INTERNATIONAL ORGANISATION FOR STANDARDISATION
ORGANISATION INTERNATIONALE DE NORMALISATION
ISO/IEC/JTC1/SC29/WG11
CODING OF MOVING PICTURES AND AUDIO INFORMATION

ISO/IEC JTC1/SC29/WG11/N 7385

Poznan, July 2005

Source: **Audio subgroup**

Status: **Approved**

Title: Text of ISO/IEC 15938-4:2002/FPDAM 2

Editor: **Matthias Gruhne, Gregoire Carpentier**

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

ISO/IEC JTC 1/SC 29

Date: 2005-08-17

ISO/IEC 15938-4:2002/FPDAM 2

ISO/IEC JTC 1/SC 29/WG 11

Secretariat:

Information technology — Multimedia content description interface — Part 4: Audio, AMENDMENT 2: High-level descriptors

Élément introductif — Élément central — Partie 4: Titre de la partie

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.

Amendment 2 to ISO/IEC 15938-4:2002 was prepared by Joint Technical Committee ISO/IEC JTC 1, *Information Technology*, Subcommittee SC 29, *Coding of Audio, Picture, Multimedia and Hypermedia Information*.

Information technology — Multimedia content description interface — Part 4: Audio, AMENDMENT 2: High-level descriptors

Add at the end of subclause 6.8.3.3.3:

6.9 Rhythmic Pattern

6.9.1 RhythmicBaseType

This descriptor contains description of one single rhythmic pattern.

6.9.1.1 Syntax

```
<!-- ################################################################ -->
<!-- Definition of RhythmicBaseD  -->
<!-- ################################################################ -->
<complexType name="RhythmicBaseType">
  <element name="PrimeIndex"   type="mpeg7:integerVector" />
  <element name="Velocity"     type="mpeg7:integerVector" />
  <element name="NumOfSamples" type="mpeg7:positiveInteger" minValue="1" />
</complexType>
```

6.9.1.2 Semantics

Name	Definition
`RhythmicBaseType`	This Type contains the basic rhythmic pattern
`PrimeIndex`	The Integer vector indicating the initial index
`Velocity`	The Integer vector indicating the velocity of the elements.
`NumOfSamples`	Positive Integer which indicates the number of elements within *Velocity* and *PrimeIndex* elements.

6.9.1.3 Usage (informative)

The RhythmicBaseType contains elements of a single rhythmic pattern with different degrees of resolution (i.e. on any different hierarchic levels). A representation of a rhythmic pattern requires indexing of the rhythmic grid with respect to the rhythmic significance of the grid position. This may be done in the following manner:

The calculation of each *PrimeIndex* of the example pattern may be done in the following manner:

1. Vector of prime factorization of the top part of the meter:
   $nomVec = \{ nom_1 \ldots nom_k \}$ sorted by size with the largest value first

2. Vector of prime factorization of the number of divisions per beat (tick):
   $mtVec = \{ mt_1 \ldots mt_k \}$

3. calculation of the prime indices from the grid positions:

```
patternLength = product(nomVec) *product(mtVec)
vector primeVec() ( size: pattenLength )
primaVec()= 0      ( set all Elements of the vector to 0 )
primeIndex = 1
primeProduct = 1
for i=1 : length(nomVec)
{
  primeProduct *= nomVec(i)
  while (count < patternLength)
 {
  if ((count/(patternLength/tempPrimeVec)== integer)
  {
      if (primeVec[count]==0)
      {
          primeVec[count] = primeIndex;
          primeIndex++;
      }
  }
  count++;
 }
}
for i=1 : length(mtVec)
{
 primeProduct *= mtVec (i)
 while (count < patternLength)
 {
  if ((count/(patternLength/tempPrimeVec)== integer)
  {
      if (primeVec[count]==0)
      {
          primeVec[count] = primeIndex;
          primeIndex++;
      }
  }
  count++;
}
```

The successive prime factorization of the nominator and the micro time is necessary, because a joint prime factorization of the maximum number of elements can lead to comparisons of patterns with different time signature (but same length) in cases of reduced rhythmic resolution.

Example 1: meter = 4/4; micro time = 2; resulting size = 4 * 2 = 8 ;

The following table shows a rhythmic pattern with a binary feeling notated as commonly done in a score-like representation: (red refers to the highest rhythmic hierarchy; blue refers to the lowest rhythmic hierarchy)

Part of the bar	1	1+	2	2+	3	3+	4	4+
Grid position	1	2	3	4	5	6	7	8
prime index	1	5	3	6	2	7	4	8
velocity	100	0	112	0	150	68	120	0

No elements will be applied for any part of the bar with velocity equal to zero:

Prime index	1	3	2	7	4
velocity	100	112	150	68	120

According to the ascending order of the prime indexes the elements will be rearranged, resulting in the final representation:

Prime index	1	2	3	4	7
velocity	100	150	112	120	68

Example 2: meter = 4/4; micro time = 3; resulting size = 4 * 3 = 12 ;

The following table shows a rhythmic pattern with a ternary feeling notated as commonly done in a score-like representation:

(red refers to the highest rhythmic hierarchy; blue refers to the lowest rhythmic hierarchy)

Part of the bar	1-1	1-2	1-3	2-1	2-2	2-3	3-1	3-2	3-3	4-1	4-2	4-3

Equivalent element index	1	2	3	4	5	6	7	8	9	10	11	12
velocity	180	0	100	200	0	99	190	0	97	205	0	101

No elements will be applied for any part of the bar with velocity equal to zero.

Index	1	3	4	6	7	9	10	12
velocity	180	100	200	99	190	97	205	101

According to the ascending order of the prime indexes the elements will be rearranged, resulting in the final representation:

Index	1	7	4	10	3	9	6	12
velocity	180	190	200	205	100	97	99	101

The following example demonstrates how two representations exhibiting different grades of resolution can be easily compared to each other. Only the number of elements of the shorter representation is taken into account. It is advantageous to compare only patterns with similar meter.

Pattern 1: meter: 4/4;

Prime index	1	2	3	4	7
velocity	100	150	112	120	68

Pattern2: meter: 4/4;

Prime index	1	2
velocity	120	145

The examples demonstrate how the use of a specific order allows the specification of a rhythmic hierarchy without any additional information.

6.9.2 AudioRhythmicPatternDS

The AudioRhythmicPatternType allows a compact representation of rhythmic patterns that are limited to the length of one musical measure. The internal structure of the representation is dependent on the underlying rhythmical structure of the pattern that is to be represented. These facts result in two main advantages: For every pattern the most compact representation can be provided, resulting in an efficient comparison of the patterns and minimal memory needed for the storage.

Regarding any further classification or matching of rhythmic patterns, the representation allows for a setting of several grades in the resolution, known as rhythmical levels.

6.9.2.1 Syntax of AudioRhythmicPatternDS

```
<!-- ################################################################ -->
<!-- Definition of AudioRhythmicPatternDS                             -->
<!-- ################################################################ -->
<complexType name="AudioRhythmicPatternDS">
  <extension base="mpeg7:AudioDSType">
    <sequence>
      <element name="Instrument"  type="mpeg7:CreationToolType />
      <element name="Recurrences"   type="mpeg7:positiveInteger"  />
      <element name="RhythmPattern" type="mpeg7:RhythmicBaseType" />
      <element name="Meter"         type="mpeg7:MeterType"         />
      <element name="AudioSegment"  type="mpeg7:AudioSegmentType "/>
    </sequence>
  </extension>
</complexType>
```

6.9.2.2 Semantics

Name	Definition
AudioRhythmicPatternDS	A description scheme providing a compact and efficient representation of rhythmical patterns.
Instrument	Describes the devices/procedure and settings used for the creation of the metadata, such as the tools used to extract the metadata or the extraction parameters. Instrument is of type CreationToolType. Musical instruments should be only drum instruments.
Recurrences	The number of recurrences of the same pattern.
RhythmPattern	The rhythmical pattern of an audio signal of *PatternType* data.
AudioSegment	The time when the pattern starts and the duration of it.

