax

Diagram			
Children	**Name**	**Type**	**Description**
	symbol	NMTOKEN	the symbols to be checked, symbol names can be predefined (see Table 116) or user defined names.
Used by	<condition>		
Source	<xs:element name="preceed" type="preceedConditionType"/> <xs:complexType name="preceedConditionType"> <xs:sequence> <xs:element name="symbol" type="xs:NMTOKEN" minOccurs="2" maxOccurs="2"/> </xs:sequence> </xs:complexType>		

10.4.24 <expression>

An EXPRESSION element is used to state a relational condition on various parameters and values: the parameters and values can be connected with plus and minus operators. The expression can contain any number of relational conditions which have to be considered in AND.

Table 130 — EXPRESSION element syntax

<table>
<tr><td>Diagram</td><td> 
</td></tr>
<tr><td rowspan="7">Children</td><td>Name</td><td>Description</td></tr>
<tr><td>gt</td><td>condition expressing that the first child element has to be greater than the second one</td></tr>
<tr><td>lt</td><td>condition expressing that the first child element has to be less than the second one</td></tr>
<tr><td>ge</td><td>condition expressing that the first child element has to be grater or equal than the second one</td></tr>
<tr><td>le</td><td>condition expressing that the first child element has to be less or equal than the second one</td></tr>
<tr><td>eq</td><td>condition expressing that the first child element has to be equal to the second one</td></tr>
<tr><td>ne</td><td>condition expressing that the first child element has to be not equal to the second one</td></tr>
<tr><td>Used by</td><td colspan="2"><condition></td></tr>
<tr><td>Source</td><td colspan="2"><pre>
<xs:element name="expression" type="expressionType"/>
<xs:complexType name="expressionType">
  <xs:choice maxOccurs="unbounded">
    <xs:element name="gt">
      <xs:complexType>
        <xs:sequence>
</pre></td></tr>
</table>

```
                <xs:group ref="operatorsGroup"/>
                <xs:group ref="operatorsGroup"/>
             </xs:sequence>
          </xs:complexType>
       </xs:element>
       <xs:element name="lt">
          <xs:complexType>
             <xs:sequence>
                <xs:group ref="operatorsGroup"/>
                <xs:group ref="operatorsGroup"/>
             </xs:sequence>
          </xs:complexType>
       </xs:element>
       <xs:element name="ge">
          <xs:complexType>
             <xs:sequence>
                <xs:group ref="operatorsGroup"/>
                <xs:group ref="operatorsGroup"/>
             </xs:sequence>
          </xs:complexType>
       </xs:element>
       <xs:element name="le">
          <xs:complexType>
             <xs:sequence>
                <xs:group ref="operatorsGroup"/>
                <xs:group ref="operatorsGroup"/>
             </xs:sequence>
          </xs:complexType>
       </xs:element>
       <xs:element name="eq">
          <xs:complexType>
             <xs:sequence>
                <xs:group ref="operatorsGroup"/>
                <xs:group ref="operatorsGroup"/>
             </xs:sequence>
          </xs:complexType>
       </xs:element>
       <xs:element name="ne">
          <xs:complexType>
             <xs:sequence>
                <xs:group ref="operatorsGroup"/>
                <xs:group ref="operatorsGroup"/>
             </xs:sequence>
          </xs:complexType>
       </xs:element>
    </xs:choice>
 </xs:complexType>
```

10.4.25 operatorsGroup

The OPERATORSGROUP elements group is used to represent the operators of an expression.

They can be:

— The MINUS element which is used to make the difference between two sub expressions

— The PLUS element which is used to make the sum between two sub expressions

— The STAFFLINES element which has to be evaluated to the number of staff lines in the measure

— The STAFFNUMBER element which has to be evaluated to the number of staff of the note/rest/chord

— The BEAM.DIS element which has to be evaluated as:

$$\text{BEAM.DIS} = |Highest| - |Lowest|$$

Where *Highest* is the highest note in the beam and *Lowest* is the lower note in the beam

— The BEAM.DELTA element which has to be evaluated as:

$$\text{BEAM.DELTA} = \frac{\text{BEAM.DIS}}{Number\ of\ notes + 1}$$

— The BEAM.MEAN element which has to be evaluated as:

$$\text{BEAM.MEAN} = \frac{\sum note\ heights}{number\ of\ notes}$$

— The BEAM.MHL element which has to be evaluated as:

$$\text{BEAM.MHL} = \frac{Highest + Lowest}{2}$$

— The CHORD.UPPERD element which has to be evaluated as the absolute value of the height of the upper note in a chord

— The CHORD.LOWERD element which has to be evaluated as the absolute value of the lowest note in a chord

— The CHORD.NUMUP element which has to be evaluated as the number of notes in a chord strictly over the mid line of the staff

— The CHORD.NUMDOWN element which has to be evaluated as the number of notes in a chord strictly below the mid line of the staff

— The VALUE element which contains a single decimal value

When the beam or chord spans over more than one staff the height of the note when calculating BEAM.DIS, BEAM.DELTA, BEAM.MEAN, BEAM.MHL, CHORD.UPPERD, CHORD.LOWERD, CHORD.NUMUP and CHORD.NUMDOWN is assumed as:

$$AbsHeight = Height - 23 * Staff$$

where *Staff* is 0 for the upper staff, 1 for the mid staff or the lower staff in case of two staves, 2 for the lower staff in case of three staves. The mapping of the relative heights into absolute heights is depicted in Figure 18.

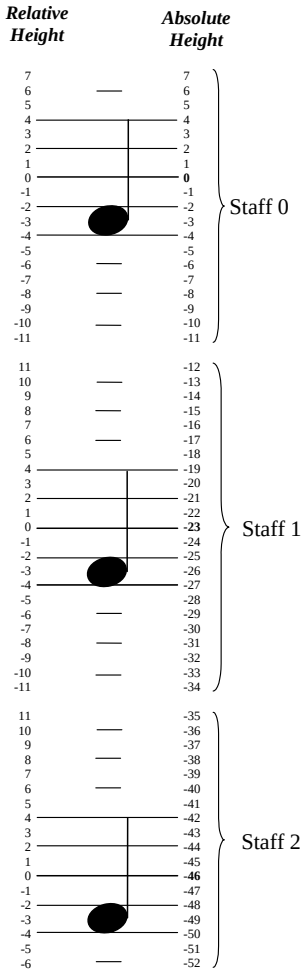
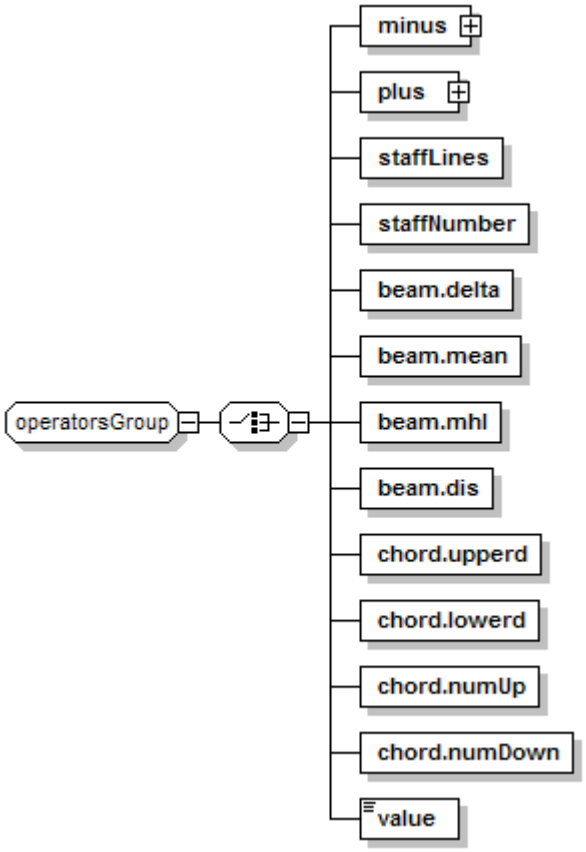


Figure 18 – Note height in multi staff measures

Table 131 — OPERATORSGROUP group syntax

Diagram	
Used by	<expression>
Source	<xs:group name="operatorsGroup"> <xs:choice> <xs:element name="minus"> <xs:complexType> <xs:sequence> <xs:group ref="operatorsGroup"/> <xs:group ref="operatorsGroup"/> </xs:sequence> </xs:complexType> </xs:element> <xs:element name="plus"> <xs:complexType> <xs:sequence> <xs:group ref="operatorsGroup"/> <xs:group ref="operatorsGroup"/> </xs:sequence> </xs:complexType> </xs:element> <xs:element name="staffLines"/> <xs:element name="staffNumber"/> <xs:element name="beam.delta"/> <xs:element name="beam.mean"/> <xs:element name="beam.mhl"/> <xs:element name="beam.dis"/> <xs:element name="chord.upperd"/> <xs:element name="chord.lowerd"/> <xs:element name="chord.numUp"/> <xs:element name="chord.numDown"/> <xs:element name="value" type="xs:decimal"/> </xs:choice>

	</xs:group>

10.5 SM-FL Examples

10.5.1 Insertion Rules and Conditions for note stem direction

For example, an insertion rule “StemUp”, applied to symbol “STEM” of the note, which sets the stem upward with respect to the notehead can be stated as:

```
<stemDirectionRule ruleId="StemUp">
   <stemUp/>
</stemDirectionRule>
```

A condition to activate this rule can be very simple. The condition could state that rule “StemUp” is applied if the note to be inserted is localized below the middle line of the staff:

```
<applyRule rule="StemUp">
   <condition>
      <note>
         <heightLE>-1</heightLE>
      </note>
   <condition>
</stemDirectionRule>
```

A different condition may state that rule “StemUp” is invoked if the note belongs to the upper voice for a measure where polyphony (InMultivoiceUpper) is present. The upper voice is the one presenting the note with the highest pitch among those to be played at the same time:

```
<applyRule rule="StemUp">
   <condition>
      <note>
         <inMultivoiceUpper/>
      </note>
   <condition>
</applyRule>
```

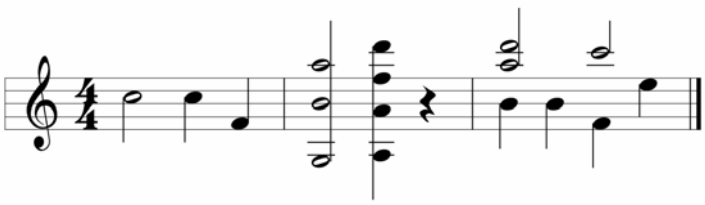


Figure 19 – The stem of notes and chords in single layer and in polyphony

Another case is when the note is in a single layer and is included in a chord:

```
<applyRule rule="StemUp">
   <condition>
      <note>
         <inSinglevoice/>
         <inChord/>
      </note>
      <expression>
         <lt>
```

```
            <minus><chord.upperd/><chord.lowerd/></minus>
            <value>0</value>
         </gt>
      </expression>
   <condition>
</applyRule>
```

Such conditions are met in the second measure of Figure 19. The notes belong to a chord (inChord), only one voice is present (inSinglevoice), and the difference between the highest and the lowest notes of the chord defines the "centre of gravity" of the chord, either above or below the middle line, that is (upperd-lowerd>0); where: upperd is the absolute value based on the distance between the highest note of the chord and the middle line of the staff, and lowerd is the absolute value based on the distance between the lowest note of the chord and the middle line of the staff.

