Contents

Standard Music Description Language

1 Scope and Field of Application

This International Standard defines an architecture for the representation of music information, either alone or in conjunction with text, graphics, or other information needed for publishing or business purposes. Multimedia time sequencing information is also supported. The architecture is known as the “Standard Music Description Language”, or “SMDL”.

SMDL is a “HyTime” application; it conforms to International Standard ISO/IEC 10744 – Hypermedia / Time-based Structuring Language (“HyTime”). Specifically, SMDL is a “derived architecture” derived from the HyTime architecture, and SMDL is expressed in this International Standard in a manner which conforms to HyTime’s specifications for the expression of architectures (also known as “meta-DTDs”) and derived architectures.

SMDL is an SGML application conforming to International Standard ISO 8879 – Standard Generalized Markup Language.

1.1 Scope

SMDL is capable of representing many (but not all) genres of music, and most (but not necessarily all) instances of works in those genres. The aim of SMDL is primarily to permit the application neutral interchange of all such music as can reasonably be expressed in common practice music notation, i.e. the written notation commonly used for Western-style art music, dance music, and commercial music. However, music represented in SMDL need not actually have ever been expressed in common Western music notation, or in any other particular notation. The use of SMDL as an abstract music representation does not preclude the rendition of special symbols or other particular notational or performance practices, in either visually or aurally perceivable source or formatted output materials.

By virtue of its total reliance on and compatibility with the HyTime International Standard, music represented in SMDL can include musically dependent or independent multimedia time sequences, such as slide show control tracks or automated lighting information. Combinations of musical and non-musical events, for example, in a dramatic production, are possible as well.

SMDL is designed for flexibility and extensibility. There are no technical prohibitions against the use of some components without the whole, or against the use of user-defined components in conjunction with standardized ones.

SMDL includes a conformance clause that identifies minimum and higher levels of support in terms of standardized architecture components and options for user extensions.

1.2 Field of Application

SMDL is designed for the transmission and storage of music and hypermedia information. It is optimized for completeness, flexibility, and ease of use. There has been some attention to human readability, principally for the benefit of applications implementers, as it is unlikely there will be much direct manual creation of SMDL documents. The architecture is not intended as a foundation for modern computer music composition, although, depending on the compositional and performance process, parts or all of it might be useful for some purposes in some works.

Typical original sources of works that can be represented in SMDL are printed scores, performances recorded in a manner that permits machine transcription, pieces that are already represented in a (non-standard) music description language, and composers using computer-based compositional and/or notational aids which have the capability of exporting the pieces as SMDL.

Multimedia information, such as digital audio recordings, can be associated and synchronized with music and described in SMDL. The multimedia elements will have their own notations and encodings. Standards for such notations and encodings are outside the scope of this International Standard. Examples of mixed music and multimedia applications include: business presentations containing synthesized music, sequenced live sound excerpts, slide change signals, and recorded or synthesized speech; and dramatic productions containing music, lighting control, and digitally recorded sound effects.

2 References

ISO 8879:1986, Information processing – Text and office systems – Standard Generalized Markup Language (SGML).

ISO/IEC 10744:1992, Information technology – Hypermedia/Time-based Structuring Language (HyTime).

3 Definitions

The definitions in ISO/IEC 10744:1992 (HyTime) also apply to this document.

NOTE 1 The terms for SMDL constructs are chosen with care, but some may be different from conventional music terminology, in the following ways:

a) The term may be used in a more restricted (or more general) sense in this International Standard than in common music parlance.
b) The term may refer to an SMDL construct that corresponds to, but is not identical to, a construct in music.
c) The term may refer to a construct from another discipline that is here being applied to music. The term “thread,” for example, refers to a concept which does not have a counterpart in conventional music terminology, but it is a metaphor like the one used when speaking of the “thread” of a story line or argument.

NOTE 2 Terms not defined here are defined when introduced in the body of the document.

3.1 active: As applied to an event, causing an effect, such as the sounding of a pitch. (In simple terms, a note is active if it is audible.) The active portion of an event will often be shorter than its duration. (See articulation.)

3.2 anacrusis: A partial measure that occurs before the first complete measure of a piece. Notes in the anacrusis are usually unstressed.

3.3 analytical domain: The portion of an SMDL work which contains music theoretical analyses.

3.4 analysis: A music theoretical analysis of the piece, such as a Shenkerian analysis. An examination of the piece as opposed to a rendition or notation of the piece.

3.5 articulation: The ratio of the time an event is active to the total duration of the event.

3.6 bibliographic data: Library identification information used to catalog and archive pieces of music (or any other works.)

3.7 cantus: The SMDL representation of the logical domain.

NOTE 3 The literal meaning of “cantus” is a song or melody, especially the principal part. Its appropriateness here derives from “cantus firmus” – the thematic foundation of a historically important contrapuntal compositional process.

3.8 cent: Hundredth of a semitone; a frequency ratio equal to the 1200th root of two.

3.9 chromatic: Conventionally means “half-step-wise.” However, in SMDL, the definition of a half step is not restricted to the definition ordinarily used (i.e., 100 cents, or the smallest available interval in 12-tone equal temperament). In precise SMDL terms, “chromatic” means “gamut-step-wise.”

3.10 common practice music notation: Common graphic music notation employing five-line horizontal staves for pitch and notes whose duration is measured in virtual time, such as is used to notate classical Western music.

3.11 diatonic: Conventionally means “scale-step-wise.” However, in SMDL the definition of a scale step is not restricted to the definition ordinarily used. In precise SMDL terms, “diatonic” means “name-step-wise.”

3.12 meter: That property of music which involves cyclic stress patterns in virtual time. The notational concepts of “measure” and “time signature” are expressions of meter.

3.13 music event: An event of the kind that is normally represented in a musical score; in common practice music notation, for example, a note or rest.

3.14 music time: Time in the conventional musical sense, as measured in beats, quarter notes, etc; also known as “virtual time.”

NOTE 4 In the logical domain of SMDL, music time is measured in virtual time units (VTUs), which are independent of music formatting considerations. For example, an event whose duration is one-quarter of a four-beat stress pattern could be formatted in several ways, e.g., a quarter note in 4/4 time or an eighth note in 2/4 time.

3.15 microtuning unit: The smallest representable pitch difference. (This may often be one cent, but it can be finer or coarser if needed.)

3.16 NIFF: “Notation Interchange File Format.” This ongoing initiative by music publishers and music information processing systems vendors is intended to result in a RIFF-based file format for interchange of music information among several existing music information processing systems.

NOTE 5 As such, NIFF files will probably be regarded as both an exceptionally useful type of formatted output instance of an SMDL document, and as the source format of a plentiful supply of music information that can be converted into SMDL for applications in which NIFF is not ideal. SMDL and NIFF are not really in competition with one another. For all application-neutral archival storage purposes, and for most research purposes, SMDL is the best choice. For delivery of music information quickly and in a maximally useful form in particular application contexts, NIFF is the best choice. (The difference between SMDL and NIFF is similar to the difference between the HyTime International Standard and the HTML hypertext delivery language used in the World Wide Web.)

3.17 normalization: The application of a set of rules during the transcription of a piece into SMDL, so as to ensure that a particular one of the potentially unbounded number of possible valid representations of that piece will always result. (In other words, if several people encode the same piece using the same normalization algorithm, they will get identical results.)

3.18 performance: A particular realization of a piece, either by mechanical means or by a musician.

3.19 pick-up: An anacrusis.

3.20 piece: A musical composition.

3.21 pitch: The fundamental audio frequency of the sound generated by an event; the tone.

NOTE 6 The psychoacoustic distinction between perceived pitch and measured frequency has been deliberately disregarded. This International Standard ignores this subtle distinction, which was first revealed by the Fletcher-Munsen experiments in the early twentieth century, just as musicians almost universally ignore it. As far as SMDL is concerned, the fundamental frequency is the pitch, and vice versa.

3.22 requirement: In this context, a normative part of this International Standard that is required and binding. (This is a special use of the term, which differs from the normal meaning of, e.g., a “user requirement.”)

3.23 reportable SMDL error (RSE): An error in SMDL usage which must be reported by a conforming SMDL application.

3.24 resource: A collection of information, often a list or table, which is globally available, e.g., a pitch gamut definition, which may be referenced by every note in the piece.

3.25 RSE: (See reportable SMDL error.)

3.26 score: A written, printed, and/or displayed piece of music; an edition.

3.27 tuplet: A group of notes whose durations are derived from the duration of a containing element, e.g., a quintuplet (five in the space of n, where n is some ordinary duple or triple subdivision of some integer number of beats).

3.28 work: The SMDL representation of a piece. A musical composition.

4 Notations Used in This International Standard

The notation and presentation conventions of this International Standard are the same as those used in the HyTime International Standard.

HyTime is, in turn, based on SGML (ISO 8879:1986) International Standard, an information markup language and architectural design tool.

NOTE 7 Reading this International Standard in the absence of a detailed understanding of SGML could create the impression that SMDL is extremely (even hopelessly) verbose, especially when compared with less abstract music description languages. This impression would be quite incorrect. SGML provides several facilities for minimizing SGML markup, and, therefore, it is not necessary to address this issue separately in the context of this International Standard. Experiments have demonstrated that SMDL documents can be represented quite tersely and compactly, and yet still readably, if SGML’s markup minimization features are used in appropriate ways.