6.9.2.3 Instantiation requirements

In order to guarantee a proper instantiation of this description scheme, the following requirements have to be fulfilled:

- The element *Meter*, as provided by the *AudioPatternType*, must be instantiated.
- The number of elements of *Velocity* and *PrimeIndex*, as provided by *RhythmicBaseType* must be equal and not 0.

6.9.2.4 Usage (Informative)

The *AudioRhythmicPattern* DS uses a sequence to distinguish different rhythmic patterns. Single patterns have been specified in *RhythmicBaseType*. To define a drum instrument that plays the actual pattern the attribute *Instrument* has been introduced. When using the *CreationToolType* for specifying an instrument it must be assured, that only drum instruments have been used. *Recurrences* describes the number of times the same pattern is played consecutively. To describe the currently played pattern *RhythmPattern* must be used. The start of the rhythmic pattern and its length (including the number of recurrences) must be indicated by *AudioSegment*.

The first step of extracting rhythmic patterns out of a polyphonic music signal would be to transcribe percussive instruments. There have been many publications in technical literature that describe how to detect and classify percussive instruments. One of them is template matching by performing a differentiation, a halfway-rectification and a Principal Component Analysis on the input data to find spectral characteristics of un-pitched percussive instruments and to transcribe the actual drum tracks.

After transcribing the input data and obtaining the drum tracks, the actual pattern could be calculated. At first, the audio signal might be segmented into similar and characteristic regions using a self-similarity method. The segmentation is motivated by the assumption, that within each region not more than one representative drum pattern occurs, and that the rhythmic features are nearly invariant. Subsequently, the temporal positions of the events are quantized on a tatum grid. The term tatum grid refers to the pulse series on the lowest metric level. Tatum period and phase is computed by means of a two-way mismatch error procedure. The pattern length might be estimated by searching for the prominent periodicity in the quantized score with periods equal to an integer multiple of the bar length. The periodicity function is obtained by calculating a similarity measure between the signal and its time shifted version. The similarity between two score representations is calculated as weighted sum of the number of simultaneously occurring notes and rests in the score. The pattern is calculated by means of a histogram representation measuring the occurrence of notes on each metrical position within the pattern for each instrument. By comparing the histogram values with an arbitrary threshold the pattern elements are chosen as frequently occurring notes.

6.9.2.5 Applications

6.9.2.5.1 Automatic Retrieval and Recommendation

When using rhythmic patterns to refer to a particular musical style or genre it is possible to query for musical content that is also characterized by one ore more representative rhythmic patterns. This mechanism can serve as a search criterion in applications proposing a number of musical titles belonging to a particular musical style or genre or sounding similar to a title being proposed by the user as a “reference sound”.

6.9.2.5.2 New ways composing and arranging music

The possibility to search for musical excerpts with a particular rhythm or containing particular rhythmic elements may have a positive influence on the process of composing and arranging new music. Musicians, Producers, Composers, or Disc Jockeys that have a specific idea concerning the rhythm could “pre-listen” how this was realized in different existing arrangements by other composers.

6.10 Chord Pattern

6.10.1 ChordBaseType

This descriptor contains a description of a single musical chord. A chord is a musical structure formed when at least two tones having different notes are played simultaneously. If the default MPEG 7 scale descriptor is used, numbers 0 to 11 indicate the half tones in the scale. The representation of the *ChordBaseType* element consists of a *RelativeChordNumber*, which represents the base note of a chord relative to the active key, the type of triad, and additional tones. For example, a *RelativeChordNumber* of 4, in the Key of C, represents a base note of E because is four diatonic tones higher than the base key.

6.10.1.1 Syntax

```
<!-- ################################################################ -->
<!-- Definition of ChordBaseD  -->
<!-- ################################################################ -->
<complexType name="ChordBaseType">
    <complexContent>
      <extension base="mpeg7:AudioDSType">
          <attribute name="RelativeChordNumber" type="mpeg7:PositiveInteger"
                    use="required"/>
          <attribute name="Triad" use="optional" default="major">
             <simpleType>
                <restriction base="string">
                   <enumeration value="major"/>
                   <enumeration value="minor"/>
                   <enumeration value="augmented"/>
                   <enumeration value="diminished"/>
                   <enumeration value="sus2"/>
                   <enumeration value="sus4"/>
                </restriction>
             </simpleType>
          </attribute>
          <attribute name="SeventhChord" use="optional" default="major7">
             <simpleType>
                <restriction base="string">
                   <enumeration value="major7"/>
                   <enumeration value="dominant7"/>
                </restriction>
             </simpleType>
          </attribute>
          <sequence>
             <element name="NoteAdded" type="mpeg7:PositiveInteger"
                           use="optional"/>
             <element name="NoteRemoved" type="mpeg7:PositiveInteger "
             use="optional" maxOccurs=12 />
          </sequence>
          <attribute name="AlternateBass" type="mpeg7:PositiveInteger"
          use="optional" maxOccurs="1" use="optional"/>
       </extension>
    </complexContent>
</complexType>
```

6.10.1.2 Semantics

Name	Definition
RelativeChordNumber	This element is an integer that describes the base note of a currently played chord. The value of *RelativeChordNumber* indicates the numbers of diatonic tones in the scale from the root note of the active Key and it must not be greater than the number of elements of the scale that is used.
Triad	This string contains an enumeration of all possible types of the triad.
SeventhChord	This enumeration extends the already existing chord to a septimal harmony.
NoteAdded	*NoteAdded* is an integer that describes a note that additionally appears within the chord. Its value is the number of diatonic tones above the base note from the currently played chord. *Note that this is relative to RelativeChordNumber*. For Example a “C major” chord is extended by a “D”, *NoteAdded* would be 2. Its value must not be greater than twice the number of items in the scale.
NoteRemoved	In the case the base chords contains notes that are not desired, they may be removed with *NoteRemoved*. The value of NoteRemoved is the number of diatonic tones above the base note of the currently played chord. *Note that this is relative to RelativeChordNumber*. For Example to remove the “E” from a “C major” chord *NoteRemoved* would be 4. Its value must not be greater than the number of items in the scale.
AlternateBass	Some chords might have an alternate bass note This attribute indicates the base note to be played with the chord. Its value is the number of diatonic tones higher than the base note of the currently played chord. For Example to add an alternate “G” to a “C major” chord *AlternateBass* would be 7. Its value must not be greater than the number of items in the scale.

6.10.1.3 Usage and Extraction (Informative)

The descriptor ChordBaseType contains the description of one single chord. Its representation consists of a base note, the type of the triad and possible additional notes.

6.10.1.3.1 Representation

Musically, a tone interval can be described as a simultaneous appearance of at least two individual notes with different tones. At least three different tones would build a chord. In musical practice it is very common to describe a chord as a triad writing the base note and type of triad. Additionally one can add special information for notes, which are different from the triad, such as 7^{th} chords. In case of the *Chord Pattern*, depending on the scale, by default only the common western music style with enharmonic change is taken into consideration. The examples of the following text refer to the standard music scale as described in IEC 15938-4:2002, *ScaleType* with enharmonic change and a range 12 half tones. Furthermore the *RelativeChordNumber* is one and refers to note “C”. The following example image (Figure 1) shows the scale and referring key numbers on a usual piano keyboard.