10.5.2 Positioning Rules and Conditions for note stem length

The positioning rules of the stem affect its length. The basic unit for stem length is the *space* defined as the distance between two staff lines. In this way, the standard length of the stem is 3.5 spaces, but it has to assume different values depending on the note height. In Figure 17 the following rules and conditions have been used for some notes:

```
<stemLengthRule ruleId="Stem3_5">
   <length>3.5</length>
</stemLengthRule>
<stemLengthRule ruleId="StemHeight">
   <noteHeight/>
</stemLengthRule>

<applyRule rule="Stem3_5">
   <condition>
      <note>
         <heightGE>0</heightGE>
         <heightLE>7</heightLE>
         <stemDown/>
      </note>
   <condition>
</applyRule>

<applyRule rule="StemHeight">
   <condition>
      <note>
         <heightGE>8</heightGE>
         <stemDown/>
      </note>
   <condition>
</applyRule>
```

The first condition is verified for the first note of the first measure of Figure 20, while the second conditions is true for the second note of the second measure and for the first of the third measure which sets the use of a stem with a length equal to the note height divided by two.

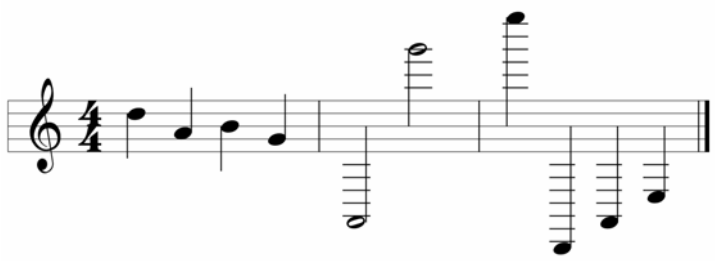


Figure 20 – The stem length for notes in single layer

The following condition takes into account the case of polyphony and it occurs, for example, in the first note of the first measure of Figure 21 in the upper voice.

```
<stemLengthRule ruleId="Stem2_5">
   <length>2.5</length>
</stemLengthRule>

<applyRule rule="Stem2_5">
   <condition>
      <note>
         <inMultivoiceUpper>
         <heightGE>1</heightGE>
      </note>
   <condition>
</applyRule>
```

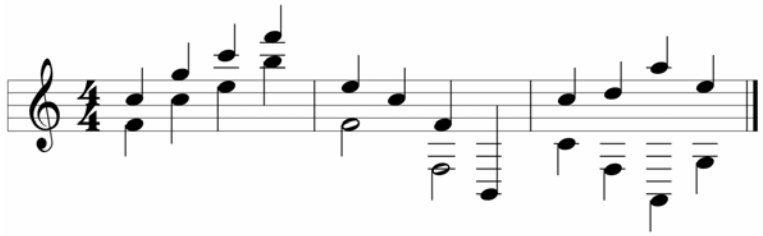


Figure 21 – The stem length for notes in polyphony

10.5.3 Insertion Rules and Conditions for beam orientation on a single staff

Insertion rules for beamed notes affect the direction of the note stems belonging to the beam. All the stems of the beamed notes have to share the same direction (except for some specific cases, depending on the presence of multistaff parts, for further explanation see next sections). The context parameters to be evaluated in order to determine the stem direction of the beam depend on the note heights belonging to the group. For example, the group in the second measure of Figure 22 satisfies the following condition:

```
<applyRule rule="Beam0">
      <condition>
         <beam>
            <inSingleVoice/>
         </beam>
         <expression>
            <eq>
               <staffNumber/>
               <value>1.0</value>
            </eq>
         </expression>
         <expression>
```

```
            <lt>
                <beam.mean/>
                <value>0.0</value>
            </lt>
        </expression>
    </condition>
</applyRule>
```

where `staffNumber =1` means that the beam has notes belonging to one staff, `beam.mean` is the mean value of the note height:

$$MEAN = \frac{\sum note\ heights}{number\ of\ notes}$$

This parameter is useful to evaluate where the notes belonging to the beam are located in terms of pitch; where:

- Σ note heights: is the summing the heights of all the notes of the group
- number of notes: is the number of notes in the group

In the 2nd and 4th measures below Figure 22 shows what happened after applying the following Insertion Rules for beaming,.

```
<beamDirectionRule ruleId="Beam0">
    <stemUp/>
</beamDirectionRule>
<beamDirectionRule ruleId="Beam1">
    <stemDown/>
</beamDirectionRule>
```

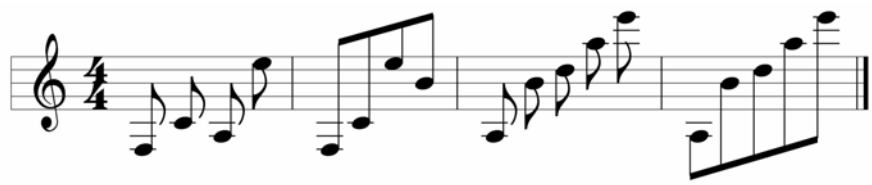


Figure 22 – The stem direction of a group of notes

10.5.4 Positioning Rules and Conditions for beam slope on a single staff

Positioning rules for beams are used to impose their slope. It typically ranges from 0 (horizontal beam) to 0.5 (maximum slope corresponding to a 45° angle).

Figure 20 presents beams with different slopes. The slopes are typically decided considering the values of parameters such as the difference between the heights of the highest and lowest notes of the beam. While estimating the conditions for the identification of the right rule for beam slope definition, the value of the status variable BEAMDIS can be used as well. BEAMDIS is estimated as the distance in terms of height/spaces between the position of the highest and lowest notes:

$$BEAMDIS = |Highest| - |Lowest|$$

where:

- Highest is the height of the highest note of the beam/group;

- Lowest is the height of the lowest note of the beam/group.

Heights are calculated assuming 0 for the midline of the staff/tablature. This parameter is useful to evaluate the dispersion of the pitch of the beam's notes around the central line of the staff/tablature.
For example, the following condition is typically used:

```
<applyRule rule="BeamMin">
       <condition>
          <expression>
             <eq>
                <staffNumber/>
                <value>1.0</value>
             </eq>
          </expression>
          <expression>
             <lt>
                <beam.delta/>
                <value>2.5</value>
             </lt>
          </expression>
       </condition>
    </applyRule>
```

where `beam.delta` is calculated by considering the number of notes composing the beam as:

$$DELTA = \frac{BEAMDIS}{Number\ of\ notes + 1}$$

This parameter is useful to evaluate the normalized dispersion of the pitch when considering the distance from the note with the highest pitch to the note with the lowest pitch and the number of notes belonging to the beam.

The above condition activates the following rule:

```
<beamSlopeRule ruleId="BeamMin">
    <slope>0.0</slope>
</beamSlopeRule>
```

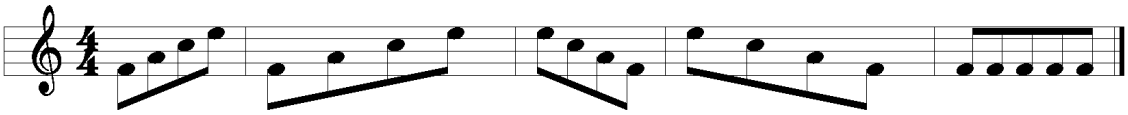


Figure 23 – Some beams with different slopes

The Positioning rule can be used to set the beam slope assigning a variable value to the `slope` parameter of the beam, furthermore the value can be constrained in a given interval:

```
<beamSlopeRule ruleId="BeamInt">
    <slopeMin>0.15</slopeMin>
    <slopeMax>0.3</slopeMax>
</beamSlopeRule>
```

The value of slope is calculated by dividing the difference between the Y coordinates of the first and last notes of the group and those on the X-axis. This value is estimated on the account of the decided justification parameter and line breaking. The slope value obtained is constrained in the interval: if it is less than the minimum, the minimum slope is assumed and if the value is greater than the maximum, the maximum slope is

assumed. However if the slope would lead to a note with a too short stem (less than 2 spaces for normal notes and less than 1 space for small notes) the horizontal beaming is used instead.

The beam slope is typically equal to 0 in presence of tablature or percussions:

```
<applyRule rule="BeamMin">
   <condition>
      <expression>
         <lt>
            <staffLines/>
            <value>5.0</value>
         </lt>
      </expression>
   </condition>
</applyRule>

<applyRule rule="BeamMin">
   <condition>
      <expression>
         <gt>
            <staffLines/>
            <value>5.0</value>
         </gt>
      </expression>
   </condition>
</applyRule>

<beamSlopeRule ruleId="BeamMin">
   <slope>0.0</slope>
</beamSlopeRule>
```

10.6 Rules and conditions for beams on multiple staves

Multistaff music scores (scores for piano, organ, harp, etc.) need more parameters than single staff music scores to assess their context. In case of multistaff parts, it is possible to beam notes, which are in different staves. To obtain the exact layout and slope desired, it is possible to write specific SM-FL conditions. It is possible to choose where to place the beams: in case of three-staves parts the beam can be placed above the staves, between the first and second staff, between the second and third staff or below the staves. SM-FL rules and conditions are specified for deciding both the position and the slope of the beam.