5 Basic Concepts

5.1 The Four Basic Domains

A single document expressed in Standard Music Description Language represents a single musical work in terms of four basic domains:

logical domain	The logical domain is the basic musical content – the essence from which all performances and editions of the work are derived, including virtual time values, nominal pitches, etc. The logical domain is describable as "the composer's intentions with respect to pitches, rhythms, harmonies, dynamics, tempi, articulations, accents, etc.," and it is the primary focus of SMDL. It can also be described as "the abstract information common to both the gestural and visual domains." The logical domain consists of any number of `cantus` elements.
gestural domain	The gestural domain is comprised of any number of performances, each of which may specify how and when components of the logical domain is rendered in a specific performance, including all the means whereby the performer actually "expresses" (acoustically instantiates) the music (intonation, agogic and dynamic stress, etc.). The gestural domain is perhaps most succinctly described as "the information added by performers," or "how the music actually sounds during particular performances." The gestural domain consists of any number of `perform` elements. Each performance may optionally indicate some or all of the detailed correspondences between itself and some cantus, including, for example, the exact mapping from the virtual timings represented in the cantus to the real timings used in the performance.
visual domain	The visual domain is comprised of any number of scores, each of which somehow specifies exactly how components of the logical domain is rendered visually in some particular printable (and/or displayable) edition, including such graphical details as symbology, symbol sets, fonts, page layout, beaming conventions and exceptions, etc. The visual domain is perhaps most succinctly described as "the information added by human editors, engravers, and typesetters," or "how the music actually looks in some particular edition." The visual domain consists of any number of `score` elements. Each score may indicate some or all of the detailed correspondences between itself and the logical domain.
analytical domain	The analytical domain is comprised of any number of theoretical analyses and/or commentaries, each of which somehow specifies opinions, exegeses, etc. about any or all of the information in the other three domains. The analytical domain consists of any number of `analysis` elements.

The four domains are structured in such a way as to provide the constructs needed to represent music. The structure of each domain is discussed in detail in the following clauses.

NOTE 8 The distinctions between the four domains may not serve all philosophies regarding the expression of musical ideas, but they make sense in light of the uses to which this International Standard will ordinarily be put. In any case, the distinctions between the logical, gestural, and visual domains are a natural outcome of the way music is performed and notated; they have long been implicit in the art of music. Music itself has probably existed since the appearance of *homo sapiens*, and, indeed, it may even be a defining characteristic of humanity. Music written thousands of years ago still exists, although exactly how to perform the older notations is a matter of scholarly controversy. The Western music notation that is commonly used today has been under continuous development for the last thousand years, and there is general agreement about how to perform it. The SMDL concept of the logical domain can be regarded as an answer to the question, "What are the abstractions common to both scores and performances?"

Existing music representations may be converted to SMDL. The process of creating an SMDL document instance is normally a matter of generating a logical domain from a score or a performance, and (optionally) of generating a visual or gestural domain which represents all the correspondences between that score or performance and the logical domain.

The original score or performance remains unchanged by this procedure; the addressing power of the HyTime location addressing model makes it possible to address any object or phenomenon within any information expressed in any notation, including any notation used for storage of gestural information, such as MIDI files or digital audio recordings, and any notation used for page description or graphics, such as SPDL or CGM.

If multiple performances and/or multiple scores specify the correspondences between themselves and a single cantus, direct comparisons between any and all of them are straightforward.

5.2 Mixing Music and Multimedia

This International Standard provides two different techniques for the inclusion of multimedia material in musical works. In one, the multimedia events can be intermixed with the musical events such that they become, in effect, musical notes. Alternatively, they can be assembled in a parallel structure so that they can be accessed separately from the music. This flexibility allows for the selection of a technique or combination of techniques which closely matches the material to be represented.

NOTE 9 For example, in the case of a slide (diapositive) show with musical accompaniment, slide-change events can be combined with the music (the note events) just as though they were notes. Alternatively, in the case of a light control sequence for theater, it may be more natural to create an light control event sequence which refers to music events that may be syntactically elsewhere (i.e., that may appear in other schedules or even other documents).

6 Base Architecture Declaration

6.1 Declaration that SMDL is a Derived Architecture

The SMDL meta-DTD indicates the fact that it is a derived architecture by using the `APPINFO` parameter of its SGML declaration. The keyword `ArcBase` is specified as a sub-parameter of the `APPINFO` parameter of the SGML declaration. It appears in the SGML declaration of the SMDL meta-DTD as follows:

```
APPINFO ArcBase
```

This indicates to the document processing system that there may be “`ArcBase`” processing instructions in the SMDL meta-DTD that will specify the base architectures from which SMDL is derived.

NOTE 10 In effect, this is an invocation of the “Base Architecture” identification facility found in Annex C of the HyTime International Standard. It does not invoke the HyTime architecture.

6.2 Identification of Base Architecture

The SMDL meta-DTD indicates the fact that its architecture is derived from the HyTime base architecture by means of an `ArcBase` processing instruction. In SGML’s reference concrete syntax, it would appear as follows:

```
<?ArcBase HyTime>
```

This processing instruction establishes that HyTime is the SMDL meta-DTD’s local name for one of its base architectures, and that after this processing instruction, the document processing system can expect to find:

a) An SGML `NOTATION` declaration that identifies the architecture definition document (in this case, the HyTime International Standard). Associated with this `NOTATION` declaration must be an attribute definition list (an `ATTLIST`) that declares the support attributes of the architecture. The HyTime International Standard itself defines these attributes, but this attribute definition list must also appear in every meta-DTD derived from HyTime. The list therefore forms a part of the SMDL meta-DTD, and it appears below using SGML’s reference concrete syntax.

b) an SGML general `ENTITY` declaration that identifies the meta-DTD of the base architecture (HyTime), and which may specify values for HyTime’s architectural support attributes.

6.2.1 Declaration of Base Architecture

The following SGML NOTATION declaration points to the HyTime International Standard. The ATTLIST that follows it defines the attribute list of the HyTime notation. These declarations have been copied here from the HyTime International Standard.

```
                 <!-- HyTime Support Declarations -->
<!NOTATION HyTime   PUBLIC "ISO/IEC 10744:1992//NOTATION
                     HyTime Architecture Definition Document//EN"
                  -- A document architecture conforming to the
                     Architectural Form Definition Requirements of
                     International Standard ISO/IEC 10744.  --
>

<!ATTLIST #NOTATION HyTime
                  -- Support attributes for all architectures --
    ArcFormA      -- Attribute name: architectural form --
                  NAME        #FIXED      HyTime
    ArcNamrA      -- Attribute name: attribute renamer --
                  NAME        #FIXED      HyNames
    ArcBridA      -- Attribute name: bridge functions --
                  NAME        #FIXED      HyBrid
    ArcDocF       -- Architectural form name: document element --
                  NAME        #FIXED      HyDoc
    ArcVer        -- Architecture version identifier --
                  CDATA       #FIXED      "ISO/IEC 10744:1992"
                  -- DerArc attributes --
                  -- (Support attributes for derived architectures.) --
    ArcCopy       -- Forms copied from base architecture --
                  NAMES       #IMPLIED
    ArcAlias      -- Aliases for names of copied forms.  Constraint:
                     base name precedes alias. lextype(NAMES). --
                  NAMES       #IMPLIED
                  -- Support attributes for HyTime only --
    hyqcnt        -- Highest quantum count ceiling.  Constraint: power
                     of 2 >= 32 (i.e., minimum value is 32). --
                  NUMBER      #REQUIRED
    base          -- Base module. lextype(NMTOKENS) --
                  CDATA       ""                  -- Default: no options --
    measure       -- Measurement module.  lextype(NMTOKENS). --
                  CDATA       #IMPLIED            -- Default: no support --
    locs          -- Location address module.  lextype(NMTOKENS). --
                  CDATA       #IMPLIED            -- Default: no support --
    links         -- Hyperlinks module.  lextype(NMTOKENS) --
                  CDATA       #IMPLIED            -- Default: no support --
    sched         -- Scheduling module.  lextype(NMTOKENS) --
                  CDATA       #IMPLIED            -- Default: no support --
    rend          -- Rendition module.  lextype(NMTOKENS) --
                  CDATA       #IMPLIED            -- Default: no support --
>
```

NOTE 11 The above declarations appear *verbatim* in this official version of the SMDL meta-DTD. However, in a version of the SMDL meta-DTD used in the context of a particular application, these declarations could conceivably be governed by an SGML declaration that defines a concrete syntax other than SGML's Reference Concrete Syntax. In such a case, they might not look exactly like the above declarations, but they would have the same effect.