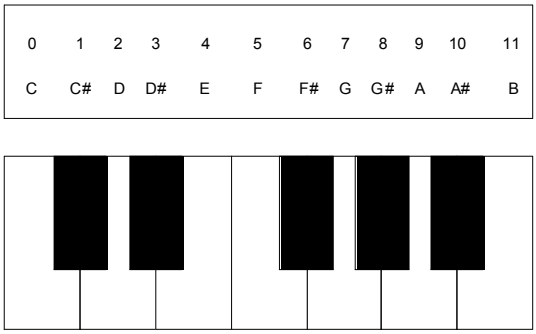

Figure 1 — Example piano keyboard and above example key and referring notes

To describe a musical chord using the *ChordBase* descriptor, one base note and five different extensions have been designed to cover all note possibilities. *RelativeChordNumber* contains the number referring to a base note of the triad. It represents the base note of a chord relative to the active key, the type of triad, and additional tones. For example, a RelativeChordNumber of 4, in the Key of C, represents a base note of E. In this example *RelativeChordNumber* is 0 and refers to note "C". The attribute *Triad* declares the mode of the triad. The default mode is a major chord. Appendix D shows a list containing 13 rows. The first row refers to the mode. The next 12 rows state the occurrence of notes in this chord by a "1" in the row, "0" otherwise. For example: The C major chord contains the notes "C", "E", and "G". Table 2 in Appendix D contains at row "major" a number only in columns 0, 4 and 7, meaning the chord consists of the base note (0→ "C"), the major third (4 diatonic tones above the base note →"E") and the fifth (7 diatonic tones above the base note). The same procedure would be fulfilled when creating the other triads. *SeventhChord* describes an appearance of a septimal harmony, where a tone 10 or 11 half tones higher than the base tone. It can be distinguished between a 7^{th} major (10 diatonic tones higher than the base tone) and a 7^{th} dominant (11 diatonic tones higher than the base tone) chord. Appendix D, Table 2 shows the additional note that needs to occur to fulfil the requirements of this chord. For Example, when the C major triad contains an additional dominant seventh, a "B*b*" is added. The chord would change to Cmajor dom7, or commonly expressed C dom7. In this case the *BaseNote* is "C", *Triad* is "major" and *SeventhChord* is "dominant7".

There might be special tones that occur additionally, that are relevant only to some music information retrieval applications. In this case the *NoteAdded* can be used to add more tones to the chord. The value specified is the number of diatonic tones above the base note of the chord. Therefore, if somebody wants to add a "D" to a "C major" chord, its value would be 2. Note that *NoteAdded* must not be used to indicate tones that are already indicated by the Triad or *SeventhChord* descriptors. The element *NoteRemoved* is embedded in a sequence as well. With this it is possible to delete tones from the chord. Its values are specified as the number of diatonic tones above the base tone. If somebody wants to discard the major third (for example "E" at the "C major" triad) from the chord, the value of this descriptor would be 4. Note that *NoteRemoved* must not be used to indicate tones that are not indicated by Triad or *SeventhChord*. *NoteRemoved* must also not be used to indicate tones that are added using *NoteAdded*.

Extraction of Chords out of a polyphonic audio signal

The tonal components are extracted from the frequency domain representation of the signal under investigation. The separation of those tonal components from noise and percussive occurrences is crucial to the effectiveness of the consecutive process. From the result of the previous stages, the notes are determined by eliminating harmonic overtones that stem from the characteristic of the instrument playing the note under consideration. The individual notes are then mapped to chords in order to determine the underlying chord pattern and to find the dominant chords for a segment of music.

Extraction of Prominent Tonal Components

The author's focus on using transformations with a variable frequency resolution providing higher density of frequency bins in the low frequency range and a lower density at high frequencies, respectively. The extraction method that has been implemented and tested is a derivative of the Warped FFT.

Separating Tonal Components from Transients and Erasing Overtones

The tonal components of interest usually possess a longer temporal duration than the transient components and a focussed occurrence in the frequency range. Thus a straightforward approach is to perform the separation based on these properties.

As a first step, all local maxima in the frequency range are identified that lie above a certain amplitude threshold. Secondly, a minimum length criterion in temporal direction is applied to the candidates found in step one. It must be stated, that in some cases, tonal components are erased also, e.g. if they are extremely short.

To perform a reliable harmonic analysis from a frequency domain representation of a musical signal, it is critical to identify the overtones and to use only the fundamentals for the further processing.

Determination of Chords

The chord analysis is based on a pattern matching method. The tonal events are compared with reference vectors representing different chord types. They consist of ones where tones exist in the related chord type and zeros elsewhere, e.g. the vector for major chords has a one in the first, fifth and eighth cell.

A matrix of twelve rows representing the tones of the western scale is built from the input. For every time frame the amplitudes of the tonal events are written into a column of this matrix. If a tone appears in more than one octave within one frame, the amplitudes for the corresponding row and column are summed up giving this tone a higher weight for calculation. Every column of the matrix is then successively compared with all reference vectors by calculating the scalar product. The input columns are shifted, so that all nonzero elements are in the first cell once for processing the comparison. This way all possible chord inversions are checked. The chord type belonging to the reference vector that yields the highest scalar product is chosen and the respective key-note is deduced from the number of shifting operations.

The presented method has the significant advantage of being very easily upgradeable to other chord types by simply adding their reference vectors to the algorithm.

6.10.2 AudioChordPatternDS

The ChordPattern DS is a description scheme which allows a compact representation of chord patterns. The internal structure of the representation is dependent on the proper chord structure of the pattern that has to be represented.

6.10.2.1 Syntax

```
<!-- ################################################################ -->
<!-- Definition of AudioChordPatternDS                                -->
<!-- ################################################################ -->
<complexType name="AudioChordPatternDS">
  <extension base="mpeg7:AudioDSType">
    <sequence>
      <element name="Scale" type="mpeg7:ScaleType" use="required" maxOccurs="1"/>
      <element name="Key" type="mpeg7:KeyType" use="required" maxOccurs="1"/>
        <sequence>
          <element name="Chord" type="mpeg7:ChordType" use="required"
                   maxOccurs="1"/>
          <element name="AudioSegment" type="mpeg7:AudioSegmentType"
                   maxOccurs="1"/>
        <sequence>
      <sequence>
    </all>
  </extension>
</complexType>
```

6.10.2.2 Semantics

Name	Definition
AudioChordPatternType	A representation of a chord pattern of an audio signal.
Key	A container type containing degree, alteration, and mode of a popular western music note.
Chord	The actual chord element.
AudioSegment	The time when the pattern starts and the duration of it.

6.10.2.3 Instantiation requirements

In order to guarantee a proper instantiation of this description scheme, the following requirements have to be fulfilled:

- The element *TimePoint*, as provided by the *AudioPatternType*, must be instantiated.
- The element *Duration*, as provided by the *AudioPatternType*, must be instantiated.
- The number of elements of *Chord*, as provided by *ChordBaseType* must be greater than 0.
- The number of elements of the scale used in Key must be the same scale as in ChordType
- If not scale is implemented, the default scale has to be taken into consideration.
- The value of the first sequence in *ChordType::RelativeChordNumber* must be “1”.

- The element *Key* must implement only one element from *KeyNote*.