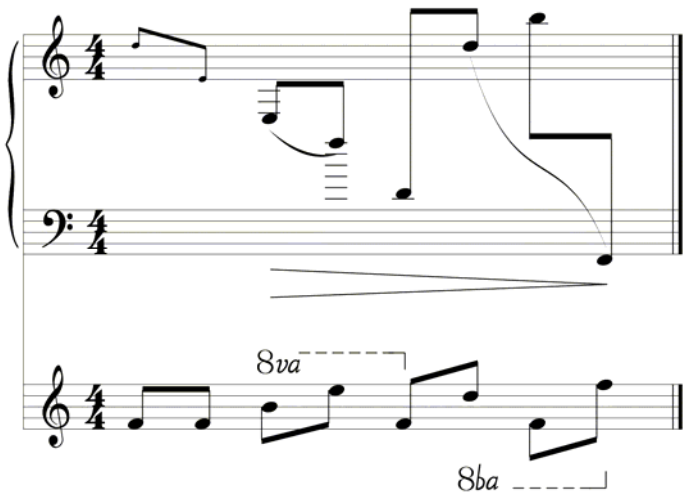


Figure 24 – A score with multistaff part

10.6.1 Insertion Rules and Conditions for beams in multistaff parts

What follows is an example on how these parameters can be used in SM-FL conditions:

```
<beamDirectionRule ruleId="Beam12">
   <beam12/>
</beamDirectionRule>
<beamDirectionRule ruleId="Beam23">
   <beam23/>
</beamDirectionRule>

<applyRule rule="Beam12">
   <condition>
      <beam>
         <inMultiStaff/>
      </beam>
      <expression>
         <le>
            <beam.dis/>
            <value>-20.0</value>
         </le>
      </expression>
      <expression>
         <gt>
            <beam.dis/>
            <value>-40.0</value>
         </gt>
      </expression>
   </condition>
</applyRule>
<applyRule rule="Beam23">
   <condition>
      <beam>
         <inMultiStaff/>
      </beam>
      <expression>
         <le>
            <beam.dis/>
            <value>-40.0</value>
         </le>
      </expression>
   </condition>
</applyRule>

<applyRule rule="BeamInt015">
   <condition>
      <expression>
         <gt>
            <beam.dis/>
            <value>-25.0</value>
         </gt>
      </expression>
      <expression>
         <gt>
            <minus>
               <beam.mean/>
               <beam.mhl/>
            </minus>
            <value>0.0</value>
         </gt>
      </expression>
```

```
    </condition>
</applyRule>

<applyRule rule="BeamIntM015">
    <condition>
        <expression>
            <gt>
                <beam.dis/>
                <value>-25.0</value>
            </gt>
        </expression>
        <expression>
            <le>
                <minus>
                    <beam.mean/>
                    <beam.mhl/>
                </minus>
                <value>0.0</value>
            </le>
        </expression>
    </condition>
</applyRule>
```

…

The condition which allows taking a decision about the rule for the beam slope setting can be defined by using the following value combined with `beam.mean`:

$$MHL = \frac{Highest + Lowest}{2}$$

This parameter is useful to evaluate the mean value of the pitch for the considered notes

10.6.2 Positioning Rules and Conditions for beams in multistaff parts

What follows is an example on how the parameters defined above can be used in SM-FL conditions:

```
<beamSlopeRule ruleId="BeamInt">
    <slopeMin>0.15</slopeMin>
    <slopeMax>0.3</slopeMax>
</beamSlopeRule>
<beamSlopeRule ruleId="BeamIntM007">
    <slopeMin>0.0</slopeMin>
    <slopeMax>-0.07</slopeMax>
</beamSlopeRule>
<beamSlopeRule ruleId="BeamInt007">
    <slopeMin>0.0</slopeMin>
    <slopeMax>0.07</slopeMax>
</beamSlopeRule>
```

…

```
<applyRule rule="BeamInt">
    <condition>
        <beam>
            <inMultiStaff/>
        </beam>
        <expression>
            <eq>
                <staffLines/>
```

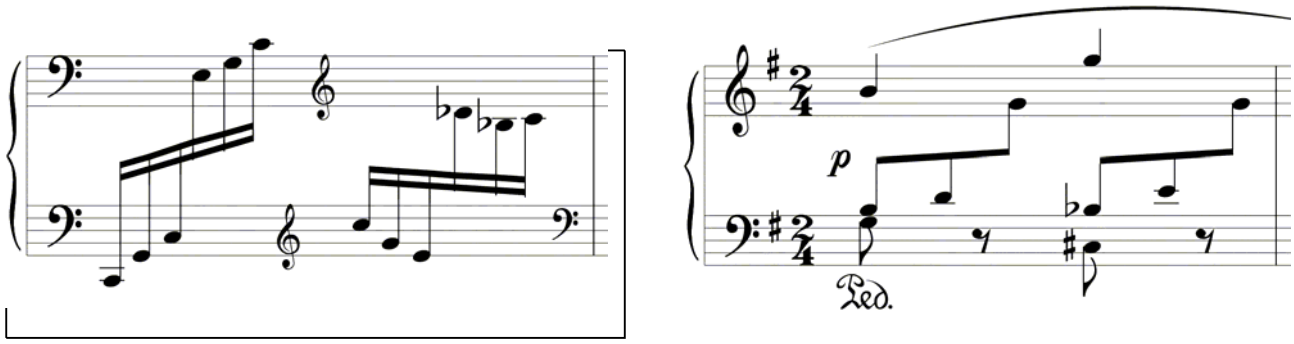
```
                <value>5.0</value>
            </eq>
        </expression>
        <expression>
            <ge>
                <beam.delta/>
                <value>-4.0</value>
            </ge>
        </expression>
        <expression>
            <le>
                <beam.delta/>
                <value>-3.0</value>
            </le>
        </expression>
        <expression>
            <eq>
                <beam.dis/>
                <value>-25.0</value>
            </eq>
        </expression>
    </condition>
</applyRule>

<applyRule rule="BeamIntM007">
    <condition>
        <beam>
            <inMultiStaff/>
        </beam>
        <expression>
            <eq>
                <staffLines/>
                <value>5.0</value>
            </eq>
        </expression>
        <expression>
            <eq>
                <beam.dis/>
                <value>-22.0</value>
            </eq>
        </expression>
    </condition>
</applyRule>

<applyRule rule="BeamInt007">
    <condition>
        <beam>
            <inMultiStaff/>
        </beam>
        <expression>
            <eq>
                <staffLines/>
                <value>5.0</value>
            </eq>
        </expression>
        <expression>
            <eq>
                <beam.delta/>
                <value>-4.0</value>
            </eq>
        </expression>
```

```
    </condition>
</applyRule>
```

…



Different slopes for beams **Slopes almost horizontal**

Figure 25 – Two measures with multistaff beams

10.6.3 Rules and conditions for automatic beaming

Depending on the time signature of each measure, the SM-FL engine automatically beams notes with hook/flag when a precise fraction of the time signature is reached during note insertion. For example, for the time signature of 4/4, the following condition can be imposed to activate rule "NumBeat4":

```
<applyRule rule="NumBeat4">
    <condition>
        <measure>
            <timeNum>4</timeNum>
            <timeDen>4</timeDen>
        </measure>
    </condition>
</applyRule>
```

The referred insertion rule imposes the division of the measure in 4 beams:

```
<autoBeamRule ruleId="NumBeat4">
    <nbeat>4</nbeat>
</autoBeamRule>
```

The system automatically beams a group of notes when a precise fraction of the time signature is reached.

The information about time signature is very useful. The time signature of the measure can vary during the music piece. In addition, the time signature could express a real value different from what is specified in the staff. For example a time signature of 6/8 can be intended as two irregular groups for a total duration of 2/4. The time written on the score is 6/8, the real duration time is 2/4. This information is provided by the user through a dialog box. Information about time signature and real duration time is necessary to compute effective duration of the notes for visualization and MIDI output.

The rules for the automatic beaming set the parameter defining how many beats the measure should contain. In this context Beat means the number of groups the system tries to make inside the measure.

```
<autoBeamRule ruleId="NumBeat1">
    <nbeat>1</nbeat>
    <tupleInBeam>false</tupleInBeam>
</autoBeamRule>
<autoBeamRule ruleId="NumBeat2">
```

```
    <nbeat>2</nbeat>
    <tupleInBeam>false</tupleInBeam>
</autoBeamRule>
<autoBeamRule ruleId="NumBeat3">
    <nbeat>3</nbeat>
    <tupleInBeam>false</tupleInBeam>
</autoBeamRule>
<autoBeamRule ruleId="NumBeat4">
    <nbeat>4</nbeat>
    <tupleInBeam>false</tupleInBeam>
</autoBeamRule>
<autoBeamRule ruleId="NumBeat5">
    <nbeat>5</nbeat>
    <tupleInBeam>false</tupleInBeam>
</autoBeamRule>
…

<applyRule rule="NumBeat1">
    <condition>
        <measure>
            <timeNum>3</timeNum>
            <timeDen>8</timeDen>
        </measure>
    </condition>
</applyRule>
<applyRule rule="NumBeat5">
    <condition>
        <measure>
            <timeNum>5</timeNum>
            <timeDen>8</timeDen>
        </measure>
    </condition>
</applyRule>
<applyRule rule="NumBeat2">
    <condition>
        <measure>
            <timeNum>6</timeNum>
            <timeDen>8</timeDen>
        </measure>
    </condition>
</applyRule>
<applyRule rule="NumBeat4">
    <condition>
        <measure>
            <timeNum>4</timeNum>
            <timeDen>8</timeDen>
        </measure>
    </condition>
</applyRule>
…

<applyRule rule="NumBeat5">
    <condition>
        <measure>
            <timeNum>15</timeNum>
            <timeDen>8</timeDen>
        </measure>
    </condition>
</applyRule>
<applyRule rule="NumBeat4">
    <condition>
```

```
        <measure>
            <timeNum>12</timeNum>
            <timeDen>16</timeDen>
        </measure>
    </condition>
</applyRule>
```

10.6.4 Insertion Rules and Conditions for Markers

Insertion rules are used to estimate the position of markers. Their position is typically related to the direction of the note stem, if any. For example, in the next figure the insertion rule for the staccato-dot has been used by the first note of the first measure:

```
<symbolDirectionRule ruleId="Stac0">
    <symbol>STAC</symbol>
    <oppositeToStem/>
</symbolDirectionRule>
```

It states that the staccato has to be drawn on the opposite side, with respect to the direction of the note stem. The opposite constraint is ONSTEM.