6.2.2 Declaration Template for the HyTime Meta-DTD Entity

The following SGML ENTITY declaration points to a file (here named “myhytime.mtd”) containing those formalized SGML aspects of the HyTime International Standard (i.e., those portions of the HyTime meta-DTD), which are actually in effect for this application. An ENTITY declaration having the same effect as this one must appear in every SMDL meta-DTD document.

```
<!ENTITY HyTime
                -- THIS ENTITY IS NOT TO BE REFERENCED! --
    SYSTEM  "myhytime.mtd"  CDATA   HyTime  [

        hyqcnt=32
        base="activity context dcnatts desctxt dvlist exidrefs HyLex
              HyPD lexord lextype refctl unparsed xpropdef"
        measure="axismdu dimref fcsmdu homogran HyFunk HyOp markfun"
        locs="anydtd anysgml uassert bigmatch coordloc HyQ mixcase
              mixspace multloc notsrc pathloc query relloc spanloc"
        links=manyanch
        sched="accanch calspec exrecon grpdex juldate manyaxes
               splitfcs"
        rend="modify patch profun project scaleref"
        ArcCopy="abstime accanch acclink acctype activity axis baton
                 batrule bibloc calspec dataloc date descdef desctab
                 desctxt dimlist dimref dimspec dvlist event evgrp
                 evsched exrecon extent extlist fcs fcsloc granule
                 ilink juldate listloc mallobj markfun marklist
                 modpatch modrule modscope mrkquery nameloc nmlist
                 nmquery notloc pathloc profun projectr proploc
                 proscope relloc rendrule scaleref timeoff treeloc
                 wand wandrule wndpatch"
        ArcAlias="  HyDoc           work
                    fcs             cantus
                    evsched         thread
                    evsched         lyric
                    evgrp           tuplet
                    event           pitched
                    event           rest
                    ilink           perform
                    ilink           score
                    ilink           analysis"
]>
```

7 Musical Work (work) and Work Segment (workseg)

7.1 Virtual time declaration

The following declaration permits cantuses to be specified in virtual (i.e., music) time. The axis element (see Section 8.2) may refer to it.

```
<!NOTATION virtime  PUBLIC -- Virtual Time Unit --
       "+//ISO/IEC 10744//NOTATION Virtual Measurement Unit//EN"
```

```
   >
```

7.2 Declaration of Real Time as a Notation

The following declaration permits `cantuses` to be specified in real time. The `axis` element (see Section 8.2) may refer to it.

```
   <!NOTATION SIsecond PUBLIC -- Real Time --
          "+//ISO/IEC 10744//NOTATION Systeme International second//EN"
   >
```

7.3 Parameter entities affecting the structure of a `work`

7.3.1 SMDL Resources (`SMDLrc`)

The parameter entity **SMDL resources** (`SMDLrc`) lists the architectural forms of elements that can appear anywhere within an SMDL document. These become inclusion exceptions specified in the content model of the `work` element.

```
                             <!-- SMDL Resources -->
                             <!-- use: work -->
   <!entity % SMDLrc        -- SMDL resources; things that can appear
                               anywhere --
          "batrule | rendrule | chordgam | pitchgam | mudef | meterspec |
%links; | %locs;"
   >
```

7.3.2 Work Segment Content Model (`m.wksg`)

The parameter entity **work segment content model** (`m.wksg`) is used to clarify that the basic structure of a music unit of work is the same regardless of whether it constitutes the entire work or a segment of an entire work.

```
                <!-- Work Segment Content Model -->
                <!-- use: work, workseg -->
   <!entity % m.wksg
                            "( workseg+ | ( cantus | perform | score)*)"
   >
```

7.4 Musical Work (`work`)

Derived from the HyTime `HyDoc` architectural form, the SMDL **musical work** (`work`) architectural form is the representation (or set of representations) of a single musical composition, or “piece.” A work encompasses the entire document, and is defined as the logical music information and all of the performances, scores, and analyses that stem from that music information.

If the work consists of a single work segment, no `workseg` element need be used.

```
                <!-- Work as a Whole -->
   <!element work  -- Document element of SMDL documents. --
                - O ( %m.wksg;) +( %SMDLrc;) >
   <!attlist work
```

```
      SMDL        NAME        work
                  -- HyTime HyDoc attributes --
      id          ID          #IMPLIED
      boslevel    -- Bounding level of HyTime bounded object set when
                     document is a hub; overridable at run time.
                     Constraint: 0=no limit; 1=root; 2 to N --
                  NUMBER      #IMPLIED    -- Default: application BOS --
      unmspace    -- Unified name space for IDs and entities --
                  (unified|separate)      separate
      docmdu      -- SMU to MDU ratio that makes the MDU a common
                     denominator of HMUs for all schedules in all FCS
                     that are in a given measurement domain.
                     Constraint: SMU name and ratio for all of the
                     measurement domains used in the document.
                     lextype((NAME,s+,frac),(s+,NAME,s+,frac)*). --
                  CDATA       #FIXED      ""      -- Default: "1 1" for
                                                     each SMU --
                  -- bibinfo attributes --
   >
```

7.5 Work Segment (`workseg`)

The SMDL **work segment** (`workseg`) architectural form is a major section of the work, e.g., a movement, a tableau, a musical number, etc. The order in which work segments appear in the content of a `work` element is the order in which they are intended to be performed and/or visually rendered.

Movements of a symphony should be placed in separate segments, as should acts in an opera. Any other major and subdivisions that are punctuated by an interruption in the flow of music time (as in the case of an intermission or pause between movements), or by major changes in mood, tempo, key, etc., are candidates for containment in `workseg` elements.

NOTE 12 The division of a work into work segments normally expresses the composer's intent that the work be rendered in such a way as to make the divisions clear to the perceiver.

Each work segment can have any number of `cantus` elements in its content, each of which is a complete and independent representation of the work segment. The order in which the `cantus` elements appear is not significant for purposes of this International Standard.

NOTE 13 Normally there is one `cantus` in the content of each `workseg`. If there is more than one, it is usually because there are different sources for the essential music information of the piece, and the sources disagree so much that it is impractical to represent them both in the same `cantus` element, while accounting for the differences by means of *ossias*. Needless to say, as in the case of *ossias*, a performer must choose which `cantus` to render in any given instance.

The attribute **class** (`class`) specifies the type of division or subdivision of which the work segment is an instance.

NOTE 14 For example, `"movement"`, `"trio"`, or `"coda"`.

The attribute **end time modifier** (endmod) specifies what should happen after the last scheduled event in the cantus has been performed, in order to make perceivable the passage from this work segment to the next. If the value is `"pause"`, there should be a pause of some perceivable length. If the value is null (`""`), there should be no end time modification; the performer should continue immediately to the next work segment, without any special indication that the boundary between the two work segments is being crossed.

```
                          <!-- Work Segment -->

   <!element worksseg
                   -- segment of musical work. --
                   - O     (%m.wksg)
   >
   <!attlist workseg
       SMDL        NAME        workseg
       id          ID          #IMPLIED
       class       CDATA       #IMPLIED    -- e.g. movement, section,
                                              musical number.  Default:
                                              not specified. --
       endmod      CDATA       "pause"     -- e.g., "15 minute
                                              intermission," "await
                                              curtain," etc.  --
   >
```

7.6 Attribute List Forms Used Within a Work

7.6.1 Authority Attribute List Form (`authorty`)

The attribute **authority** (`authorty`) identifies the particular score, performance, or analysis of the piece from which its representation as a `cantus`, `perform`, or `score` element was derived. If the `authorty` attribute on a `cantus` element is not specified, the cantus itself is the authoritative souce. If the `authorty` attribute on a `perform` or `score` element is not specified, then the authority is the `cantus` element(s) with which it is associated.

```
                          <!-- authority attributes -->

   <!attlist authorty  -- use: cantus, perform, score --
       authorty    -- reference(s) to the source document(s) or
                      authority(ies) from which this SMDL representation
                      was derived.  --
                   IDREFS      #IMPLIED
   >
```

7.6.2 Bibliographic Information Attribute List Form (`bibinfo`)

The attribute **bibliographic information** (`bibinfo`) refers to a `bibinfo` element (see Section 10) which sets forth the bibliographic information which should be used to catalog and/or describe the element on which the `bibinfo` attribute appears. If it is not defined or no value is specified for the `bibinfo` attribute, then the value of the `bibinfo` attribute of the nearest ancestor element with a one, if any, will be used.

The attribute **main thematic material** (`themes`) refers to the main music thematic materials appearing in the element on which the `themes` attribute appears.

```
                     <!-- Bibliographic Information Attributes -->

   <!attlist bibinfo
                   -- use: work, workseg, cantus, perform, score,
                      analysis --
```

```
       bibinfo     -- references to SMDL bibliographic data elements.
                      May be locally overridden in cantus, perform,
                      score, and analysis elements.  reftype(bibinfo). --
                   IDREFS      #IMPLIED    -- Default: get value from
                                              bibinfo attribute of
                                              nearest ancestor, if any.
                                              Otherwise, not
                                              specified. --
       themes      -- pointer(s) to the main thematic materials in
                      content. --
                   IDREFS      #IMPLIED    -- Default: not specified. --
   >
```

8 Logical domain (`cantus`)

The concept of the cantus is based on, and is entirely consistent with, the HyTime **finite coordinate space (FCS)** concept, and with those of its various HyTime-defined substructures and resources.

NOTE 15 To the HyTime FCS concept, SMDL adds one significant enhancement: **stress templates** (see Section 8.7).

The **cantus** architectural form represents the essential music information. It forms the common ground on which all of the performances, scores, and analyses are constructed. It contains the logical musical material as opposed to the performance or score specific material.

8.1 Cantus (`cantus`)

Derived from the HyTime `fcs` architectural form, the SMDL **cantus** (`cantus`) architectural form represents a finite coordinate space with a single axis measured in virtual time.

The `thread` (see Section 8.3) and `lyric` (see Section 8.4) architectural forms in the content of the `cantus` architectural form are derived from the HyTime `evsched` architectural form. The abstract notes and rests of the music are represented in `threads`, and the lyrics, if any, in the `lyrics`. The elements conforming to the HyTime-defined `baton` architectural form (see Section 8.5) are used to describe and/or govern the tempi of actual performances of the `cantus`. The elements conforming to the HyTime-defined `wand` architectural form (see Section 8.6) are used to govern and/or describe the various ways in which instrumental and other sounds may be modified, muted, articulated, etc.

NOTE 16 The music time (virtual time) specifications found in a `cantus` are more abstract than the timing instructions given in a musical score written in common practice notation. The same cantus could, for example, be scored as sixteenth-note triplets in the context of a 4/8 time signature, or as unmodified eighth notes in the context of a 12/8 time signature. The cantus would not offer guidance as to the which time signature should be used; it would only specify that a 4-beat stress pattern is in effect, and that each of the notes has a duration of one third of a beat.