6.10.2.4 Usage and Extraction (informative)

6.10.2.4.1 Representation of Chord Pattern

In common western popular music often chord progression patterns that repeat but change slightly can be found, for example the chords C major, F major, G major and again C major. Musically this issue is called cadence. But the same cadence can be reached with the chords “D major”, “F# major” and “A major” because it consists of keynote, dominant, subdominant and again the keynote and is independent of its actual key. To compare two different chord patterns it is therefore mostly not relevant whether the base chord is “C major” or “D major”. It is more important that the successive chords have the same distance from the first chord. Therefore *AudioChordPatternDS* has been designed not to state the absolute key number in *ChordType* but its relative distance to the first occurred chord. That means if the chord progression starts with “D major”, the *RelativeChordNumber* in *ChordType* must be 0. In this case Key would be “D”. If the following chord is “A major”, *RelativeChordNumber* in *ChordType* must be 7, because in this scale “A” is in the default scale 7 diatonic tones above the “D”. Therefore it is easy to compare chord progression patterns independent of their keynote. The following image (Figure 2) shows the issue in comparison to a musical chord progression pattern with a C major cadence in example.

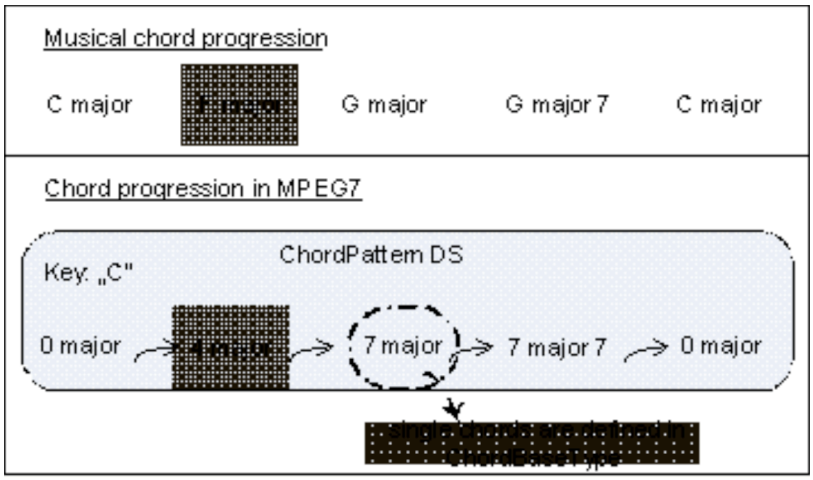

Figure 2 — Chord progression in MPEG 7 in comparison to a musical chord progression

6.10.2.4.2 Chord Pattern example

The following example shows an XML schema using the *ChordPattern* descriptor with the chords “C major”, F major, “G major“ and then again “C major” (cadence of C major).

```
<AudioDescriptor xsi:type="ChordPatternType">
  <Scale>1 2 3 4 5 6 7 8 9 10 11 12</Scale>
  <Key>
    <KeyNote>C</KeyNote>
  </Key>
  <Chord>
    <RelativeChordNumber>1</RelativeChordNumber>
    <AudioSegment>
      <MediaTime>
        <MediaTimePoint>T00:00:00</MediaTimePoint>
        <MediaDuration>PT1M30S</MediaDuration>
      </MediaTime>
    </AudioSegment>
  </Chord>
  <Chord>
    <RelativeChordNumber>6</RelativeChordNumber>
    <AudioSegment>
      <MediaTime>
        <MediaTimePoint>T00:00:10</MediaTimePoint>
        <MediaDuration>PT1M30S</MediaDuration>
      </MediaTime>
    </AudioSegment>
  </Chord>
  <Chord>
    <RelativeChordNumber>8</RelativeChordNumber>
    <AudioSegment>
      <MediaTime>
        <MediaTimePoint>T00:00:20</MediaTimePoint>
      <MediaDuration>PT1M30S</MediaDuration>
    </MediaTime>
  </AudioSegment>
  </Chord>
  <Chord>
    <RelativeChordNumber>1</RelativeChordNumber>
    <AudioSegment>
      <MediaTime>
        <MediaTimePoint>T00:00:30</MediaTimePoint>
        <MediaDuration>PT1M30S</MediaDuration>
      </MediaTime>
    </AudioSegment>
  </Chord>
</AudioDescriptor>
```

6.10.2.4.3 Automatic Retrieval and Classification

A similarity matching of chord progression patterns could easily be done by comparing the chord progression patterns without considering the actual key. A standard mean square metric may then be used for evaluating the degree of similarity between two chord patterns.

6.10.2.4.4 Applications

Automatic Retrieval and Recommendation

When using chord patterns to describe a certain musical mood, it is possible to query for musical content that is also characterized by one ore more representative chord patterns. This mechanism can serve as a search criterion in applications proposing a number of musical titles belonging to a particular musical style or genre or sounding similar to a title being proposed by the user as a “reference sound”.

New ways of composing and arranging music

The possibility to search for musical excerpts with a particular chord may have a positive influence on the process of composing and arranging new music. Musicians or DJs that have a specific idea concerning the harmony could “pre-listen” how this was realized in different existing arrangements by other composers.

6.11 Scales & Modes

6.11.1 Pitch Profile DS

The *PitchProfile* DS is an enhanced representation of monophonic melodies, based on the *Melody* DS and extended to allow a fine description of musical scales and modes on which melodies are built. The *PitchProfile* DS integrates the *Scale* type (part *of Melody* DS), which provides a description of the tuning system used by the melody, and extend it to capture the relative importance of the various components of a scale, such that the scale can be identified.

6.11.1.1 Syntax

```
<!-- ######################################## -->
<!-- Definition of PitchProfile DS            -->
<!-- ######################################## -->
<complexType name="PitchProfileType">
  <complexContent>
    <extension base="mpeg7:MelodyType">
      <sequence>
        <element name="ReferencePitch" type="mpeg7:ReferencePitchType"
                                       minOccurs="0"/>
        <element name="TransposingRatio" type="mpeg7:float" minOccurs="0"
                                         default="12"/>
        <element name="ProfileWeights" type="mpeg7:ProfileWeightsType"
                                       minOccurs="0"/>
      </sequence>
    </extension>
  </complexContent>
</complexType>
```

6.11.1.2 Semantics

Name	Definition
PitchProfileType	A structure containing optional elements that support the description of the mode on which the melody is built. The mode can be either a traditional basis scale (e.g. the major scale) or any mode derived from this basis scale (e.g. Dorian, Phrygian, etc.).
ReferencePitch	A container for the base pitch of the mode, either in absolute frequency expressed in unit of Hz, or a pitch marker using a semitone scale, in case the mode is related to a precise western music temperament.
TransposingRatio	A ratio of frequency expressed in semitones, specifying the point at which the mode repeats. The default value of this descriptor is taken to be 12. A *TransposingRatio* value of 0 (zero) should be understood as describing a musical mode that does not repeat. See section 7.1.8.2 for more details.
ProfileWeights	An array containing the relative prevalence (cumulative duration) of each pitch class (or each degree) of the mode. The following formula (informative) is applicative: $ProfileWeights[n] = \max\left(\frac{Prevalence[n]}{Prevalence[0]}, 1\right)$, where *Prevalence*[*i*] is the cumulative duration of i^{th} degree (starting from index zero).

6.11.1.3 Instantiation requirements

In order to guarantee a proper instantiation of this description scheme, the following requirements have to be fulfilled:

- In any case the *Scale* D of the *Melody* DS must be instantiated as soon as the *ProfileWeights* D is instantiated.

- Although the *Scale* D and the *ProfileWeights* D are both float vectors addressing melody degrees features. They do not have necessarily the same length, but the i^{th} element of the *ProfileWeights* vector corresponds to the i^{th} element of the Scale vector. In most cases, the last element of the *Scale* D is the *TransposingRatio*, which is equivalent to the base pitch regarding its hierarchical position in the scale or in the mode. There is therefore no need to attach a weight to this last element. For an instantiation example see section 6.11.1.5.