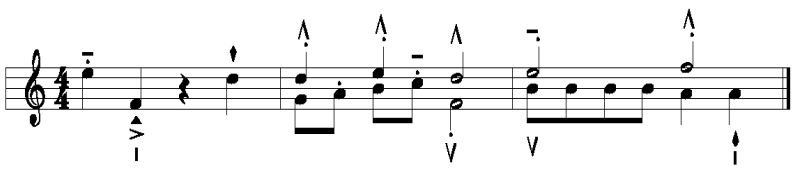


Figure 26 – Notes with markers

A condition activating the above rule is:

```
<applyRule rule="Stac0">
    <condition>
        <note>
            <inSingleVoice/>
        </note>
    </condition>
</applyRule>
```

10.6.5 Positioning Rules and Conditions for Markers

Positioning rules of these symbols are used to impose parameters stating specific relative positions:

- in the space between staff lines or on the staff line: ONSPACE, ONLINE;
- outside or inside the staff: OUTSIDESTAFF, ONSTAFF;
- at an estimated distance from the note stem or from the note head: DX, DY;
- above or below the staff: ABOVE, BELOW.

A composition of such constraints helps to specify the exact position of the symbols with respect to the note and the staff. For example, for the staccato dot associated with the first note of the first measure as shown in Figure 26, the following rule was applied:

```
<symbolPositionRule ruleId="Stac2">
    <symbol>STAC</symbol>
    <onSpace/>
```

```
    <outsideStaff/>
    <dx>0</dx>
    <dy>0</dy>
</symbolPositionRule>
```

The rule was activated by condition:

```
<applyRule rule="Stac2">
    <condition>
        <symbol>
            <oppositeToStem/>
        </symbol>
    </condition>
</applyRule>
```

The above conditions and rules represent examples showing how it becomes possible with SM-FL to transform classical exceptions into rules.

10.6.6 Priority Rule for symbols referred to notes

Markers and other symbols may refer to the same note. When this occurs, the order of these symbols is drawn depending on its significance. The Priority Rule defines the order of positioning and accordingly symbols must be drawn above and/or below the note. The positioning does not depend on the order symbols have been inserted. The Priority Rule is applied when the editor window is redrawn.

To edit the priority rule is something which can be used to change the order of symbols without modifying the symbolic description of music score. The priority rule has the form of a list of symbol identifiers shown in decreasing order of priority. For example,

```
<symbolOrderRule>
    <symbol>STAC</symbol>
    <symbol>TEN</symbol>
    <symbol>LEG</symbol>
    <symbol>MARTF</symbol>
    <symbol>ACCE</symbol>
    <symbol>SFOR</symbol>
    <symbol>MART</symbol>
    <symbol>ARCO</symbol>
    <symbol>PUNTA</symbol>
    <symbol>TALLONE</symbol>
    <symbol>PONTICELLO</symbol>
    <symbol>TASTIERA</symbol>
    <symbol>ARCATASU</symbol>
    <symbol>ARCATAGIU</symbol>
    <symbol>CORDA</symbol>
    <symbol>PIZZICATO</symbol>

    …
</symbolOrderRule>
```

The above priority rule was applied in the following figure. The staccato-dot had a higher priority than the other symbols, as it can be seen in the first six notes. The PUNTA (a big P) marker has a higher priority than the PONTICELLO (pont.) marker and evidence is found in the first note of the third measure. The slur (SLUR) has less priority than the staccato-dot (STACCATO) and the tenuto marker (TENUTO), but a higher priority than any other symbol as shown by the slur between the second and third measure of Figure 27.

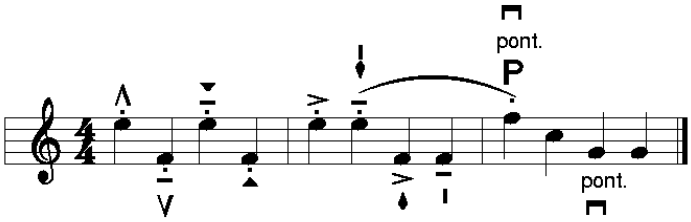


Figure 27 – The appearance order of the symbols is established in the priority rule

10.6.7 User-defined expression symbols

SM-FL allows the definition of new symbols, which are considered as generic expression symbols related to a note. For these new symbols, insertion and positioning rules can be defined. Since symbols rules are usually very similar, symbols are grouped and rules are defined for group of symbols. However specific rules can be defined as well.

A new symbol can be defined using the `symbolDef` command:

```
<groupDef name="faces" font="myexpr.fon">
    <symbolDef name="smile">
        <code>33</code>
    </symbolDef>
</groupDef>
```

specifying:
- the name of the symbol (“smile”),
- the group it belongs to (“faces”),
- the name of the font where such symbol can be found (“myexpr.fon”), if the font reference is omitted, then the name of the font is assumed to be the same as the name of the group,
- the code of the character representing the symbol (36).

The insertion and positioning rule for a group of symbols or for a specific symbol can be defined in the same way as for other symbols related to a note:

```
<symbolDirectionRule ruleId="faces0">
    <group>faces</group>
    <oppositeToStem/>
</symbolDirectionRule>
<symbolDirectionRule ruleId="faces1">
    <group>faces</group>
    <onStem/>
</symbolDirectionRule>

<applyRule rule="faces0">
    <condition>
        <note>
            <inSingleVoice/>
        </note>
    </condition>
</applyRule>
<applyRule rule="faces1">
    <condition>
        <note>
            <inMultivoice/>
        </note>
    </condition>
</applyRule>
```

```
<symbolPositionRule ruleId="faces2">
    <group>faces</group>
    <onSpace/>
    <outsideStaff/>
    <dx>0</dx>
    <dy>0</dy>
</symbolPositionRule>

<applyRule rule="faces2">
    <condition>
        <symbol>
            <oppositeToStem/>
        </symbol>
    </condition>
</applyRule>
<applyRule rule="faces2">
    <condition>
        <symbol>
            <onStem/>
        </symbol>
    </condition>
</applyRule>
…
```

The symbol name (and not the group name) has to be used in the priority rule to state the vertical relation of the symbol with other symbols:

```
<symbolOrderRule>
    <symbol>STAC</symbol>
    <symbol>TEN</symbol>
    <symbol>LEG</symbol>
    <symbol>MARTF</symbol>
    …
    <symbol>PIZZICATO</symbol>
    <symbol>smile</symbol>
    <symbol>star5</symbol>
    <symbol>sad</symbol>
    <symbol>STRING</symbol>
    …
</symbolOrderRule>
```

meaning that the “smile” symbol has to be over the pizzicato and below the “star5” symbol.

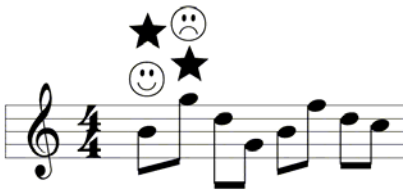


Figure 28 – Notes with user defined symbols

10.6.8 Note head rules

SM-FL positioning rules can be defined to manage the note head appearance. Two kinds of rules are related to the note head: (i) the rules used to find the proper font symbol to be used as note head, (ii) rules used to adjust the stem start position in relation with the note head type.

The note head can belong to one of the following types:

CLASSIC, DDIESIS, MOON, RHYTHMIC, TRIANG, TRIANG_UP, DIAMOND, PLUS_HEAD, CIRCLEX, X_HEAD, ALPHANUM, ALPHANUM_SQUARE, ALPHANUM_REVERSE.

However, the note head type does not completely define the note head appearance (the font character code) and some other conditions have to be considered. For example the classic note head depends on the note duration: for note less than 1/2 the black note head is used, while for notes with duration greater or equal to 1/2 the white note head is used.

What follows is a list of some rules used for CLASSIC notes:

```
<headRule ruleId="CodeEllipse">
   <code>216</code>
</headRule>
<headRule ruleId="CodeMinima">
   <code>163</code>
</headRule>
…

<applyRule rule="CodeEllipse">
   <condition>
      <note>
         <head>CLASSIC</head>
      </note>
   </condition>
</applyRule>
<applyRule rule="CodeMinima">
   <condition>
      <note>
         <head>CLASSIC</head>
         <duration>D1_2</duration>
      </note>
   </condition>
</applyRule>
…
```

Similar rules are used for rhythmic note heads:

```
<headRule ruleId="CodeRhythmicT">
   <code>170</code>
</headRule>
<headRule ruleId="CodeRhythmicB">
   <code>171</code>
</headRule>
<headRule ruleId="CodeRhythmicW">
   <code>172</code>
</headRule>
…

<applyRule rule="CodeRhythmicT">
   <condition>
      <note>
         <head>RHYTHMIC</head>
      </note>
   </condition>
</applyRule>
<applyRule rule="CodeRhythmicB">
   <condition>
      <note>
```

```
            <head>RHYTHMIC</head>
            <duration>D1_4</duration>
        </note>
    </condition>
</applyRule>
<applyRule rule="CodeRhythmicW">
    <condition>
        <note>
            <head>RHYTHMIC</head>
            <duration>D1_2</duration>
        </note>
    </condition>
</applyRule>
<applyRule rule="CodeRhythmicW">
    <condition>
        <note>
            <head>RHYTHMIC</head>
            <duration>D1</duration>
        </note>
    </condition>
</applyRule>
…
```

Figure 29 – Rhythmic note heads

Another positioning rule, which was defined with SM-FL, is the stem start position. It depends mainly on the note head type and the direction of the stem (up/down).