It is interesting to note that common practice music notation has undergone a largely inexplicable centuries-long process sometimes called “metric notational inflation,” in which ordinary performance practice has been to play the same note value as a longer and longer real-time duration, while composers have been notating their music using shorter and shorter note values. What was long ago called a *minim* – the shortest note – is now called a half-note, which is 128 times longer than the shortest notes Beethoven used; what used to be called a *brevis* (meaning “short”) is now known as a “double whole note,” the longest note value used in common practice notation (and only rarely), and what used to be called a “longa” is never seen any more; it is too large to appear in a modern measure. All this tends to support the idea that an abstract representation of music, such as is used in SMDL, may be particularly useful for archival purposes. Such a representation will be immune to changes in notational and performance practices.

The use of this International Standard does not ensure that the cantus of a given piece must always be represented in some particular way. If desired, an algorithm can be applied which translates any arbitrary cantus into some canonical form. The attribute **normalization algorithm** (norm) states which algorithm (if any) has been used to normalize the cantus. The user may create such an algorithm to fit the needs of the application. The normalization algorithm must be declared as an SGML NOTATION.

```
                      <!-- Structure of the Cantus -->

<!element cantus - O (thread | lyric | baton | wand)* >
<!attlist cantus
    SMDL        NAME        cantus
                -- bibinfo attributes --
                -- authorty attributes --
    norm        -- Normalization according to the named algorithm
                   (declared as a notation) has been applied. --
                NOTATION    #IMPLIED    -- Default: not normalized --
                -- HyTime fcs attributes --
    id          ID          #REQUIRED
    fcsname     -- Name of semantic FCS described by the element --
                NAME        #IMPLIED    -- Default: FCS is unnamed --
    fcsmdu      -- SMU to MDU ratio that makes the MDU a common
                   denominator of HMUs for all schedules on all axes
                   of FCS in a given measurement domain.  Constraint:
                   SMU name and ratio for one or more of the domains
                   used in the FCS.  An SMU name can occur only once.
                   lextype((NAME,s+,frac),(s+,NAME,s+,frac)*). --
                CDATA       #FIXED      ""      -- Default: equal to
                                                   docmdu --
    axisdefs    -- Definitions of FCS axes.  Constraint: GIs of axis
                   element types.  lextype(GIL). --
                NAMES       #FIXED      virtime
>
```

8.2 Axis (axis)

The HyTime architectural form **axis** (axis) is used to specify that the one-dimensional finite coordinate space described by an SMDL cantus is measured in virtual time and occupies some number of virtual time units. (See the HyTime standard for details.)

NOTE 17 The calendar attribute of the HyTime axis architectural form is not used in most musical contexts. However, pieces do exist that are intended to be played (or that were played) at a particular date. If such pieces are represented in SMDL, they will need to use the HyTime calspec architectural form. Since calspec is used for such exceptional pieces exactly as it appears in HyTime, there is no need to describe it in the context of this International Standard. It is available to users of SMDL by virtue of the value of the ArcCopy attribute of the declaration of the HyTime meta-DTD as an ENTITY in the SMDL meta-DTD (see Section 6.2.2).

NOTE 18 What follows is an example of how an axis might be declared in specific circumstances.

The parameter entity %tactvtu; gives the number of virtual time units (VTUs) per tactus (beat). It should be an integer that would allow all named note values to be calculated without remainders. The value given below allows equal division of the beat into 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 12, 14, 16, 18, 20, 21, 24, 25, 27, 28, 30, 32, etc. equal parts. It divides even a *largo* tempo (mm=30 beats per minute) into over 40,000 parts per second, each of which is of sufficiently short duration as to be imperceptible. Given this default value and the minimum hyqcnt value of 32 (i.e., 4,294,967,295 quanta), a given cantus's time axis will allow 53,261 beats. CCARH's 1991 issue of *Computing in Musicology* cites Don Byrd's Guinness-esque "list of [music] notational records" as giving 1,154 measures as the longest movement. Assuming a generous 8 beats per measure, we would need only 9,232 beats, giving a comfortable factor of 5 for expansion.

```
<!ENTITY % tactvtu "80640" >
The following parameter entity declarations could be used in
conjunction with the above %tactvtu; entity:
```

```
<!ENTITY % av.cxdm  -- Could use this as the value of the axisdim
                       attribute of an axis measured in music time.
                       (Maximum possible value given hyqcnt=32.) --
                    "4294967295"
>
<!ENTITY % av.cxbg  -- Could use this as the value of the basegran
                       attribute of threads, batons, and wands. --
                    "vtu"
>
<!ENTITY % av.cxgh  -- Could use this as the value of the gran2hmu
                       attribute of threads, batons, and wands. --
                    "1 1"
>
<!ENTITY % av.cxpg  -- Could use this as the value of pls2gran
                       attribute of threads, batons, and wands. --
                    "1 %tactvtu;"
>
<!ENTITY % av.cxfm  -- Could use this as the value of the docmdu
                       attribute (of work), the fcsmdu attribute (of
                       cantus), and/or the axismdu attribute (of
                       axis). --
                    "virtime 1 1"
>
```

8.3 Thread (`thread`)

Derived from the HyTime `evsched` architectural form, the SMDL **thread** (`thread`) architectural form is a schedule of music events. The purpose of `thread` is to allow the `cantus` to be separated, in any fashion that may be convenient, into concurrent, overlapping, sequential, and/or disjunct streams and/or lists of notes and other events.

NOTE 19 For example, threads may be used to encapsulate musical parts and/or voices.

NOTE 20 As in HyTime event schedules, there is no requirement that the events in a `cantus` appear in the order in which they are intended to be rendered. However, it is often syntactically very convenient to do so, especially in music, and it is certainly normally advisable to do so, if only because to do otherwise would be counter-intuitive and contrary to longstanding musical tradition.

The entity `&e.music;` is a list of music event types. A musical note is represented by the event type `pitched`. A chord change of the kind typically notated by common chord symbols or thoroughbass is represented by a `chordchg` element. A musical rest is represented by the event type `rest`.

NOTE 21 As always in any application of an architecture it is possible to use a more constrained list of element types than the one given as the content of the `%e.music;` parameter entity. However, it should also be noted that the `&e.music;` parameter entity allows indefinite expansion of the number of event types, including all manner of multimedia event types, since the full generality of the HyTime `event` architectural form is available (see Section 8.3.3.5).

The attribute **nominal instrument** (`nominst`) specifies the kind of music performing resource (e.g., `"1st violin"`) associated with the events in the `thread`.

NOTE 22 The remaining attributes are defined by the HyTime International Standard.

```
                              <!-- Thread -->

<!entity % e.music  -- musical events and event groups --
    "tuplet | ces | pitched | rest | chordchg | event" >

<!element thread    -- schedule of music events and event groups. --
                    - O (%e.music;)*
>
<!attlist thread
    SMDL        NAME        thread
                -- HyTime sched attributes --
    axisord     -- Order of axes in schedule.  Constraint: GIs of axis
                   element types. Omitted GI means all events in
```

```
                  schedule fully occupy the omitted axis.
                  lextype(GIL). --
               CDATA       #FIXED      ""      -- Default: axisdefs
                                                  in FCS --
      apporder    -- Order of schedule elements is significant to the
                  application and must be preserved --
               (order|disorder)        order
      sorted      -- Representation of schedule elements is sorted by
                  order of position on axes of schedule --
               (sorted|unsorted)       unsorted
               -- HyTime schdmeas attributes --
      basegran    -- Base granule for each axis.  lextype(words) --
               CDATA       #FIXED      ""      -- Default: SMU for
                                                  each.  --
      gran2hmu    -- Granule to HMU ratio for each axis.  Constraint: 1
                  ratio per axis, or 1 for all.  lextype(fracs). --
               NUMBERS     #FIXED      "1 1"
               -- HyTime overrun attributes --
      overrun     -- Handling of dimension that overruns range --
               (error|wrap|trunc|ignore)       error
               -- HyTime pls2gran attributes --
      pls2gran    -- Pulse to granule ratio for each axis.  Constraint:
                  1 ratio per axis, or 1 for all.  lextype(fracs) --
               NUMBERS                         "1 1"
      nominst     -- nominal instrument(s) or other performing
                  resource(s) allocated to this thread. --
               CDATA       #IMPLIED    -- Default: not specified. --
   >
```

8.3.1 Tuplet (`tuplet`)

Derived from the HyTime `evgrp` architectural form, the SMDL `tuplet` architectural form is used to represent non-duple (or, in certain compound meter contexts, non-triple) subdivisions of virtual time duration.

In all events after the syntactically first event, it is an RSE if the first marker is explicitly specified.

NOTE 23 The use of HyTime `evgrps` is fully described in the HyTime International Standard.

```
                     <!-- Tuplet -->
   <!element tuplet
                    - O (%e.music;)
   >
   <!attlist tuplet
      SMDL        NAME        tuplet
      id          ID          #IMPLIED    -- Default: none --
      grpscope    -- Group scope (nominal extent of group).  For each
                  axis, the dimension is the lowest first quantum of
                  any group member through the highest last quantum
                  of any group member on that axis.  All of a
                  member's extents are included.  reftype(extent). --
               IDREF       #IMPLIED    -- Default: to be calculated --
      grpdex      -- Group derived extent specification.  Use for
```

```
                    resizing, rearrangement, repetition.  Group members
                    are given the derived extents.  If nominal extent
                    is also wanted, it must be specified as one of the
                    derived extents.  Constraint: multiple references
                    concatenated.  reftype(extlist). --
                 IDREFS      #IMPLIED    -- Default: no derivation --
                 -- pls2gran attributes --
      pls2gran   -- Pulse to granule ratio for each axis.  Constraint:
                    1 ratio per axis, or 1 for all.  lextype(fracs). --
                 NUMBERS     "1 1"
  >
```

8.3.2 Cantus event sequence (ces)

The SMDL `ces` architectural form is used to associate certain special semantics with arbitrary groups of syntactically sequential music events.