6.11.1.4 Usage and extraction (Informative)

6.11.1.4.1 Automatic retrieval and recommendation

When using musical scales and/or modes to refer to a particular musical style or genre it is possible to query for musical content that is also characterized by one ore more representative scales and/or modes. This mechanism can serve as a search criterion in applications proposing a number of musical titles belonging to a particular musical style or genre or sounding similar to a title being proposed by the user as a “reference sound”.

6.11.1.4.2 Usage of the transposing ratio

In most cases the tuning system will be the equal-tempered scale and the transposing ration will be 12. In other words a given musical scale or a given musical mode repeats itself after 12 semitones, i.e. after an octave. However the last value of the Scale vector should *only* be understood as the frequency ratio after which the *tuning system* repeats itself, whereas the transposing ratio of a given mode built on this tuning system can be different:

- Considering for instance a scale built on the repetition of a minor second and a major second. Such a scale has been used and defined by the 20^{th} century French composer Olivier Messiaen as a “limited-transposition mode”, meaning that it can only be transposed (semitone by semitone) a small number of times, in opposition to traditional scales (e.g. the major scale that can be transposed 12 times). Such a mode can be described with the *PitchProfile* DS by assigning a non-zero weight to degrees 1, 3, 4, 6, 7, 9, and 10, and using a *TransposingRatio*’s value of 12. An alternative description consists of using a *TransposingRatio*’s value of 3 and a *ProfileWeights* vector of length 2, with a non-zero weight assigned to its first element. With this latter representation the degrees 0 (base degree), 3, 6 and 9 on one hand, and 1, 4, 7 and 10 on the other hand, are considered as equivalents, and the *TransposingRatio* reflects the “limited-transposition mode” property, in accordance to the composer’s thought.

- Considering now a scale built on the following repetitive intervals: a minor third, a major third and another minor third (or in semitones intervals, 3-4-3). Starting from C1 such a scale would be: C1, Eb1, G1, Bb1, Db2, F2, Ab2, B2, D#3, etc, and would reach its octave transposition only after 6 octaves. It can easily be described with the *PitchProfile* DS, by using a 9-length *ProfileWeights* vector, assigning a non-zero weight to degrees 3 and 7, and using a *TransposingRatio*’s value of 10.

6.11.1.4.3 New ways composing and arranging music

The possibility to search for musical excerpts with a particular musical scale or mode may have a positive influence on the process of composing and arranging new music. Musicians, Producers, Composers, or Disc Jockeys that have a specific idea concerning the musical scale or mode could “pre-listen” how this was realized in different existing arrangements by other composers.

6.11.1.4.4 Extraction (Informative)

Whereas automatic extraction from polyphonic data is still an open issue, the *PitchProfile* DS can be extracted from monophonic audio files, under the assumption of the equal-tempered tuning system with of *TransposingRatio*'s value of 12. The extraction procedure consists in the following steps:

- *Extraction of instantaneous fundamental frequencies.* For 44.1 kHz sampled files significant results have been obtained with a hop size of 256 and a 2940-length analysis window, for a frequency output range of 60 Hz to 11.025 kHz. The fundamental frequency is therefore estimated 172 times per second. This extraction can be easily performed with the YIN algorithm[1].

- *Detection of attacks and pitch segmentation.* The pitch standard deviation is computed inside a fixed-length window, moving along the output of step 1. An attack is detected each time the standard deviation falls under a given threshold; a release is detected each time the standard deviation rises above the threshold. The output of step 1 is therefore indexed with attack and release timestamps.

- *Pitch quantization and pitch class prevalence computation.* To each attack-release interval the closest theoretic pitch to the interval's median frequency is assigned. The spectrum of pitches is then reduced to pitch classes modulo the *TransposingRatio*. For each pitch class the *prevalence* (cumulative duration) is computed; the set of all pitch classes and the corresponding prevalence vector is called *pitch profile*.

- Selection of the *base pitch*. The *ReferencePitch* is chosen to be the "most important pitch of the profile". The pitch profile is compared to theoretic profiles in order to select the reference. For major and minor melodies, Krumhansl and Kessler's key finding profiles[2] give satisfying results.

- Computation of *ProfileWeights*. Once the base pitch has been identified, the *ProfileWeights* descriptor is instantiated according to the formula given in section 6.11.1.2.

6.11.1.5 Example (Informative)

The following example is an excerpt from the song "Moon River" by Henry Mancini:

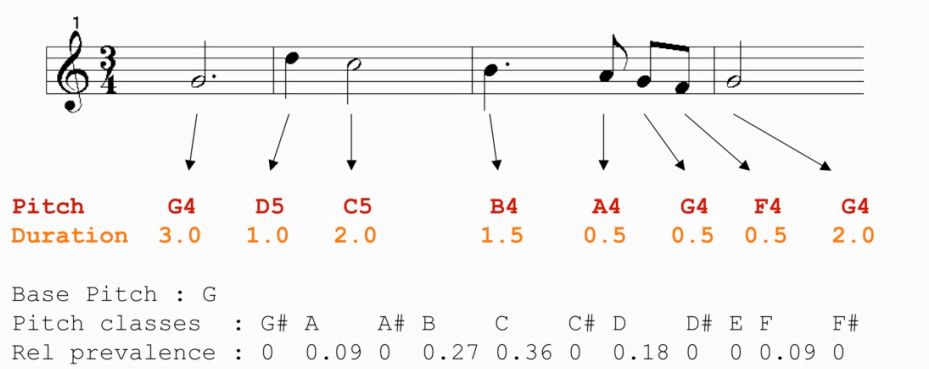

Figure 3 — PitchProfile description of "Moon River"

---

[1] de Cheveigné, A., and Kawahara, H. (2002). "*YIN, a fundamental frequency estimator for speech and music*", J. Acoust. Soc. Am., 111, 1917-1930.

[2] Krumhansl, C. L. (1990). "*Cognitive Foundations of Musical Pitch*". New York: Oxford University Press.

Using *PitchProfile DS*, it could be instantiated in the following manner:

```
<AudioDescriptionScheme xsi:type="PitchProfileType">
  <Scale>1 2 3 4 5 6 7 8 9 10 11 12</Scale>
  <ReferencePitch>
    <BasePitchNumber>7</BasePitchNumber>
   </ReferencePitch>
  <TransposingRatio>12</TransposingRatio>
  <ProfileWeights>0 0.09 0 0.27 0.36 0 0.18 0 0 0.09 0</ModeWeights>
</AudioDescriptionScheme>
```

6.11.2 ReferencePitch

The *ReferencePitch* contains information about the “base pitch” or “base frequency” of the mode, i.e. the note on which the whole mode is built. The reference can be described either in absolute frequency expressed in unit of Hz, or with a pitch marker using a semitone scale. For instance, the reference of the Phrygian mode derived from the C-major scale (C D E F G A B) will be E; the reference of the minor pentatonic mode derived from the C-major pentatonic scale (C D E G A) would be A.

6.11.2.1 Syntax

```
<!-- ################################### -->
<!-- Definition of ReferencePitch D      -->
<!-- ################################### -->
<complexType name="ReferencePitchType">
  <complexContent>
    <extension base="mpeg7:AudioDType">
      <sequence>
        <element name="BaseFrequency" type="float" minOccurs="0"/>
        <element name="BasePitchNumber" type="mpeg7:PositiveInteger"
                                        minOccurs="0">
      </sequence>
    </extension>
  </complexContent>
</complexType>
```

6.11.2.2 Semantics

Name	Definition
ReferencePitchType	The pitch (or frequency) on which the mode is built (i.e. the fundamental degree of the mode), expressed either in absolute frequency or with a positive integer referring to a given pitch class (see below).
BaseFrequency	The absolute frequency of the first degree of the mode, expressed in units of Hertz, when the tuning system used is NOT the equal-tempered context.
BasePitchNumber	The pitch class of the first degree of the mode, when the tuning system used is the equal-tempered context. The *BasePitchNumber* is an MPEG-7 positive integer, with the following correspondence: 0=”C”, 1=”Db”, 2=”D”, etc. Note that enharmonic features are therefore not considered here.