The start point of the stem is identified with respect to the note head center using DX and DY parameters that can be equal to –1, 0, 1 (left/bottom, center, right/up) subsequently 8 valid combinations are possible:

```
<stemStartRule ruleId="StemStartTopLeft">
    <alignx>LEFT</alignx>
    <aligny>TOP</aligny>
</stemStartRule>
<stemStartRule ruleId="StemStartMiddleLeft">
    <alignx>LEFT</alignx>
    <aligny>CENTER</aligny>
</stemStartRule>
<stemStartRule ruleId="StemStartBottomLeft">
    <alignx>LEFT</alignx>
    <aligny>BOTTOM</aligny>
    </stemStartRule>

<stemStartRule ruleId="StemStartTopCenter">
    <alignx>CENTER</alignx>
    <aligny>TOP</aligny>
    </stemStartRule>
<stemStartRule ruleId="StemStartBottomCenter">
    <alignx>CENTER</alignx>
    <aligny>BOTTOM</aligny>
    </stemStartRule>
```

```
<stemStartRule ruleId="StemStartTopRight">
    <alignx>RIGHT</alignx>
    <aligny>TOP</aligny>
</stemStartRule>
<stemStartRule ruleId="StemStartMiddleRight">
    <alignx>RIGHT</alignx>
    <aligny>CENTER</aligny>
</stemStartRule>
<stemStartRule ruleId="StemStartBottomRight">
    <alignx>RIGHT</alignx>
    <aligny>BOTTOM</aligny>
    </stemStartRule>

<applyRule rule="StemStartMiddleRight">
    <condition>
        <note>
            <head>CLASSIC</head>
            <stemUp/>
        </note>
    </condition>
</applyRule>
<applyRule rule="StemStartMiddleLeft">
    <condition>
        <note>
            <head>CLASSIC</head>
            <stemDown/>
        </note>
    </condition>
</applyRule>

<applyRule rule="StemStartTopRight">
    <condition>
        <note>
            <head>RHYTHMIC</head>
            <stemUp/>
        </note>
    </condition>
</applyRule>
<applyRule rule="StemStartBottomLeft">
    <condition>
        <note>
            <head>RHYTHMIC</head>
            <stemDown/>
        </note>
    </condition>
</applyRule>

<applyRule rule="StemStartTopCenter">
    <condition>
        <note>
            <head>ALPHANUM</head>
            <stemUp/>
        </note>
    </condition>
</applyRule>
<applyRule rule="StemStartBottomCenter">
    <condition>
        <note>
            <head>ALPHANUM</head>
            <stemDown/>
        </note>
```

```
    </condition>
</applyRule>
```

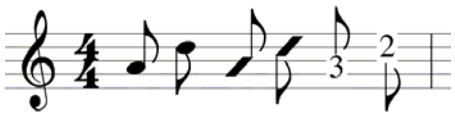


Figure 30 – Examples of stem start for classic, rhythmic and alphanum notehead

11 Relationship of SMR with other parts of the standard

11.1 Introduction

Content integrating SMR information is a rather complex piece of information that the SMR standard tries to structure and abstract in a simple way. Integration of SMR into MPEG-4 creates very rich content, but at the same time the nature itself of SMR, a music (and then related and synchronized with audio) format rendered by visual and graphic symbols or at the same time by structured audio events, requires several relationships with other existing tools in order to fully exploit its potential richness.

A fundamental relationship, allowing integration and synchronization with all other media types, is of course that with MPEG-4 Systems. Given the double audio-visual nature of SMR, specific functionality exist in MPEG-4 Systems to support it in all its aspects.

MIDI, being a protocol based on symbolic information, even if not notation information, it is also rather straightforward to enhance the richness and flexibility of the SMR toolset by a direct relationship with MPEG-4 Structured Audio (through which MIDI information can be carried over MPEG-4). The SMR decoder provides a direct support of SA streams containing MIDI objects.

Finally, given that the SMR format is an extensible one, allowing the definition and graphic specification of new symbols, other than the graphic representation of the traditional CWMN ones and others, a relationship with MPEG-4 fonts is also provided to simplify the approach to this feature and re-use at best already specified technology.

The following figure reports a simple example of an MPEG-4 player supporting MPEG-4 SMR. The player uses the SMR node in the BIFS scene to render the symbolic music information in the scene (for example, by exploiting functionality of other BIFS nodes) as the SMR decoder decodes it. The user can interact with the SMR (to change a page, view, transpose, and so on) through the SMR interface node, using sensors in association with other nodes defining the audiovisual, interactive content. The user sends commands from the SMR node fields to the SMR decoder (dashed lines in the figure), which generates a new view for displaying the scene. A user client tool automatically converts MIDI files (through a specific algorithm) into SMR on the client side and render them. Similarly, the server might only deliver the SMR. In these cases, the client can generate the MIDI information from SMR for use with MIDI-compliant devices. This is particularly important to guarantee straightforward adaptation of current devices.

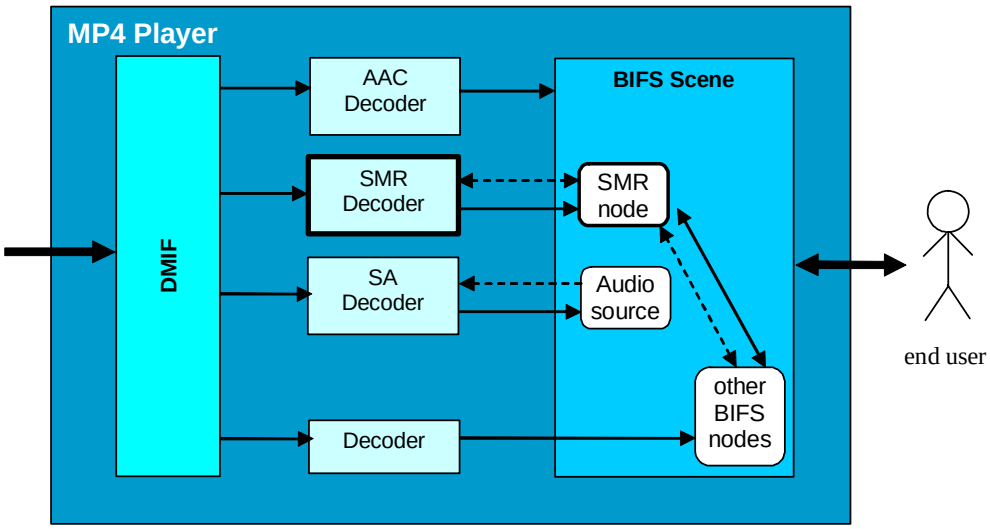


Figure 31 – MPEG-4 Player Architecture

11.2 SMR and MPEG-4 Systems

Interoperability between MPEG-4 SMR inside the MPEG-4 architecture is provided through the normative interface described in ISO/IEC 14496-11 (Scene Description).

Rendering of Symbolic Music allows different solutions ranging from bitmap to vector graphics. Two nodes are defined to provide support to SMR for different rendering solutions: the **MusicScore** node displays the score and specifies a flexible interface allowing interaction through a multimedia scene on several control parameters linked to the underlying SMR decoder; **ScoreShape** is used to map a **MusicScore** on a geometry, and has a **MusicScore** as a child node. In such a way different solutions are allowed, including vector graphics and bitmaps.

The **url** exposedField defines the SMR data stream to be attached to the MPEG-4 scene; the stream may be composed by different data chunks for parts, lyrics, score, and synchronization info as described in this normative text.

When an **executeCommand** eventIn is dispatched the command set in **commandOnExecute** has to be performed by the decoder, considering the values of field **argumentsOnExecute** and those of field **mousePosition**. Commands that shall be supported by the *commandOnExecute,* according to the profile, are:

- "ADD_TEXT_ANNOTATION"
  the first value in *argumentsOnExecute* contains the text to be added to the score in the position where the user will click.
- "ADD_LABEL"
  the first value in *argumentsOnExecute* contains the label text to be added to the measure where the user will click if the measure already has a label the label is substituted
- "ADD_NOTE"
  the first value in *argumentsOnExecute* contains the note duration: D1, D1_2, D1_4, D1_8, D1_16, D1_32, D1_64; the second value indicates the notehead type: "CLASSIC", … the note is inserted where the user clicks or it is added to a chord if sufficiently near to another note/chord.
- "ADD_REST"
  the first value in *argumentsOnExecute* contains the rest duration: D1, D1_2, D1_4, D1_8, D1_16, D1_32, D1_64; the rest is inserted where the user clicks.
- "SET_ALTERATION"
  the first value in *argumentsOnExecute* contains the alteration to be set on the note, it can be:

"SHARP","DSHARP","FLAT","DFLAT","NATURAL". The alteration is set to the note where the user clicks.

- "SET_DOTS"
  the first value in *argumentsOnExecute* contains the number of dots to be set on the note, it can be: "0","1","2". The dots are set to the note where the user clicks.
- "ADD_SYMBOL"
  the first value in *argumentsOnExecute* contains the symbol to be added on the note/rest/measure, it can be: "STACCATO","TENUTO" or any symbol defined using the formatting language. The symbol is added where the user clicks.
- "ADD_MEASURE"
  adds a measure to the score, the first value in *argumentsOnExecute* can be: "BEFORE", "AFTER" or "APPEND", the second value in *argumentsOnExecute* indicates the measure number with respect to the new measure is added. If the second value is not present or empty the reference measure is determined using the last *mousePosition* eventIn.
- "DEL_MEASURE"
  removes a measure of the score; the first value in *argumentsOnExecute* indicates the measure number to be removed. If the value is not present or empty the measure to be deleted is determined from the last *mousePosition* eventIn.
- "CHANGE_CLEF"
  changes the clef of a measure and for all the following until another clef change or to the end. The first value in *argumentsOnExecute* contains the clef type, it can be: "TREBLE", "SOPRANO", "BASS", "TENOR"…, The clef change applies to the measure where the user clicks.
- "CHANGE_KEYSIGNATURE"
  changes the key signature of a measure and for all the following until another key signature change or to the end. The first value in *argumentsOnExecute* contains the key signature type, it can be: "DOdM", "FAdM", "SIM", … The key signature change applies to the measure where the user clicks.
- "CHANGE_TIME"
  changes the time of a measure and for all the following until another time change or to the end. The first value in *argumentsOnExecute* contains the time, it can be: "4/4", "3/4", "2/4", "C" or "C/". The time change applies to the measure where the user clicks.
- "SET_METRONOME"
  sets the metronome for the whole piece. The first value in *argumentsOnExecute* contains the reference note duration (D1, D1_2, D1_4,…) the second value contains "TRUE" if the reference note is with augmentation dot ("FALSE" or empty otherwise), the third value indicates the number of reference notes in one minute. For example ["D1_4", "TRUE", "100"] sets a metronome with 100 dotted quarters in one minute. The metronome is set using the *executeCommand* eventIn.
- "DELETE"
  allows deleting any symbol, note, rest, alteration, label and annotation added by the user in the position where the user clicks.
- "TRANSPOSE"
  allows transposing the score. The first value in *argumentsOnExecute* contains the part to be transposed (0 for the whole main score, 1 for the first upper part, 2 the second part, …), the second value indicates the measure from which to start the transposition, the third value indicates the measure where to end transposition (the measure is included) a value of 0 or negative indicates to transpose until the last measure, the fourth value indicates the amount of transposition in half tones (e.g. 1 to increase of a half tone, 2 to increase of a tone, -1 to decrease of a half tone). This command does not depend on the mouse position and it is executed when the *executeCommand* eventIn is issued.