NOTE 24 “Syntactically sequential” does necessarily not mean “sequentially scheduled.” The starting quantum of an event is determined by its extent specification, which may, if the first marker is defaulted (i.e., not explicitly specified), be determined by the event’s syntactic position (i.e., it will begin after the end of the preceding event). If the first marker is specified, however, the event’s syntactic position has no effect on its scheduled extent.

NOTE 25 The musical concept of *chord*, characterizable as a phenomenon in which several notes have the same start time and often share the same performing resource, is represented in SMDL by means of a `ces` in which all the events it contains have the same nominal start time.

The attribute **ornamentation style** (`ornstyle`) is used to indicate the style(s) of ornamentation (perhaps by naming a historical period (e.g. `"late Baroque"`, performance styles (e.g. `"double-dotted French overture"`, `"Mississippi delta"`, `"Hank Williams"`, `"south Indian classical"`), etc.

The attribute **pitch gamut** (`pitchgam`) specifies which pitch gamut will be used to look up pitch names. The gamut selected will remain in effect until another occurrence of this attribute.

The attribute **musica ficta gamut** (`fictagam`) specifies which musica ficta gamut will be used to look up gamut step adjustments to be applied to pitch names. The music ficta gamut selected will remain in effect until another occurrence of this attribute.

The attribute **microtuning unit definition** (`mudef`) identifies the definition that will apply when a quantity of microtuning units is specified. The definition selected will remain in use until another occurrence of this attribute.

The attribute **divisi** (`divisi`) indicates whether simultaneous notes, if any, should be allocated among the available performing resources, or whether all simultaneous notes should be played by all available performing resources.

The attribute **arpeggio** (`arpeggio`) indicates whether any events scheduled to occur simultaneously with the earliest-scheduled event should be arpeggiated, and, if so, whether they should be arpeggiated so that the lowest pitch is played first (`rollup`) or last (`rolldown`).

The attribute **grace** (`grace`) indicates whether or not these notes are grace notes.

NOTE 26 If the notes are grace notes, they may be rendered differently from the other notes in a score, often at a smaller point size.

The attribute **choice** (`choice`) is used to indicate whether all of the subelements should be performed, or just one of them.

NOTE 27 An *ossia* is represented by an event group with a choice of “one.” An application could adopt a convention that the first element in the group is the preferred choice.

The attribute **choice criteria** (`choicrit`) supplies information that might be useful to a performer making the choice. The value of this attribute is ignored unless the value of the `choice` attribute is `one`.

NOTE 28 For example, `"Play the first choice the first time, and the second choice the second time."`.

The `conloc` attribute can be used, for example, to allow passages to be repeated with some variation without recopying their entire contents. (See the HyTime International Standard for details.)

The attribute **repeats** (`repeats`) gives the number of times the passage should be repeated. If the value is `"1"`, there are no repeats. If the value is greater than 1, each repeat will begin on the quantum immediately following the last quantum occupied by any event during the previous cycle. The events occupying the earliest occupied quantum of the `ces` also form the beginning of each repeat, and, during any repeat cycle, the start-times and end-times of all events bear the same numerical quantum relationships to each other that they had in the first cycle.

NOTE 29 In other words, even where absolute addressing was used to establish the dimensions of events during the first cycle, all dimensions of all events will be relative to each other, and bear the same numerical quantum differences between each other, in all repeat cycles. Among the many possible uses for `cess` whose `repeat` attribute’s value is greater than 1 is to notate *Alberti bass* passages compactly.

```
                     <!-- Cantus Event Sequence -->
<!element ces
                    - O (%e.music;)
>
<!attlist ces
    SMDL        NAME        ces
    id          ID          #IMPLIED    -- Default: none --
    ornstyle    -- Ornamentation style: e.g., period --
                CDATA       #IMPLIED
    pitchgam    -- Current pitch gamut --
                IDREF       #IMPLIED    -- Default: value of
                                           syntactically previous
                                           pitchgam attribute. --
    fictagam    -- current ficta gamut --
                IDREF       #IMPLIED    -- Default: value of
                                           syntactically previous
                                           fictagam attribute. --
    mudef       -- governing microtuning unit definition --
                IDREF       #IMPLIED    -- Default: value of
                                           syntactically previous
                                           mudef attribute. --
    divisi      -- either allocate simultaneous notes among performing
                   resources, or make everybody play everything. --
                (tutti|divisi)          tutti   -- default: everybody
                                                   plays everything --
    arpeggio    -- Either arpeggiate or don’t arpeggiate.  Applies
                only to the onset of the events scheduled to start
                together, and only when that starting time is the
                earliest scheduled start time in the entire ces.  --
                (rollup|rolldown|noroll)
```

```
                                                noroll
                                                        -- default: no
                                                           arpeggiation. --
      grace       -- are the contained events grace notes? --
                  (grace|nograce)            nograce
                                                     -- default: these
                                                        are not grace
                                                        notes --
      choice      -- Is only one subelement, chosen by the performer, to
                     be played at any one time, or are all of them to be
                     played?  (attribute for handling ossia, 1st/2nd
                     ending, etc. --
                  (all|one)                      all
      choicrit    -- Criteria by which choice should be made. --
                  CDATA       #IMPLIED    -- default: no choice needs to
                                             be made (choice=all) --
      conloc      -- ID of schedule, evgrp, or event to be used in place
                     of, or as content of, this element (see HyTime
                     International Standard). --
                  IDREF       #CONREF
      repeats     -- Number of repetitions beginning at end of last
                     quantum occupied. lextype(unzi). --
                  CDATA                      1
   >
```

8.3.3 Music Event Types (**`rest`**, **`pitched`**, **`chordchg`**)

8.3.3.1 **`%a.event;`** parameter entity

The parameter entity `%a.event;` defines the attributes used in both pitched events and in rests. All of these attributes are fully defined in the HyTime International Standard.

```
   <!entity % a.event "
                  -- HyTime event attributes --
      id          ID          #IMPLIED       -- Default: none --
      exrecon     -- Extent reconciliation rule if object won't fit.
                     reftype(exrecon) --
                  IDREF       #IMPLIED       -- Default: none --
                  -- HyTime exspec attributes --
      exspec      -- Extent specification.  Constraint: multiple
                     references concatenated.  reftype(extlist). --
                  IDREFS      #REQUIRED
                  -- HyTime pls2gran attributes --
      pls2gran    -- Pulse to granule ratio for each axis.  Constraint:
                     1 ratio per axis, or 1 for all.  lextype(fracs) --
                  NUMBERS                    "1 1"
                  -- HyTime conloc attribute --
      conloc
                  IDREF       #CONREF        -- Default: in content --
   ">
```

8.3.3.2 Rest Event (**`rest`**)

The architectural form **rest** (`rest`) is an imperceivable event. It has duration in music time, but it has no content whatsoever.

NOTE 30 In fact, there is no difference between a `rest` event occupying some quanta and simply having no event occupying the same quanta. If, however, HyTime's first marker defaulting mechanism is used, `rests` are useful as placeholders, so that the first marker of the succeeding event can be defaulted to the quantum immediately after the last quantum of the `rest`.

```
                            <!-- Rest -->
<!element rest  -- Imperceivable music event --
                - O     ( extlist?)
>
<!attlist rest
    SMDL        NAME        rest
    %a.event;
>
```

8.3.3.3 Pitched Note Event (`pitched`)

The architectural form **pitched note event** (`pitched`) specifies the duration and pitch of a musical note.

NOTE 31 `pitched` is the normal way to represent a musical note in SMDL.

The `nompitch`-form element (see Section 8.8.1) in the content of a `pitched`-form element is the means whereby `pitched` events gain access to virtually all of the musical pitch specification apparatus of SMDL (see Section 8.8).

The architectural form **variation** (`vary`) specifies a variation of the pitch, such as a vibrato or portamento, which can occur before, during, and/or after the nominal pitch. The variation is given as a formula.

The attribute **detune value** (`detune`) specifies how far "out of tune" the pitch should be played. The value is given in microtuning units (see Section 8.8.4).

NOTE 32 Detuning and/or variation need not be distinguished from the nominal pitch when it is not convenient or sensible to do so. In many cases, these alterations are more usefully dealt with as part of the timbre of an instrument, or as inherent properties of the nominal pitch, as in some computer music. In the latter case, the formula for the event type would account for any detuning or variation.

The **tie** (`tie`) attribute specifies the successor event to which this event is tied. It is an RSE if the first quantum occupied by the successor event is not immediately adjacent to the last quantum occupied by this event. The purpose of `tie` is to indicate that the successor event is to be perceived as the continuation of the current event.

NOTE 33 The exact method by which the successor event is to be made to be "perceived as the continuation of the current event" is not specified. However, it seems reasonable to expect that the onset articulation of the succeeding event should not be rendered, or at least be made as imperceivable as possible.

```
                         <!-- Note Event -->
<!element pitched
                -- A musically-pitched note.  Used in: %e.music; --
                - O (extlist?, nompitch, vary?)
>
<!attlist pitched
    SMDL        NAME        pitched                 -- HyTime: event --
    %a.event;
    detune      -- Signed number of MUs to add or subtract.  Default:
```

```
                     play in tune (0).  lextype(snum). --
                  NMTOKEN                     0
      tie         -- Immediate successor to which this event is tied--
                  IDREF       #IMPLIED        -- Default: not tied --
   >
   <!element vary  -- Variation in pitch value of note, expressed as a
                     formula.  --
                  O O (#PCDATA)
   >
   <!attlist vary
      SMDL        NAME        vary
      notation
              -- Data content notation.  (Actual notation names are
                 application-specific).  (attribute defined by HyTime)
                 --
                  NOTATION    #REQUIRED
      conloc      -- Formula source (if not in content). --
                  IDREF       #CONREF
   >
```

8.3.3.4 Chord change

A chord change is a specification of a sonority in terms of its intervallic structure (as opposed to actual simultaneous notes, which would be represented by a cantus event group.) It is normally used in works that require (or permit) improvisation. Its purpose is to convey a harmonic environment rather than the presence of particular notes.