6.11.2.3 Usage and extraction (informative)

6.11.2.3.1 Content retrieval

The *ReferencePitch* offers a new criterion for music browsing. Music composers for instance might have a strong interest in searching for musical excerpts with a particular reference pitch. Consider a composer writing a piece in Bb major, he/she might ask a search engine to find music files in which the melody uses Bb as the reference pitch.

Another good example is novice instrumentalists looking for small pieces for music studying. Considering for instance that a young violoncellist, who only knows the first position, is unable to play an A-flat without any finger extentions. The *ReferncePitch*, combined with the *ProfileWeights*, can help him finding music pieces where the A-flat never appears. Indeed the following example indicates the presence in the melody of pitches D, F, G, A and C, in other words the D minor pentatonic scale, which does not use any A-flat. Typically such a scale can be played on the first position of the cello, without any extensions.

```
<AudioDescriptionScheme xsi:type="PitchProfileType">
  <Scale>1 2 3 4 5 6 7 8 9 10 11 12</Scale>
  <ReferencePitch>
    <BasePitchNumber>2</BasePitchNumber>
   </ReferencePitch>
  <TransposingRatio>12</TransposingRatio>
  <ProfileWeights>0 0 0.4 0 0.6 0 0.3 0 0 0.7 0</ModeWeights>
</AudioDescriptionScheme>
```

6.11.2.3.2 Extraction (informative)

See section 6.11.1.4.4.

6.11.3 ProfileWeights D

The *ProfileWeights* descriptor is intended to describe which degrees of a given tuning system are used by a melody, and the relative importance of each degree to the melody's reference degree.

6.11.3.1 Syntax

```
<!-- ################################################ -->
<!-- Definition of ProfileWeights D                   -->
<!-- ################################################ -->
<complexType name="ProfileWeightsType">
  <simpleContent>
    <restriction base="mpeg7:floatVector"/>
  </simpleContent>
</complexType>
```

6.11.3.2 Semantics

Name	Definition
`ProfileWeightsType`	Array of weights describing the relative importance of each degree in the mode. Each weight is defined as the relative prevalence (or cumulative duration) of each pitch class (or each degree) of the mode. The following formula (informative) is applicative: $ProfileWeights[n] = \max\left(\frac{prevalence[n]}{prevalence[0]}, 1\right)$, where *Prevalence*[*i*] is the cumulative duration of ith degree of the mode (starting from index zero).

6.11.3.3 Usage and Extraction (informative)

6.11.3.3.1 Designing distances on the melodies' space

In the equal-tempered context a straightforward distance on the descriptors' space is the *cosine* distance. Is this case the mode descriptor is reduced two a 11-dimensions vector of weights and the following formula (informative) applies for a cosine distance between melodies A & B:

$$d(A,B) = \frac{\sum_{i=1}^{11} a_i b_i}{\sqrt{\sum_{i=1}^{11} a_i^2 \sum_{i=1}^{11} b_i^2}}$$

where $(a_i)_{1 \le i \le 11}$ and $(b_i)_{1 \le i \le 11}$ are instances of the *ProfileWeights* descriptor for melodies A & B.

With such a distance melodies may easily be grouped into clusters and sorted by order of similarity. The following issues are addressed:

- "Given an input melody, I want an engine search to recover all the melodies in a given database that bear some significant resemblance with it, on a scale or mode similarity criterion."
- "Given a database of melodies, can I extract some clusters that have a musical meaning in terms of scale or mode similarity?"
- "Given a set of melodies, can I construct relevant semantic links on this set? On which criterions can I say that melody A is closer to melody B than melody C?"

6.11.3.3.2 Extraction (informative)

See section 6.11.1.4.4.

6.12 Enhanced AudioSignature

In ISO/IEC 15938-4 (MPEG-7 Audio) the AudioSignatureType description scheme became standardized in order to recognise music. The recognition performance of this descriptor is sufficient for the vast majority of applications like broadcast monitoring, recognition of personal audio on PCs etc. There are, however, commercially interesting applications which require a reliable recognition of extremely distorted audio signals, like GSM-coded cell phone signals, which are not accommodated by the current AudioSignatureType.

6.12.1 Syntax

```
<!-- ################################################################ -->
<! -- Definition of EnhancedAudioSignatureDS                         -->
<! --################################################################ -->
<complexType name="EnhancedAudioSignatureType">
 <complexContent>
   <element name="Envelope" type="mpeg7:AudioSpectrumEnvelopeType"
                                  minOccurs="1" maxOccurs="1"/>
   <element name="Flatness" type="mpeg7: AudioSignatureType"
                                  minOccurs="0" maxOccurs="1"/>
 </complexContent>
</complexType>
```

6.12.2 Semantics

Name	Definition
EnhancedAudioSignatureType	A structure containing a condensed representation as a unique content identifier for an audio signal for the purpose of automatic identification for very robust audio signals
Envelope	The spectrum envelope of the signal of AudioSpectrumEnvelopeType data.
Flatness	The spectrum flatness of the signal of AudioSignatureType data

6.12.3 Instantiation requirements

In order to constitute a valid EnhancedAudioSignatureType descriptor, the following requirements have to be satisfied:

- The syntax of the Envelope part is restricted to SeriesOfVectorType.
- The log field of the Envelope part has to be instantiated, as provided by the SeriesOfVectorType syntax.
- The logBase parameter, as provided by the SeriesOfVectorType syntax, is fixed at 10.0
- The loEdge parameter, as provided by the AudioSpectrumAttributeGrp syntax, is fixed at 250 Hz.
- The hiEdge parameter, as provided by the AudioSpectrumAttributeGrp syntax, is fixed at 4000 Hz.
- The octaveResolution parameter, provided by the AudioSpectrumAttributeGrp syntax, must be 1 or less.

There must be no overlap between subsequent calculations. Therefore the `HopSize` is fixed at 30 ms.

The Scaling Ratio as provided by `ScalableSeriesType` can adopt values of 1, 2 or 4.

6.12.4 Usage and Examples (Informative)

There are numerous examples of applications for the *EnhancedAudioSignature* description scheme conceivable, including automatic identification of an unknown piece of audio based on a database of registered audio items. The description scheme is inherited from the AudioSignature description scheme and the basic innovation is the introduction of the AudioSignatureEnvelope type. That enables a broader range of application and quality improvement of previous applications.

The AudioSignature description scheme may be used for identifying slightly distorted signals, but the enhanced descriptor performs much better on strongly distorted signals as for e.g. recorded music from cell phones. The element “Envelope” is based on the feature AudioSpectrumEnvelope and its feature space is orthogonal to that of AudioSignatureDS used feature AudioSpectrumFlatness which enables a lot of other application areas.

A similarity matching of undistorted or slightly distorted query items might be done as in ISO/IEC MPEG 7 described. In order to match similarity between strong distorted signals and the reference signal it is recommended to use the data of AudioSignatureEnvelopeType. A standard mean square metric may then be used for evaluating the degree of similarity between two signatures. In order to improve the performance of recognizing music the difference of adjacent Scalable Series might be calculated.