When a **gotoLabel** eventIn is dispatched the decoder shall build a view for the score on the page containing the specified label (one of the *availableLabels*).

When a **gotoMeasure** eventIn is dispatched the decoder shall build a view of the score on the page containing the specified measure.

When a **highlightTimePosition** eventIn arrives the decoder shall highlight the time position indicated relative to the scoreOffset field.

The **firstVisibleMeasure** exposedField is the first measure currently visible.

The **partsLyrics** exposedField is an array of strings indicating for which part to view the lyrics and in which language (e.g. ["it", "en", ""] to view lyrics for part 1 in Italian and for part 2 in English) it should be interpreded by the decoder to show the requested lyrics.

The **partsShown** exposedField is an array of integers indicating which parts have to be shown; the number is the position in the array of parts names; if **partShown** is empty all parts will be visible (e.g. [ ] to view main score with all parts, [2] to view single part number 2, [1,3] view main score with parts 1 and 3, etc.). It shall be interpreded by the decoder to build the correct view.

The **scoreOffset** exposedField indicates the initial (or point 0) offset from the beginning of the score; it may be used to change page or move inside the score before starting it, or in pause etc. **scoreOffset** is indicated in seconds from the beginning of the score. **scoreOffset** can be used only if synchronization information is provided or a metronome indication is present in the score. It is used by the decoder to generate the correct view.

The **size** exposedField parameter expresses the width and height of the music score specified in the units of the local coordinate system. A size of -1 in either coordinate means that the **MusicScore** node is not specified in size in that dimension, and that the size is adjusted to the size of the parent node. This value is used by the decoder to build an image of the appropriate size.

The **transpose** exposedField defines the transposition in units and cents, when transposition changes the decoder have to transpose the currently visible parts.

The **urlSA** exposedField defines a possibly associated SA (i.e. MIDI) data stream, it can be used from the decoder to build a view as symbolic music of the MIDI data.

The **viewType** exposedField indicates the kind of view to be used (one of the *availableViewTypes*), it used from the decoder to build the view.

The **availableCommands** eventOut is an array of commands that can be performed on the score by the user when the user clicks on the score itself (e.g. ["ADD_LABEL", "ADD_TEXT_ANNOTATION", "DELETE"]).

The **availableLabels** eventOut provides to the decoder an array of strings with labels (e.g. ["A", "B", "SEGNO", "CODA"]).

The **availableLyricLanguages** eventOut is an array of strings specifying for each part the list of languages (using the ISO 639-2 standard), separated by ";", for which the lyrics are available (e.g. ["en;it", "en;it", ""])

The **availableViewTypes** eventOut is an array of strings describing which view types are available for the score and for the decoder (e.g. ["CWMN", "braille", "neumes"]).

The **highlightPosition** eventOut outputs the highlight position in local coordinates.

The **lastVisibleMeasure** eventOut is the last measure currently visible.

The **numMeasures** eventOut is the number of measures in the score.

The **partNames** eventOut is an array of strings with part names (instruments, e.g. ["soprano", "baritone", "piano"]).

11.3 SMR and MIDI (through MPEG-4 Structured Audio)

The MPEG-4 standard ISO/IEC 14496-3 subpart 5 (Structured Audio) describes the normative decoding process for supporting MIDI commands and standard files (Object Type 1), and the normative mapping from MIDI events in the stream information header and bitstream data into SAOL semantics (Object Types 3 and 4).

The MIDI standards referenced are standardised externally by the MIDI Manufacturers Association. In particular, we reference the Standard MIDI File format, the MIDI protocol, and the General MIDI patch mapping, all standardised in [13]. The MIDI terminology used in this subclause is defined in that document.

Structured Audio allows then integrating standard MIDI events and files into an MPEG-4 stream. The **urlSA** exposedField of the **MusicScore** BIFS node defines a possibly associated audio Object Type 15 (SA object type 1, i.e., General MIDI) data stream. SMR decoders shall be capable to decode access units belonging to SA streams and containing data compatible with Object Type 15.

Audio Object Types 13 and 16 (also supporting MIDI, but with all the other tools of SA included) are not yet normatively supported inside SMR. Other chunks of information contained in the SA access units, like SASL chunks, can be ignored by the decoder.

11.4 SMR and MPEG fonts

The MPEG-4 standard ISO/IEC 14496-18 describes technology for Font compression and streaming. More in particular, MPEG-4 Part 18 defines: a font format (OpenType); a font compression technology (for TrueType and OpenType fonts); and the syntax and semantics of coded data streams.

Two access unit formats are specified, the basic access unit, needing a header configuration in the DecoderSpecificInfo, and the enhanced access unit, providing a self-contained format with all the necessary information about the font data. Only this second access unit format shall be used inside SMR streams, as in this way fonts can be directly integrated into the SMR data as specific chunk of information. The syntax for EnhancedFontAccessUnit is specified in ISO/IEC 14496-18, subclause 5.2.2.

Three different Profiles are defined for MPEG-4 fonts; access units supported by the SMR decoder shall be limited to the Simple Text Profile, which provides the possibility to use all existing TrueType fonts and OpenType fonts with TrueType outlines containing the set of required tables and embedded bitmaps. The Font Data Format and the normative way to decode access units at the Simple Text Profile is specified in ISO/IEC 14496-18, clauses 3 and 4.

In the SMR bitstream, chunks containing font data information shall be received before or at the same time as the first score data chunk (main score or part file) to be considered as valid. In particular if font data are contained in data chunks of an access unit x, main score and part files received in the SymbolicMusicSpecificInfo or in access units received before x shall be decoded using, when possible, available default fonts and the font in access unit x may be ignored by the decoder.

Rendering symbols making use of specific fonts in the BIFS scene will be done directly through the **MusicScore** and **ScoreShape** BIFS nodes and then the Text and FontStyle nodes are not needed in this case.

12 SMR Object Types for Profiles

There are two object types standardised for this subpart based on the values that the **codingType** field can have in the SMR bitstream (see subclause 7.3). Each of these object types corresponds to a particular set of application requirements.

No Levels are defined so far.

12.1 Simple Object Type

In the Simple Profile only values 0x01 and 0x10 are allowed for **codingType**. It is determined by objectType 36 in AudioSpecificConfig().

12.2 Main Object Type

In the Main Profile all values of **codingType** specified in subclause 7.3 are allowed. It is determined by objectType 37 in AudioSpecificConfig().

13 List of digital annexes

- SMR XML schema: (normative)
    - **smxf_lyric.xsd:** schema of the multilingual lyric
    - **smxf_main.xsd:** main score XML schema
    - **smxf_part.xsd**: parts XML schema
    - **smfl.xsd:** schema of the formatting language
    - **SMR_font_table.xls**: Table of symbols defined in font files

Bibliography

[1] P. Bellini, P. Nesi, and G. Zoia "Symbolic Music Representation in MPEG," IEEE Multimedia Magazine, vol. 12, no.4, October-December 2005, pp.12-20

[2] Music notation glossary from MUSICNETWORK: http://www.interactivemusicnetwork.org/glossary/index.htm

[3] WWW.WEDELMUSIC.ORG, the WEDELMUSIC Editor and tools.

[4] WEDELMUSIC Editor Manual: WWW.WEDELMUSIC.ORG

[5] SC 29 N 6689, *Call for Proposals for Symbolic Music Representation*

[6] P. Bellini, R. Della Santa, and P. Nesi, "Automatic Formatting of Music Sheet. Proc. of the First International Conference on WEB Delivering of Music," WEDELMUSIC-2001, IEEE Press, 23-24 November, Florence, Italy, pp.170-177, 2001.

[7] P. Bellini, I. Bruno, P. Nesi, “Automatic Formatting of Music Sheets through MILLA Rule-Based Language and Engine,” Journal of New Music Research, 2005.

[8] P. Bellini, J. Barthelemy, I. Bruno, P. Nesi, and M. B. Spinu, “Multimedia Music Sharing among Mediateques, Archives and Distribution to their Attendees,” Journal on Applied Artificial Intelligence, Taylor and Francis, vol.17, no.8-9, pp.773-795, 2003.

[9] P. Bellini, F. Fioravanti, and P. Nesi, “Managing Music in Orchestras,” IEEE Computer, September 1999, pp.26-34, http://www.dsi.unifi.it/~moods/.

[10] P. Bellini, P. Nesi, and M.B. Spinu, "Cooperative Visual Manipulation of Music Notation. ACM Transactions on Computer-Human Interaction," September, vol.9, no.3, pp.194-237, 2002.

[11] P. Bellini, and P. Nesi, “WEDELMUSIC FORMAT: An XML Music Notation Format for Emerging Applications,” Proceedings of the 1st International Conference of Web Delivering of Music, IEEE press, 23-24 November 2001, Florence, Italy, pp. 79-86.

[12] MUSICNETWORK MPEG SMR web page: http://www.interactivemusicnetwork.org/mpeg-ahg

[13] MIDI Manufacturers Association, The Complete MIDI 1.0 Detailed Specification v. 96.2.