In SMDL, chord changes are event types. There are two kinds: common chord symbols (see Section 8.8.5.3) and figured bass (thoroughbass) (see Section 8.8.5.4). Both include a reference pitch (the root or bass) and a chord body consisting of a set of intervals, but there are some differences between the two stemming from their normal application: common chord symbols are normally used in popular music and jazz, while figured bass is used chiefly in Baroque music. The visual representation of a common chord symbol is usually an alphameric name (like C7), while that of figured bass is (as its name implies) a series of numbers that represent intervals from the bass pitch.

(For an explanation of the `tie` and `detune` attributes, see Section 8.3.3.3.)

```
                        <!-- Chord Change Event -->
   <!element chordchg  -- Chord change event --
                       - O ( extlist?, ( ccschord | figbass)?)?
   >
   <!attlist chordchg
      SMDL        NAME        chordchg
      %a.event;
      detune      -- Signed number of MUs to add or subtract.  Default:
                     play in tune (0).  lextype(snum). --
                  NMTOKEN                     0
      tie         -- Immediate successor to which this event is tied--
                  IDREF       #IMPLIED        -- Default: not tied --
   >
```

8.3.3.5 Event (`event`)

The HyTime `event` architectural form should be used to schedule anything that needs to be scheduled in terms of music, or in the context of music. The value of the `SMDL` attribute should be `event`, and there need not be a `HyTime` attribute. See the HyTime International Standard for details.

8.4 Lyric (`lyric`)

The architectural form **lyric** (`lyric`) is a time-based sequence of syllables that are sung to the music of a thread. The dimension specifications of the lyric events (“sung syllable events”) are specified with dimension references to express the synchronization with the thread events.

The attribute **thread** (`thread`) identifies the thread to which this lyric is to be sung.

```
                               <!-- Lyric -->
<!element lyric     -- Time-ordered sequence of sung syllables.
                       Synchronized with a thread. --
                    - O ( syllable | rest)+
>
<!attlist lyric
    SMDL        NAME        lyric
    thread      -- Notes to which lyric is sung --
                IDREF       #REQUIRED
                -- HyTime sched attributes --
    axisord     -- Order of axes in schedule.  Constraint: GIs of axis
                   element types. Omitted GI means all events in
                   schedule fully occupy the omitted axis.
                   lextype(GIL). --
                CDATA       #FIXED      ""      -- Default: axisdefs
                                                   in FCS --
    apporder    -- Order of schedule elements is significant to the
                   application and must be preserved --
                (order|disorder)        order
    sorted      -- Representation of schedule elements is sorted by
                   order of position on axes of schedule --
                (sorted|unsorted)       unsorted
                -- HyTime schdmeas attributes --
    basegran    -- Base granule for each axis.  lextype(words) --
                CDATA       #FIXED      ""      -- Default: SMU for
                                                   each.  --
    gran2hmu    -- Granule to HMU ratio for each axis.  Constraint: 1
                   ratio per axis, or 1 for all.  lextype(fracs). --
                NUMBERS     #FIXED      "1 1"
                -- HyTime overrun attributes --
    overrun     -- Handling of dimension that overruns range --
                (error|wrap|trunc|ignore)       error
                -- HyTime pls2gran attributes --
    pls2gran    -- Pulse to granule ratio for each axis.  Constraint:
                   1 ratio per axis, or 1 for all.  lextype(fracs) --
                NUMBERS                         "1 1"
>
```

8.4.1 Sung syllable (`syllable`)

The architectural form **sung syllable** (`syllable`) is an event, containing a character text object, whose duration is specified by a dimension reference to the corresponding thread event to which the syllable is sung.

NOTE 34 A single syllable element could comprise more than one actual text syllable, as when it is necessary to sing multiple syllables to a single note.

```
                        <!-- Sung Syllable -->
<!element syllable  -- Sung syllable --
                    - O ( extlist?, sylltext?)?
>
<!attlist syllable
    SMDL        NAME        syllable
    %a.event;
>
```

8.4.2 Syllable text (`sylltext`)

The architectural form **syllable text** (`sylltext`) is a character text object that occurs in a sung syllable event.

```
<!element sylltext  -- Syllable text --
                    - O ( #PCDATA)
>
<!attlist sylltext
    SMDL        NAME        sylltext
>
```

8.5 Baton (`baton`)

The HyTime `baton` architectural form is used in SMDL primarily to govern the “projection” of events from a `cantus` onto another finite coordinate space. In SMDL, the target finite coordinate space is generally measured in real time. (See the HyTime International Standard for more information about the `baton` architectural form.)

8.6 Wand (wand)

The HyTime `wand` architectural form is used in SMDL primarily to govern the application of sound modification techniques to the sound output of musical instruments, etc.

NOTE 35 Articulations, mutes, modulations, filters, etc. are examples of modifications that can be applied.

(See the HyTime International Standard for more information about the `wand` architectural form.)

8.7 Metric Stress Pattern

The concepts of meter, measure, “swing,” etc. are all represented in SMDL by *stress templates*. A stress template defines one cycle of a repeated pattern of stresses, and it can be applied to any number and combination of cantus event sequences.

NOTE 36 For example, a passage in 4/4 time may refer to a template with 4 points, and may mark the first as having maximum stress and the third as having moderate stress. In the case of a complex metric situation, such as a measure of five which is felt as two and three, a nested structure of stress patterns can be used to accurately indicate the feel. (This situation can also be handled using nested event sequences.) If ambiguity is desired however, the measure can be represented as simply five beats.

NOTE 37 The inclusion of meter in the logical domain reflects a conviction that measures refer to a basic logical concept in a significant portion of all music, and that they are not merely a matter for the visual domain.

8.7.1 Stress pattern template

The **stress pattern template** (`stresstem`) represents a single stress pattern which can be applied to cantus event sequences of any length, and which can represent any number of MDUs. The stress pattern contains pairs of elements that relate positions in the pattern to lists of stress information for each position. The positions are listed in ascending order, but the set of defined positions can be incomplete.

These positions are specified in terms of points, rather than VTUs. The relationship of points to VTUs is determined when the template is used, not when it is defined, thus making the template scalable. When the pattern is associated with a cantus event sequence, the relationship between VTUs in the sequence and positions in the pattern (points) is specified. This allows one pattern to be used in several different situations, each having a different duration onto which the pattern is mapped.

The attribute **point count** (`pointcnt`) specifies the number of positions in the pattern. These positions are evenly distributed over the total time occupied by the pattern.

```
                      <!-- Stress Pattern Template -->
   <!element strestem
                   -- Beat cycle definition; dynamic stress pattern.
                      Pointnums in ascending order. --
                   - O     (pointnum, stresval*)*
   >
   <!attlist strestem
       SMDL        NAME        strestem
       id          ID          #IMPLIED
       pointcnt    -- Number of stress points in pattern. --
                   NUMBER      #REQUIRED
   >
```

8.7.1.1 Stress point number

The architectural form **stress point number** (`pointnum`) contains an integer in the range from 1 to the `pointcnt` value. It simply specifies the position in the stress pattern at which the related stress information will be applied.

```
                        <!-- Stress Point Number -->
   <!element pointnum
                   -- Stress point receiving agogic or dynamic stresses.
                      Integer from 1 to pointcnt --
                   O O (#PCDATA)
   >
   <!attlist pointnum
       SMDL        NAME        pointnum
   >
```

8.7.1.2 Stress type values

The architectural form **stress type values** (`stresval`) contains a list of stresses and other modifications to be applied to the events in an event sequence. Alternatively, it can express a relationship to the next level inward in a structure of nested stress patterns.

NOTE 38 The ability to nest stress patterns provides a way to specify complex metric situations.

The attribute **stress template use** (`stresuse`) identifies a stress pattern use element that associates a nested stress pattern with the current point number of the current template.

The architectural form **stress value text** (`strestxt`) specifies an imprecise stress specification as a character string.

```
                        <!-- Stress Type Values -->
<!element stresval
                -- Stresses applied to specified stress point --
                - O (strestxt?, formula?)
>
<!attlist stresval
    SMDL        NAME        stresval
    stresuse    -- ID of stresuse for nested stress pattern. --
                IDREF       #CONREF     -- Default: no nested pattern.
                                           --
>
<!element strestxt
                -- Imprecise stress specification: loud, downbeat --
                - O (#PCDATA)
>
<!attlist strestxt
    SMDL        NAME        strestxt
>
```

8.7.2 Stress pattern use

The architectural form **stress pattern use** (`stresuse`) relates a particular stress pattern to a tempo directive in a baton or to a stress type value in an outer stress pattern in which this one is nested. If the content of the element is a music time duration, the relation is to a tempo directive; if it is a stress pattern duration, the relation is to an outer stress pattern. If the content is empty, the relation is to a tempo directive and the pattern is mapped over the entire duration of the tempo directive.

The stress pattern to be used is referenced by the idr attribute. In a tempo directive, this pattern will remain in effect until changed by another occurrence of a stress pattern use element. To turn off the current stress pattern without starting another, define an empty stress pattern template (one with no content) and reference it.

The content of this element is a dimension specification which specifies the time into which one cycle of the pattern will be mapped. Note that this duration is specified in VTUs (directly or indirectly) for a tempo directive.