Remove subclauses 5.2.3. and add following subclauses

5.2.3 SeriesOfScalarType

This descriptor represents a series of scalars, at full resolution or scaled. Use this type within descriptor definitions to represent a series of feature values.

5.2.3.1 Syntax

```
<!-- ################################################################ -->
<!-- Definition of SeriesOfScalar datatype                            -->
<!-- ################################################################ -->
<complexType name="SeriesOfScalarType">
  <complexContent>
    <extension base="mpeg7:ScalableSeriesType">
      <sequence>
        <element name="Raw" type="mpeg7:floatVector" minOccurs="0"/>
        <element name="Min" type="mpeg7:floatVector" minOccurs="0"/>
        <element name="Max" type="mpeg7:floatVector" minOccurs="0"/>
        <element name="Mean" type="mpeg7:floatVector" minOccurs="0"/>
        <element name="Random" type="mpeg7:floatVector" minOccurs="0"/>
        <element name="First" type="mpeg7:floatVector" minOccurs="0"/>
        <element name="Last" type="mpeg7:floatVector" minOccurs="0"/>
        <element name="Variance" type="mpeg7:floatVector" minOccurs="0"/>
        <element name="Weight" type="mpeg7:floatVector" minOccurs="0"/>
        <element name="LogBase" type ="float" default="1" minOccurs="0" />

      </sequence>
    </extension>
  </complexContent>
</complexType>
```

5.2.3.2 Semantics

Name	Definition
`SeriesOfScalarType`	A representation of a series of scalar values of a feature.
`Raw`	Series of unscaled samples (full resolution). Use only if scaling is absent to indicate the entire series.
`Min`	Series of minima of groups of samples. The value of `numOfElements` shall equal the length of the vector. This element shall be absent or empty if the `Raw` element is present.
`Max`	Series of maxima of groups of samples. The value of `numOfElements` shall equal the length of the vector. This element shall be absent or empty if the `Raw` element is present.
`Mean`	Series of means of groups of samples. The value of `numOfElements` shall equal the length of the vector. This element shall be absent or empty if the `Raw` element is present.
`Random`	Downsampled series (one sample selected at random from each group of samples). The value of `numOfElements` shall equal the length of the vector. This element shall be absent or empty if the `Raw` element is present.

First	Downsampled series (first sample selected from each group of samples). The value of numOfElements shall equal the length of the vector. This element shall be absent or empty if the Raw element is present.
Last	Downsampled series (last sample selected from each group of samples). The value of numOfElements shall equal the length of the vector. This element shall be absent or empty if the Raw element is present.
Variance	Series of variances of groups of samples. The value of numOfElements shall equal the length of the vector. This element shall be absent or empty if the Raw element is present. Mean must be present in order for Variance to be present.
Weight	Optional series of weights. Contrary to other fields, these do not represent values of the descriptor itself, but rather auxiliary weights to control scaling (see below). The value of numOfElements shall equal the length of the vector.
LogBase	Base of a logarithm that is performed on the input data. If the value is equal the default value, no logarithm is performed. Note that the value of LogBase must be greater than 0.

Note: Data of a full resolution series (ratio = 1) are stored in the Raw field. Accompanying zero-sized fields (such as Mean) indicate how the series may be scaled, if the need for scaling arises. The data are then stored in the scaled field(s) and the Raw field disappears.

In the case, that the value of LogScale is different from its default value, a logarithm must be performed on the data. The following formula shows the rule for this calculation. *Base* contains the base of the logarithm and is defined in LogScale. In case the logarithmic calculation is invalid (log 0) or the calculated output is smaller than -1.0e^4 the output value is fixed to -1.0e^4.

$$outputValue = \log_{base}(inputValue)$$

Scalable Series allow data to be stored at reduced resolution, according to a number of possible scaling operations. The allowable operations are those that are *scalable* in the following sense. Suppose the original series is scaled by a scale ratio of P, and this scaled series is then rescaled by a factor of Q. The result is the same as if the original series had been scaled by a scale ratio of $N = PQ$.

Figure 3 illustrates the scalability property. This scaled series can be derived indifferently from the original series by applying the scaling operation with the ratios shown, or from the scaled Series of Figure 2 by applying the appropriate rescaling operation. The result is identical. Scaling operations are chosen among those for which this property can be enforced.

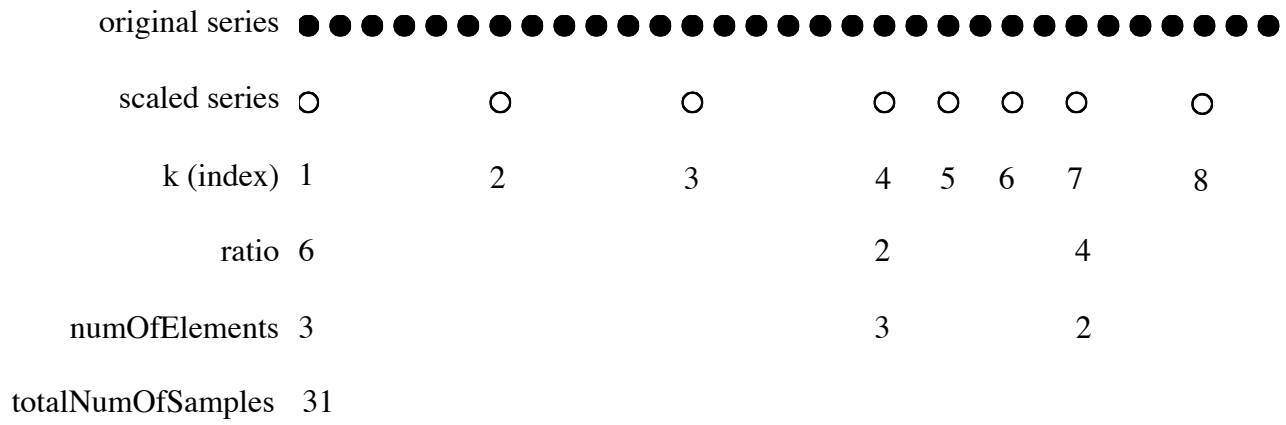

Figure 3 — An illustration of the scalability property

If the scaling operations are used, they shall be computed as follows.

Name	Definition	Definition if Weight present
Min	$m_k = \min_{i=1+(k-1)N}^{kN} x_i$	Ignore samples with zero weight. If all have zero weight, set to zero by convention.
Max	$M_k = \max_{i=1+(k-1)N}^{kN} x_i$	Ignore samples with zero weight. If all have zero weight, set to zero by convention.
Mean	$\bar{x}_k = (1/N) \sum_{i=1+(k-1)N}^{kN} x_i$	$\bar{x}_k = \sum_{i=1+(k-1)N}^{kN} w_i x_i \Big/ \sum_{i=1+(k-1)N}^{kN} w_i$ If all samples have zero weight, set to zero by convention.
Random	choose at random among N samples	Choose at random with probabilities proportional to weights. If all samples have zero weight, set to zero by convention.
First	choose the first of N samples	Choose first non-zero-weight sample. If all samples have zero weight, set to zero by convention.
Last	choose the last of N samples	Choose last non-zero-weight sample. If all samples have zero weight, set to zero by convention.
Variance	$z_k = (1/N) \sum_{i=1+(k-1)N}^{kN} (x_i - \bar{x}_k)^2$ $= (1/N) \sum_{i=1+(k-1)N}^{kN} x_i^2 - \bar{x}_k^2$	$z_k = \sum_{i=1+(k-1)N}^{kN} w_i (x_i - \bar{x}_k)^2 \Big/ \sum_{i=1+(k-1)N}^{kN} w_i$ If all samples have zero weight, set to zero by convention.
Weight	$\bar{w}_k = (1/N) \sum_{i=1+(k-1)N}^{kN} w_i$	

In these formulae, k is an index in the scaled series, and i an index in the original series. N is the number of samples summarized by each scaled sample. In case *logBase* is not equal to the default value, X is the logarithm of the input data, otherwise the raw input data. The formula for `Variance` differs from the standard formula for unbiased variance by the presence of N rather than $N-1$. Unbiased variance is easy to derive from it. If the `Weight` field is present, the terms of all sums are weighted.