Symbol	Description	font	ID	XM-XF name	XM-FL name
	Double flat	musica	167	DFLAT	DFLAT
	Small double flat	musica	96	DFLAT	DFLAT
	double sharp	musica	154	DSHARP	DSHARP
	double flat with brakets	musica2	138	DFLAT	DFLAT
	small double flat with brakets	musica2	195	DFLAT	DFLAT
	double sharp with brakets	musica2	137	DSHARP	DSHARP
	small double sharp with brakets	musica2	194	DSHARP	DSHARP
	a quarter flat	musica	178	FLAT1Q	FLAT1Q
	small flat of a quarter	musica	188	FLAT1Q	FLAT1Q
	3/4 flat	musica	180	FLAT3Q	FLAT3Q
	small 3/4 flat	musica	191	FLAT3Q	FLAT3Q
	flat	musica	98	FLAT	FLAT
	flat with brakets	musica2	134	FLAT	FLAT
	small flat	musica	105	FLAT	FLAT
	small flat with brakets	musica2	191	FLAT	FLAT
	quarter flat with brakets	musica2	133	FLAT1Q	FLAT1Q
	small quarter flat with brakets	musica2	190	FLAT1Q	FLAT1Q
	3/4 flat with brakets	musica2	135	FLAT3Q	FLAT3Q
	small 3/4 flat with brakets	musica2	192	FLAT3Q	FLAT3Q
	natural	musica	110	NATURAL	NATURAL
	natural with brakets	musica2	136	NATURAL	NATURAL
	small natural	musica	222	NATURAL	NATURAL
	small natural with brakets	musica2	193	NATURAL	NATURAL
	quarter natural	musica	218	NATURALUP	NATURALUP
	quarter natural with brakets	musica2	139	NATURALUP	NATURALUP
	small quarter natural	musica	220	NATURALUP	NATURALUP
	small quarter natural with brakets	musica2	196	NATURALUP	NATURALUP
	3/4 natural	musica	219	NATURALDOWN	NATURALDOWN
	3/4 natural with brakets	musica2	140	NATURALDOWN	NATURALDOWN
	small 3/4 natural	musica	221	NATURALDOWN	NATURALDOWN
	small 3/4 natural with brakets	musica2	197	NATURALDOWN	NATURALDOWN
	quarter sharp	musica	176	SHARP1Q	SHARP1Q
	small quarter sharp	musica	186	SHARP1Q	SHARP1Q
	3/4 sharp	musica	177	SHARP3Q	SHARP3Q
	small 3/4 sharp	musica	187	SHARP3Q	SHARP3Q

Symbol	Description	font	ID	XM-XF name	XM-FL name
	sharp	musica	35	SHARP	SHARP
(#)	sharp with brakets	musica2	131	SHARP	SHARP
	small sharp	musica	73	SHARP	SHARP
(#)	small sharp with brakets	musica2	188	SHARP	SHARP
	quarter sharp with brakets	musica2	130	SHARP1Q	SHARP1Q
	quarter sharp with brakets	musica2	187	SHARP1Q	SAHRP1Q
	3/4 sharp with brakets	musica2	132	SHARP3Q	SHARP3Q
	small 3/4 sharp with brakets	musica2	189	SHARP3Q	SHARP3Q
𝄪	small double sharp	musica	93	DSHARP	DSHARP
:	semicolon used in barlines	musica	145		
: :	Double semicolon used in inizio fine barlines	musica	185		
,	Take a break please	musica	44		
	Be careful to this notes	musica2	33		
	Basso clef	musica		BASS	CLEFCHANGE
	Small Basso clef	musica		BASS	CLEFCHANGE
	Baritone clef	musica	63	BARITONE	CLEFCHANGE
	Small Baritone clef	musica	193	BARITONE	CLEFCHANGE
	Bass clef one 0ctave above middle C	musica	140	8BASS	CLEFCHANGE
	Small Bass clef one 0ctave above middle C	musica	168	8BASS	CLEFCHANGE
	Bass clef one 0ctave below middle C	musica	203	BASS8	CLEFCHANGE
	Small Bass clef one 0ctave below middle C	musica	204	BASS8	CLEFCHANGE
	Old basso clef	musica	130	BASSOLD	CLEFCHANGE
	small old basso clef	musica	160	BASSOLD	CLEFCHANGE
	clef for percussions	musica	131	PERCUS2LINES	
	small clef for percussion	musica	161	PERCUS2LINES	
	clef for percussions	musica	135	PERCUSBOX	
	small clef for percussion	musica	34	PERCUSBOX	
	Clef for tablatures	musica	139	TAB	
	small clef for tablatures	musica	212	TAB	
	Tenor clef	musica	66	TENOR	CLEFCHANGE

Symbol	Description	font	ID	XM-XF name	XM-FL name
	Small Tenor clef	musica	192	TENOR	CLEFCHANGE
	Tenor clef one 0ctave below middle C	musica	81	TENOR8	CLEFCHANGE
	Small Tenor clef one 0ctave below middle C	musica	33	TENOR8	CLEFCHANGE
	Contralto clef	musica	66	ALTO	CLEFCHANGE
	Small Contralto clef	musica	192	ALTO	CLEFCHANGE
	Mezzo-soprano clef	musica	66	MEZZOSOPRANO	CLEFCHANGE
	Small mezzo-soprano clef	musica	192	MEZZOSOPRANO	CLEFCHANGE
	Soprano clef	musica	66	SOPRANO	CLEFCHANGE
	small soprano clef	musica	192	SOPRANO	CLEFCHANGE
	Treble cleff	musica	38	TREBLE	CLEFCHANGE
	small treble cleff	musica	194	TREBLE	CLEFCHANGE
	Treble clef one 0ctave above middle C	musica	69	8TREBLE	CLEFCHANGE
	Small Treble clef one octave above middle C	musica	159	8TREBLE	CLEFCHANGE
	Treble clef one octave below middle C	musica	86	TREBLE8	CLEFCHANGE
	Small Treble clef one octave below middle C	musica	195	TREBLE8	CLEFCHANGE
f	(Forte) Loud	musica	102	F	
ff	(Fortissimo) Very Loud	musica	224	FF	
fff	Fortississimo	musica	225	FFF	
ffff	Fortissississimo	musica	137	FFFF	
fffff	Fortisssississimo	musica	226	FFFFF	
ffffff	Fortissississississimo	musica	227	FFFFFF	
ffp	Fortissimopiano	musica	240	FFP	
ffpp	Fortissimopianissimo	musica	241	FFPP	
fp	Fortepiano	musica	136	FP	
fpp	Fortepianissimo	musica2	113	FPP	
fz	(forzando) forced, sudden accent	musica	125	FZ	
mf	(mezzo forte) Moderately loud	musica	70	MF	
mp	(mezzo piano) Moderately soft	musica	80	MP	
p	(piano) Soft	musica	112	P	
pp	(pianissimo) Very soft	musica	213	PP	

Symbol	Description	font	ID	XM-XF name	XM-FL name
ppp	pianississimo	musica	122	PPP	
pppp	pianissississimo	musica	123	PPPP	
ppppp	pianississississimo	musica	228	PPPPP	
pppppp	pianissississississimo	musica	229	PPPPPP	
rf	rinforzato	musica	155	RF	
rfz	(rinsforzando) Reinforced	musica2	114	RFZ	
sf	sforzato	musica	116	SF	
sff	sfozatissimo	musica2	115	SFF	
sfff	sforzatississimo	musica2	116	SFFF	
sffp	sforzatissimopiano	musica2	117	SFFP	
sffz	sforzatissimo	musica2	118	SFFZ	
sfp	sforzatopiano	musica2	119	SFP	
sfpp	sforzatopianissmo	musica2	120	SFPP	
sfz	(sforzando) Strongly accented	musica	126	SFZ	
sp	……	musica	115	SP	
>	accent	musica	62	ACCENTO	ACCENTO
𝅘𝅥	martellato dolce	musica	61	MARTDOLCE	MARTDOLCE
𝅘𝅥	martellato dolce	musica	60	MARTDOLCE	MARTDOLCE
𝅘𝅥	matellato	musica	210	MARTELLATO	MARTELLATO
𝅘𝅥	matellato	musica	169	MARTELLATO	MATELLATO
𝅘𝅥	punto allungato	musica	79	PUNTOALLUNGATO	PUNTOALLUNGATO
V	sforzato	musica	118	SFORZATO	SFORZATO
Λ	sforzato	musica	94	SFORZATO	SFORZATO
.	staccato dot	musica	107	STACCATO	STACCATO
-	tenuto	musica	95	TENUTO	TENUTO
𝄐	fermata medium	musica	173		FERMATA
𝄐	fermata short	musica	175		FERMATA
𝄐	fermata medium	musica2	123		FERMATA
𝄐	fermata short	musica2	124		FERMATA
𝄐	fermata long	musica	117		FERMATA
𝄐	fermata long	musica	85		FERMATA
0	finger empty	musica2	48		FINGER
⓪	finger empty	musica2	65		FINGER
⓪	finger empty	musica2	71		FINGER
1	finger 1	musica2	49		FINGER

Symbol	Description	font	ID	XM-XF name	XM-FL name
①	finger 1	musica2	66		FINGER
⓵	finger 1	musica2	72		FINGER
2	finger 2	musica2	50		FINGER
②	finger 2	musica2	67		FINGER
⓶	finger 2	musica2	73		FINGER
3	finger 3	musica2	51		FINGER
③	finger 3	musica2	68		FINGER
⓷	finger 3	musica2	74		FINGER
4	finger 4	musica2	52		FINGER
④	finger 4	musica2	69		FINGER
⓸	finger 4	musica2	75		FINGER
5	finger 5	musica2	53		FINGER
⑤	finger 5	musica2	70		FINGER
⓹	finger 5	musica2	76		FINGER
6	number 6	musica2	54		FINGER
7	number 7	musica2	55		FINGER
○	Harmonic for arcs	musica	200	STRINGS	HARMONIC
◇	Harmonics for flutes	musica	201	FLUTES	HARMONIC
15ba	quindicesima, 15th	musica2	156	QUINDBA	
15ma	quindicesima, 15th	musica2	155	QUINDMA	
15ma	15th lines	musica	19		
8ba	octave, 8th	musica2	154	OCTBA	OCTAVE
8va	octave, 8th	musica	199	OCTVA	OCTAVE
8ba	octave, 8th, lines	musica	20		
⌐¬	Bend for guitars	musica	23	BEND	
⌐- - -¬	Bend dashed for guitars	musica	23	BENDDASH	
⌐····¬	Bend dotted for guitars	musica	23	BENDDOT	
◁	crescendo	musica	4	CRESCENDO	
▷	decrescendo	musica	4	DIMINUENDO	
◁	crescendo dotted	musica	4	DIMDASH	
▷	decrescendo dotted	musica	4	DIMDOT	
◁	crescendo dashed	musica	4	CREDOT	
▷	decrescendo dashed	musica	4	CREDASH	
𝔹	Labels for measures	musica	15		
(♩= 5)	Metronomic indication	musica	12		