NOTE 39 For example, if the pattern is specifying a measure of 4/4 time, and the VTU of the music falls on the quarter note, the stresuse duration will be 4 VTUs. The duration of the music to which this pattern applies will usually be much longer than the pattern duration, so the pattern will be repeated.

The architectural form **stress pattern duration** (`stresdur`) specifies the number of points in the outer pattern into which one cycle of this pattern will be mapped.

The attribute **stress point** (`strespt`) specifies the position in the pattern (in points) at which to start. In other words, the offset to the pick-up portion of the pattern. This will normally be 1 (the beginning), but may be offset in certain special circumstances, such as an anacrusis.

```
                     <!-- Stress Pattern Use -->
<!element stresuse
                -- Use of stress pattern in tempo or outer stress
                   pattern.  Default: entire tempo of tempo
                   directive. --
                - O (dimspec | stresdur)?
>
<!attlist stresuse
    SMDL        NAME        stresuse
    id          ID          #REQUIRED
    idr
            -- ID of stress pattern --
                IDREF       #REQUIRED
    strespt
            -- Stress pt # of pattern on which use begins.  Not 1 only
               if anacrusis. --
                NUMBER                     1
>
<!element stresdur
                -- Duration of one iteration of nested stress pattern.
                   Format: NUMBER of points. --
                   - O (#PCDATA)
>
<!attlist stresdur
    SMDL        NAME        stresdur
>
```

8.8 Pitch Model

The pitch of a note can be specified directly as a frequency, by an expression in a user-defined notation (such as a computer music notelist), as an interval from some reference pitch, or as a named pitch that is defined in a pitch gamut.

In an SMDL pitch gamut, pitch names can be associated with frequencies but are manipulated independently of them. This technique allows a gamut to be treated as an equal-tempered gamut for purposes of modulation regardless of how it is actually tuned. The technique also permits uniform application of gamut-based intervals, either alone or in chord changes. A gamut can be defined with no frequencies associated with the pitch names, with generated equal-tempered frequencies, with specified arbitrary frequencies, or with any combination of the above (e.g., an equal-tempered gamut with one or more pitch classes detuned).

A gamut contains a number of “gamut steps” (analogous to pitch classes), and an equal or lesser number of “name step groups” (analogous to white keys on a piano). One or more “pitch names” can be assigned to each gamut step, in either the same “name step group” (e.g., “C” and “DO”) or different name step groups (e.g., “C#” and “Db”).

Unnamed “gamut steps” (called “musica ficta” in SMDL for purely historical reasons) can be determined via a pitch name and a “gamut step distance” (e.g., C# can be referenced as C with a gamut step distance of +1). The pitch definition associated with such a reference (which may be needed to determine the frequency) can be found as follows:

a) Find the first pitch definition in the gamut definition that contains the pitch name.

b) Add the gamut step number of the pitch definition to the gamut step distance to obtain the target gamut step number.
c) Search the gamut definition in the direction of the distance (down for positive distances, up for negative) and use the first pitch definition with the target gamut step number.

Gamut steps (both named and unnamed) can also be referenced by an interval from a reference gamut pitch name. Such a reference includes both a name step distance and a gamut step distance. The target pitch name is found as follows:

a) Count the name step distance from the name step group containing the reference pitch name to find the target name step group.
b) Add the gamut step distance to the gamut step of the reference pitch to obtain the target gamut step.
c) Starting with the target name step group, find the first pitch definition with the target gamut step. As usual, the search direction depends upon the sign of the gamut step distance. For positive, searching starts with the first definition in the name step group; for negative, it starts with the last.

All of the interval arithmetic is modulo the highest step number (name step or gamut step) in the gamut definition. “Wrapping the gamut” in this way requires raising (or lowering) the frequency of the pitch by an “octave ratio” defined for the gamut.

NOTE 40 Although a pitch gamut can represent a conventional equal-tempered scale in a straightforward way, it is highly generalized, and it can represent arbitrary (even bizarre) pitch systems. This flexibility is because of the following properties:

a) There can be any number (at least one) of name steps and any number (at least one) of gamut steps.
b) There is no inherent connection between frequency and pitch name. Only the names are used in constructing gamut intervals; the frequencies can have any values at all. Ascending names could even have decending frequencies.
c) Each gamut step can have several pitch names, all with the same or different frequencies (or undefined frequencies). For example, two pitches named “C#” and “Db” could be at the same gamut step with the same frequency, at two different gamut steps but with the same frequency, or at the same gamut step with either the same or differing frequencies.
d) There is no need to define a pitch name for every gamut step. The unnamed steps (“musica ficta”) can be accessed via an offset (“ficta adjustment”) from a named pitch. While this mechanism supports conventional accidentals in an intuitive manner, it is also generalized in that it will work even if a pitch name (or even multiple pitch names) is defined for an accidental. It also works for any relationship between named and unnamed gamut steps (6 white keys and 47 black, for example, distributed among the white in any way you like, with any frequencies you may care to assign to them).

8.8.1 Nominal pitch (`nompitch`)

The architectural form **nominal pitch** (`nompitch`) specifies the basic pitch (before detuning). It contains no actual information; it is a structural handle which selects one of three ways of specifying pitch: as a selection from a gamut; as an explicit frequency; or as a just intonation pitch (relative pitch). The `nompitch` form is used only in the content of a `pitched` note event (see Section 8.3.3.3).

```
                        <!-- Nominal Pitch -->
<!element nompitch
                O O ( gampitch | freqspec | jipit)
>
<!attlist nompitch
    SMDL        NAME        nompitch
>
```

8.8.1.1 Gamut-based pitch (`gampitch`)

The **gamut-based pitch** (`gampitch`) architectural form specifies a pitch by reference to the pitch gamut currently in effect. The pitch is looked up by name, and then may be modified by an optional octave and accidental.

```
                      <!-- Gamut-based Pitch -->
<!element gampitch
                -- Named pitch from gamut definition --
                O O (octave?, pitchnm, fictadj?)
>
<!attlist gampitch
    SMDL        NAME        gampitch
>
```

8.8.1.1.1 Octave offset (`octave`)

The architectural form **octave offset** (`octave`) specifies the octave of a named pitch as a signed integer offset from the octave of its gamut. The frequency (if any) of the pitch is multiplied by the octave ratio raised to power of the octave offset. An octave offset of zero refers to the octave of the gamut.

```
                        <!-- Octave Offset -->
<!element octave
                -- Octave offset from named pitch.  Signed integer: 0
                   means pitch is in definition octave.  Default: no
                   change from octave of syntactically previous pitch.
                   lextype( snum). --
                O O (#PCDATA)
>
<!attlist octave
    SMDL        NAME        octave
>
```

8.8.1.1.2 Fictum adjustment (`fictadj`)

The architectural form **fictum adjustment** (`fictadj`) specifies the accidental of a pitch as an offset in gamut steps from the named pitch, as modified by any applicable fictagam.

```
                      <!-- Fictum Adjustment -->
<!element fictadj
                -- Fictum (#/b) adjustment from pitchnm: signed
                   integer.  Default: no adjustment. --
                O O (gamintvl)
>
<!attlist fictadj
    SMDL        NAME        fictadj
>
```

8.8.1.2 Frequency specification (`freqspec`)

The architectural form **frequency specification** (`freqspec`) specifies an unnamed pitch as either an exact frequency in Hertz, or as a formula.

The architectural form **absolute frequency** (`hertz`) specifies an exact frequency in Hertz.

```
                   <!-- Frequency Specification -->
```

```
<!element freqspec
                -- Frequency specification --
                O O (hertz | formula)
>
<!attlist freqspec
    SMDL        NAME        freqspec
>
<!element hertz
                -- Absolute frequency in Hertz (decimal fraction). --
                O O (#PCDATA)
>
<!attlist hertz
    SMDL        NAME        hertz
>
```

8.8.2 Pitch gamut definition (`pitchgam`)

The architectural form **pitch gamut definition** (`pitchgam`) establishes a lookup table that maps pitch names to frequencies, and establishes the intervalic relationship between the pitch names. Typically, a pitch gamut represents a scale.

In particular, a pitch gamut is a set of pitch definitions that each associate a pitch name with a name step, a gamut step, and a frequency. It is the name step and gamut step that represent the intervalic relationships between the pitches.

The ordering of the name steps is represented by the order in which “name step group” elements appear in the pitch gamut. Every pitch definition must occur within one of these groups, which thereby associates each pitch name with a name step.

The ordering of the gamut steps is represented by numbering them explicitly. A pitch name is associated with a gamut step by including a “gamut step number” explicitly in its pitch definition.

The attribute **gamut description** (`gamutdes`) is an arbitrary text field that can be used for identification.

The attribute **highest gamut step** (`highstep`) specifies the highest gamut step number in the gamut. If not specified, it is the highest gamut step number occurring in any pitch definition in the gamut.

NOTE 41 In other words, not every gamut step number need occur in a pitch definition. Or, to put it another way, gamut steps can be unnamed. These unnamed steps correspond exactly to the “musica ficta” (accidentals) of the conventional diatonic/chromatic scale.

This attribute has two purposes:

a) When specifying pitch as a gamut-based interval from a named pitch, the number of gamut steps is taken modulo `highstep` + 1.
b) When generating a default equal-tempered tuning for the gamut, the number of equal divisions in the octave is `highstep` + 1.

NOTE 42 In the conventional equal-tempered scale, there are twelve gamut steps, numbered 0 to 11, so the value of `highstep` is 11.