Remove subclauses 5.2.5. and add following subclauses

5.2.5 SeriesOfVectorType

This descriptor represents a series of vectors.

5.2.5.1 Syntax

```
<!-- ################################################################ -->
<!-- Definition of SeriesOfVector datatype                            -->
<!-- ################################################################ -->
<complexType name="SeriesOfVectorType">
  <complexContent>
    <extension base="mpeg7:ScalableSeriesType">
      <sequence>
        <element name="Raw" type="mpeg7:FloatMatrixType" minOccurs="0"/>
        <element name="Min" type="mpeg7:FloatMatrixType" minOccurs="0"/>
        <element name="Max" type="mpeg7:FloatMatrixType" minOccurs="0"/>
        <element name="Mean" type="mpeg7:FloatMatrixType" minOccurs="0"/>
        <element name="Random" type="mpeg7:FloatMatrixType" minOccurs="0"/>
        <element name="First" type="mpeg7:FloatMatrixType" minOccurs="0"/>
        <element name="Last" type="mpeg7:FloatMatrixType" minOccurs="0"/>
        <element name="Variance" type="mpeg7:FloatMatrixType" minOccurs="0"/>
        <element name="LogBase" type ="float" default="1.0" minOccurs="0" />
        <element name="Covariance" type="mpeg7:FloatMatrixType"
                 minOccurs="0"/>
        <element name="VarianceSummed" type="mpeg7:floatVector"
                 minOccurs="0"/>
        <element name="MaxSqDist" type="mpeg7:floatVector" minOccurs="0"/>
        <element name="Weight" type="mpeg7:floatVector" minOccurs="0"/>
      </sequence>
      <attribute name="vectorSize" type="positiveInteger" default="1"/>
    </extension>
  </complexContent>
</complexType>
```

5.2.5.2 Semantics

Name	Definition
`SeriesOfVectorType`	A type for scaled series of vectors.
`Raw`	Series of unscaled samples (full resolution). Use only if `ratio`=1 for the entire series.
`Min`	Series of minima of groups of samples. Number of rows must equal `numOfElements`, number of columns must equal `vectorSize`. This element must be absent or empty if the element `Raw` is present.

Name	Definition
Max	Series of maxima of groups of samples. Number of rows must equal numOfElements, number of columns must equal vectorSize. This element must be absent or empty if the element Raw is present.
Mean	Series of means of groups of samples. Number of rows must equal numOfElements, number of columns must equal vectorSize. This element must be absent or empty if the element Raw is present.
Random	Downsampled series (one sample selected at random from each group of samples). Number of rows must equal numOfElements, number of columns must equal vectorSize. This element must be absent or empty if the element Raw is present.
First	Downsampled series (first sample selected from each group of samples). Number of rows must equal numOfElements, number of columns must equal vectorSize. This element must be absent or empty if the element Raw is present.
Last	Downsampled series (last sample selected from each group of samples). Number of rows must equal numOfElements, number of columns must equal vectorSize. This element must be absent or empty if the element Raw is present.
Variance	Series of variance vectors of groups of vector samples. Number of rows must equal numOfElements, number of columns must equal vectorSize. This element must be absent or empty if the element Raw is present. Mean must be present in order for Variance to be present.
LogBase	Base of a logarithm that is performed on the input data. If the value is equal the default value, no logarithm is performed. Note that the value of LogBase must be greater than 0.
Covariance	Series of covariance matrices of groups of vector samples. This is a three-dimensional matrix. Number of rows must equal numOfElements, number of columns and number of pages must both equal vectorSize. This element must be absent or empty if the element Raw is present. Mean must be present in order for Covariance to be present.
VarianceSummed	Series of summed variance coefficients of groups of samples. Size of the vector must equal numOfElements. This element must be absent or empty if the element Raw is present. Mean must be present in order for VarianceSummed to be present.
MaxSqDist	Maximum Squared Distance (MSD). Series of coefficients representing an upper bound of the distance between groups of samples and their mean. Size of array must equal numOfElements. This element must be absent or empty if the element Raw is present. If MaxSqDist is present, Mean must also be present.
Weight	Optional series of weights. Weights control downsampling of other fields (see explanation for SeriesOfScalars). Size of array must equal numOfElements.
vectorSize	The number of elements of each vector within the series.

Most of the above operations are straightforward extensions of operations previously defined in section 5.2.3.2 for series of scalars, applied uniformly to each dimension of the vectors. Operations that are specific to vectors are defined here:

Name	**Definition**	**Definition if `Weight` present**
`Covariance`	$\Box_k^{jj'} = \frac{1}{N} \sum_{i=1+(k-1)N}^{kN} (x_i^j - \bar{x}^j)(x_i^{j'} - \bar{x}^{j'})$	$\Box_k^{jj'} = \sum_{i=1+(k-1)N}^{kN} w_i (x_i^j - \bar{x}^j)(x_i^{j'} - \bar{x}^{j'}) \Big/ \sum_{i=1+(k-1)N}^{kN} w_i$
`VarianceSummed`	$z_k = (1/N) \sum_{j=1}^{D} \sum_{i=1+(k-1)N}^{kN} (x_i^j - \bar{x}_i^j)^2$	$z_k = \sum_{j=1}^{D} \sum_{i=1+(k-1)N}^{kN} w_i (x_i^j - \bar{x}_i^j)^2 \Big/ \sum_{i=1+(k-1)N}^{kN} w_i$ If all samples have zero weight, set to zero by convention.
`MaxSqDist`	$MSD_k = \max_{i=1+(k-1)N}^{kN} \left\| x_i - \bar{x}_k \right\|^2$	Ignore samples with zero weight. If all samples have zero weight, set to zero by convention

In these formulae, k is an index in the scaled series and i an index in the original series. N is the number of vectors summarized by each scaled vector. D is the size of each vector and j is an index into each vector.

If *logBase* is not equal to the default value, $\bar{x}_i^j$ is the mean of the logarithm base *logBase* of N samples and X is the logarithm of the input data. Following formula shows the rule for this calculation. *Base* contains the base of the logarithm and is defined in *LogScale*. If the logarithmic calculation is invalid (log n where n is less than or equal to zerol) or the calculated output is smaller than -1.0e^4 the output value is fixed to -1.0e^4.

$$outputValue = \log_{base}(inputValue)$$

In case *logBase* is equal to the default value, $\bar{x}_i^j$ is the mean of N samples and X are the raw input data.

The various variance/covariance options offer a choice of several cost/performance tradeoffs for the representation of variability.

Add a new Annex after Annex C

Annex D
(normative)

Title

Table D.1 — This table shows the design of chords at different chord modes.

number of diatonic tones from the base tone	0	1	2	3	4	5	6	7	8	9	10	11
sample notes	c	c#	d	d#	e	f	g#	g	g#	a	a#	b
major	1				1			1				
minor	1			1				1				
augmented	1			1			1					
diminished	1				1				1			
suspended 2	1		1					1				
suspended 4	1					1		1				

Table D.2 — This table shows additional notes (marked with 1) on 7th chords.

Major7										1		
Dominant 7											1	