Symbol	Description	font	ID	XM-XF name	XM-FL name
allegro (♩= 5)	Movement indication	musica	13		
|||	To define how the measure is beat by the conductor	musica	11		
+	with the mute	musica	43	PLUS	WITHMUTE
E	remove the mute	musica	134	OFF	WITHOUTMUTE
E	with the mute	musica	133	ON	WITHMUTE
-	remove the mute	musica	45	MINUS	WITHOUTMUTE
Sord.	with the mute	musica	27	IN	WITHMUTE
Via sord.	remove the mute	musica	26	VIA	WITHOUTMUTE
	Hook of notes	musica	108		
	Hook of notes	musica	71		
	Hook of notes	musica	74		
	Hook of notes	musica	106		
	Hook of notes	musica	208		
	Hook of small notes	musica	59		
	Hook of notes	musica	150		
	Hook of small notes	musica	58		
⊗	Notehead	musica	206	CIRCLEX	CIRCLEX
+	Notehead	musica	205		PLUS_HEAD
X	Notehead	musica	207		X-HEAD
	Notehead	musica	165	MOON	MOON
	Notehead	musica	166	MOON	MOON
	Notehead	musica	172	RHYTHMIC	RHYTHMIC
	Notehead	musica	171	RHYTHMIC	RHYTHMIC
	Notehead	musica	170		RHYTHMIC
	Notehead	musica	181		TRIANG
▲	Notehead	musica	183		TRIANG_UP
△	Notehead	musica	184		TRIANG_UP
	Notehead	musica	182		TRIANG
◆	Notehead	musica	189		DIAMOND
◇	Notehead	musica	190	DIAMOND	DIAMOND
	Notehead	musica	216	CLASSIC	CLASSIC
	Notehead	musica	163	notehead for the 1/note	CLASSIC
	Notehead	musica	89		CLASSIC
	1/2 note	musica	92		CLASSIC
	Whole Note	musica	119		NOTE1

Symbol	Description	font	ID	XM-XF name	XM-FL name
	Small whole note	musica	91		NOTE1
	Breve notes	musica	164		NOTE2
	Small breve note	musica	174		NOTE2
	Arpeggio	musica	214		ARPEGGIO
	Arpeggio Up	musica	231		ARPEGGIO
	Arpeggio down	musica	223		ARPEGGIO
	Glissando	musica	25	GLISSLINE	GLISSANDO
	Glissando with wave	musica	24	GLISSWAVE	GLISSANDO
	Mordent	musica	77	MORDENT	MORDENT
	Superior Mordent	musica	109	MORDENTSUP	MORDENT
	dobule superior mordent	musica	196	DMORDENTSUP	MORDENT
	double mordent	musica	202	DMORDENT	MORDENT
tr	Trill without wave	musica	235	TRILL	TRILL
	Tremolo	musica	16	TREMOLO	TREMOLO
	Trill with wave	musica	22	(H) TRILL	TRILL
	the wave of the Tril with wave	musica	21	(H) TRILL	TRILL
	Turn up	musica2	121	TURNUP	TURN
	Turn slash	musica2	122	TURNSLASH	TURN
	Turn back	musica	82	TURNBACK	TURN
	Turn	musica	84	TURN	TURN
	wave	musica	114		
•	Dot for Piano pedal closure	musica	239		
	Pedal down of the piano	musica	234		
	Pedal Up for the piano	musica	42		
	Pedal with Pallino for the piano, over the staff	musica	18	PIANOPEDAL	
	Pedal with Pallino for the piano, over the staff	musica	17	PIANOPEDAL	
)	Braket	musica	41		
(	Braket	musica	40		
)	Braket	musica	233		
(	Braket	musica	232		
	Coda Sign for reference on measures	musica	36	CODA	
	refrain reference	musica	9		
	refrain reference	musica	10		
	Sign used for marking measures	musica	37	SEGNO	

Symbol	Description	font	ID	XM-XF name	XM-FL name
D.C.	Da Capo	none	14	DC	
D.C. al Fine	Da Capo al Fine	none	14	DCAS	
D.S.	Dal Segno	none	14	DS	
D.S. al Fine	Dal Segno al Fine	none	14		
D.S. al Coda	Dal Segno al Coda	none	14		
Fine	Fine	none	14		
	Repeat of 2 Measures	musica	179		
	Repeat a measure, a number can be associated with the symbol	musica	152		
	Repeat of 1/2 of the measure	musica	153		
	repeat of a tempo	musica	151		
	Repeated note	musica	113	STEMTREMOLO	
empty	Reserved	musica	1		
empty	Reserved	musica	2		
empty	Reserved	musica	3		
empty	Reserved	musica	30		
empty	Reserved	musica	39		
empty	Reserved	musica	156		
empty	Reserved	musica	197		
empty	Reserved	musica	236		
	Rest of 4/4	musica	7		
	small rest of 4/4	musica	5		
	rest of 1/128	musica	75		
	small rest of 1/128	musica2	206		
	test of 1/16	musica	101		REST1_16
	small rest of 1/16	musica2	203		REST1_16
	rest of 1/32	musica	90		REST1_32
	small rest of 1/32	musica2	204		REST1_32
	rest of 1/4	musica	215		REST1_4
	small rest of 1/4	musica2	201		REST1_4
	rest of 1/64	musica	147		REST1_64
	small rest of 1/64	musica2	205		REST1_64
	rest of 1/8	musica	132		REST1_8
	small rest of 1/8	musica2	202		REST1_8
	rest of 2/4	musica	8		

Symbol	Description	font	ID	XM-XF name	XM-FL name
	small rest of 2/4	musica	6		
	rest of 2	musica	148		
	small rest of 2	musica2	199		
	rest of 4	musica	149		
	small rest of 4	musica2	200		
	generic rest, a number is typically assocaited with the bar	musica	146		
	Rectangle supporting the rests of 2/4 or 4/4	musica	138		
	Rectangle supporting the rests of 2/4 or 4/4	musica2	207		
	Vertical small space	generic1	33		vspace1
	Vertical large space	generic1	34		vspace2
(1)	strings	musica2	166		STRING
(2)	strings	musica2	167		STRING
(3)	strings	musica2	168		STRING
(4)	strings	musica2	169		STRING
(5)	strings	musica2	170		STRING
(6)	strings	musica2	171		STRING
(7)	strings	musica2	172		STRING
(I)	strings	musica2	86		STRING
(II)	strings	musica2	87		STRING
(III)	strings	musica2	88		STRING
(IV)	strings	musica2	89		STRING
(V)	strings	musica2	90		STRING
(VI)	strings	musica2	91		STRING
(VII)	strings	musica2	92		STRING
1	strings	musica2	159		STRING
①	strings	musica2	173		STRING
①	strings	musica2	180		STRING
2	strings	musica2	160		STRING
②	strings	musica2	174		STRING
②	strings	musica2	181		STRING
3	strings	musica2	161		STRING
③	strings	musica2	175		STRING
③	strings	musica2	182		STRING
4	strings	musica2	162		STRING

Symbol	Description	font	ID	XM-XF name	XM-FL name
④	strings	musica2	176		STRING
④	strings	musica2	183		STRING
5	strings	musica2	163		STRING
⑤	strings	musica2	177		STRING
⑤	strings	musica2	184		STRING
6	strings	musica2	164		STRING
⑥	strings	musica2	178		STRING
⑥	strings	musica2	185		STRING
7	strings	musica2	165		STRING
⑦	strings	musica2	179		STRING
⑦	strings	musica2	186		STRING
I	strings	musica2	79		STRING
Ⓘ	strings	musica2	93		STRING
Ⓘ	strings	musica2	100		STRING
II	strings	musica2	80		STRING
Ⓘ	strings	musica2	94		STRING
Ⓘ	strings	musica2	101		STRING
III	strings	musica2	81		STRING
Ⓘ	strings	musica2	95		STRING
Ⓘ	strings	musica2	102		STRING
IV	strings	musica2	82		STRING
Ⓘ	strings	musica2	96		STRING
Ⓘ	strings	musica2	103		STRING
V	strings	musica2	83		STRING
Ⓥ	strings	musica2	97		STRING
Ⓥ	strings	musica2	104		STRING
VI	strings	musica2	84		STRING
Ⓘ	strings	musica2	98		STRING
Ⓘ	strings	musica2	105		STRING
VII	strings	musica2	85		STRING
Ⓘ	strings	musica2	99		STRING
Ⓘ	strings	musica2	106		STRING
⊓	Bow down for strings	musica	88		BOWDOWN
V	Bow up for strings	musica	104		BOWUP
Pizz	Pizziccato	musica	31		PIZZICATO

Symbol	Description	font	ID	XM-XF name	XM-FL name
Pont	Ponticello	musica	28		PONTICELLO
P	Punta	musica	121		PUNTA
T	Tallone	musica	120		TALLONE
Tast	Tastiera	musica	29		TASTIERA
	Korean Sliding tone to	musica3	33	SLIDINGUP	KORN
	Korean Sliding tone to lower	musica3	34	SLIDINGDOWN	KORN
	Korean Two times Sliding tone to higher	musica3	35	SLIDINGDUP	KORN
	Korean Sliding tone to lower and to higher	musica3	36	SLIDINGDOWNUP	KORN
	Korean Nonghyun (vibrato)	musica3	37	NONGHYNINC	KORN
	Korean Nonghyun (vibrato)	musica3	38	NONGHYNDEC	KORN
	Korean Nonghyun (vibrato)	musica3	39	NONGHYNNARROW	KORN
	Korean Nonghyun (vibrato)	musica3	40	NONGHYNWIDE	KORN
0	Alfanumeric 0	musica	48		
1	Alfanumeric 1	musica	49		
2	Alfanumeric 2	musica	50		
3	Alfanumeric 3	musica	51		
4	Alfanumeric 4	musica	52		
5	Alfanumeric 5	musica	53		
6	Alfanumeric 6	musica	54		
7	Alfanumeric 7	musica	55		
8	Alfanumeric 8	musica	56		
9	Alfanumeric 9	musica	57		