The attribute **octave ratio** (`octratio`) defines a factor that is used in calculating the frequency of a named pitch when an octave offset is specified for it (that is, when it occurs in an octave other than that of the gamut). It is also used when generating a frequency base for the gamut. The conventional equal-tempered scale uses an octave ratio of 2 (the next note of the same name is twice the frequency.)

NOTE 43 There is no indication of the “absolute” octave number of a gamut (other than by implication from any frequencies that may be defined in it).

```
                   <!-- Pitch Gamut Definition -->
<!element pitchgam
                -- Pitch gamut definition.  If genfreq is omitted, no
                   default frequencies are generated. --
                - O (genfreq?, namestep+)
>
<!entity % FRAC "NUMBERS" -- fraction: numerator, denominator? (=1) -- >
<!attlist pitchgam
    SMDL        NAME        pitchgam
    id          ID          #REQUIRED
    gamutdes    -- Description of gamut. --
                CDATA       #IMPLIED        -- Default:
                                               undescribed. --
    highstep    -- Highest gamut step number: positive integer. --
                NUMBER      #IMPLIED        -- Default: highest
                                               gamstep. --
    octratio    -- Octave ratio ("1 1" = no octaves) --
                %FRAC;                  "2 1"   -- Default: normal
                                                   octaves. --
>
```

8.8.2.1 Generated frequency base (genfreq)

The presence of the architectural form **generated frequency base** (genfreq) indicates that the gamut will automatically generate default equal-tempered frequencies. This means that no frequency in the gamut need be explicitly listed; those omitted will be derived mathematically. If the frequency is stated explicitly, the self-generated frequency will be overridden.

The element contains a particular gamut step and its frequency specification. This is the base around which all others are calculated.

The formula for an equal-tempered scale is:

```
         Pitch[n+1] = Pitch[n] * ((highstep+1) root of (octave ratio))
```

```
                  <!-- Generated Frequency Base -->
<!element genfreq       -- Generated frequency base --
                O O (gamstep, freqspec)
>
<!attlist genfreq
    SMDL        NAME        genfreq
>
```

8.8.2.2 Name step group (namestep)

The architectural form **name step group** (namestep) contains the pitch definitions for all pitches associated with the same name step.

NOTE 44 Typically, their pitch names will reflect this relationship; for example, C, C flat, C sharp, and C double sharp.

The name step groups can be thought of as the logical concept behind scale steps, white keys, or lines and spaces in the staff. In conventional music there will be 7 pitch name groups in the gamut.

If more than one gamut step occurs in the set of pitch definitions comprising a name step group, the one in the first pitch definition of the group is considered the unaltered form of the name step.

```
                          <!-- Name Step Group -->
<!element namestep      -- Name step group: pitches based on same name
                           (C, C#, C##).  In conventional music: 7
                           namesteps (letter names A-G) --
                    - O (pitchdef+)
>
<!attlist namestep
    SMDL        NAME        namestep
>
```

8.8.2.3 Pitch definition (`pitchdef`)

The architectural form **pitch definition** (`pitchdef`) relates a pitch name to a specific chromatic position in the gamut (gamut step) and an optional frequency. The frequency can be specified explicitly, or by reference to another pitch in the gamut and a ratio.

If the frequency is specified, any automatically generated frequency is overridden. For imprecise tunings, specify the pitch name only (e.g., “high”, “medium” and “low” pitches for percussion instruments).

The architectural form **gamut step** (`gamstep`) specifies the chromatic position in the gamut of this pitch name. Conventional music will use 12 gamut steps per octave. (This is equivalent to the Forte analysis pitch class.)

For a simple gamut (i.e., one with no musica ficta), the gamut step can be omitted. Simple gamuts can be used for variant tunings, just intonation tonal centers, tuning systems that do not require modulation, etc. If the gamut step is omitted, it is assumed to be one greater than the previous pitch definition, or zero for the first.

The architectural form **relative frequency** (`relfreq`) specifies a frequency by reference to another pitch in the gamut. The resultant frequency is the referenced frequency times the integer ratio.

NOTE 45 SMDL does not prohibit circular definitions. Applications are responsible for avoiding inconsistencies.

```
                         <!-- Pitch Definition -->
<!element pitchdef      -- Pitch definition (member of a pitch gamut).
                           If gamstep omitted: next sequential (0 if
                           first).  Freqspec overrides generated
                           frequency for gamstep. --
                    - O ( pitchnm, gamstep?, (freqspec | relfreq)?)
>
<!attlist pitchdef
    SMDL        NAME        pitchdef
>
<!element gamstep       -- Gamut step: Non-negative integer (>= 0).
                           Conventional music: 12 gamut steps
                           (chromatic scale). --
                    O O ( #PCDATA)
>
```

```
<!attlist gamstep
    SMDL        NAME        gamstep
>
<!element relfreq       -- Relative frequency with respect to another
                           pitch. --
                    - O (pitchnm, iratio)
>
<!attlist relfreq
    SMDL        NAME        relfreq
>
```

8.8.2.3.1 Pitch name (`pitchnm`)

The architectural form **pitch name** (`pitchnm`) holds the name of the pitch (typically a letter like 'C'). The information in the gamut is keyed by this name. If the pitch name is an empty string, the pitch is musica ficta (unnamed).

```
                         <!-- Pitch Name -->
<!element pitchnm       -- Name of pitch in a gamut --
                    O O (#PCDATA)
>
<!attlist pitchnum
    SMDL        NAME        pitchnum
>
```

8.8.2.4 Musica ficta gamut (`fictagam`)

The architectural form **musica ficta gamut** (`fictagam`) specifies a list of pitch names paired with fictum adjustments (accidentals). The gamstep of a specified pitch is implicitly offset by the given number of gamut steps, in addition to any explicit fictum adjustment that may be specified. This element has the logical function of a key signature.

When this gamut is current, the pitch offsets will be applied to the current pitch gamut. If there is no pitch gamut, or if a pitch does not reference the pitch gamut, then this gamut will have no effect.

```
                      <!-- Musica Ficta Gamut-->
<!element fictagam      -- Musica ficta gamut definition. --
                    - O (pitchnm, fictadj)+
>
<!attlist fictagam
    SMDL        NAME        fictagam
>
```

8.8.3 Interval (`interval`)

The architectural form **interval** (`interval`) specifies a scale difference between two pitches ("gamintvl") or a frequency ratio ("arbintvl" and "jipit").

The interval can be specified in three different ways, depending on the requirements of the music: as a scale interval (such as a "minor third"), as an arbitrary interval (either as a difference in gamut steps or as a frequency ratio), or as the ratio of two just intonation pitches. The scale interval is called a "gamut-based interval" because it is meaningful only in the context of some pitch gamut.

```
                              <!-- Interval -->
   <!element interval      -- Scale difference or frequency ratio of two
                              pitches. --
                        O O (gamintvl | arbintvl | jipitint)
   >
   <!attlist interval
       SMDL        NAME        interval
   >
```

8.8.3.1 Gamut-based interval (`gamintvl`)

The architectural form **gamut-based interval** (`gamintvl`) specifies a scale interval. It consists of a diatonic step value and a chromatic step value. (There is no restriction that these step sizes be those of conventional, equal-tempered music.)

NOTE 46 Both of these values are necessary in order to distinguish between enharmonically equivalent intervals such as “minor third” and “augmented second.”

NOTE 47

a) To determine the resultant pitch, add the name step distance to the base name step, and add the gamut step distance to the base gamut step. The resulting name step, gamut step pair can then be looked up in the pitch gamut.
b) The `gamintvl` “0 0” is a unison.

```
                        <!-- Gamut-based Interval -->
   <!element gamintvl      -- Gamut-based interval: number of name steps
                              and gamut steps between two gamut pitch
                              definitions --
                        O O (nsdist, gsdist)
   >
   <!attlist gamintvl
       SMDL        NAME        gamintvl
   >
```

8.8.3.1.1 Name step distance (`nsdist`)

The architectural form **name step distance** (`nsdist`) specifies the difference in position of two name step groups in a pitch gamut definition.

```
                         <!-- Name Step Distance -->
   <!element nsdist    -- Name step distance. lextype( snum). --
                        O O      (#PCDATA)
   >
   <!attlist nsdist
       SMDL        NAME        nsdist
   >
```

8.8.3.1.2 Gamut-step distance (`gsdist`)

The architectural form **gamut step distance** (`gsdist`) specifies the difference in position of two gamut steps in a pitch gamut definition.

```
                         <!-- Gamut Step Distance -->
   <!element gsdist    -- Gamut step distance. lextype( snum). --
```

```
                    O O      (#PCDATA)
>
<!attlist gsdist
    SMDL        NAME        gsdist
>
```

8.8.3.2 Arbitrary interval (`arbintvl`)

The architectural form **arbitrary interval** (`arbintvl`) specifies a simple interval from a reference pitch in the conventional musical sense. It can be specified as a chromatic step value (there is no restriction that these step sizes be those of conventional, equal tempered music), an integer ratio, or an arbitrary formula.

To determine the resultant pitch:

gamut step distance	Add the gamut step distance to the reference gamut step. The resulting gamut step can then be looked up in the pitch gamut.
integer ratio	Multiply the reference frequency by the integer ratio.
formula	Apply the formula to the reference frequency.

The architectural form **integer ratio** (`iratio`) specifies an integer ratio as two integers (numerator and denominator).

```
                     <!-- Arbitrary Interval -->
<!element arbintvl      -- Arbitrary interval --
                        O O (gsdist | iratio | formula)
>
<!attlist arbintvl
    SMDL        NAME        arbintvl
>
<!element iratio        -- Integer ratio interval. lextype( snzi, s+,
                           snzi); --
                        O O      (#PCDATA)
>
<!attlist iratio
    SMDL        NAME        iratio
>
```

8.8.4 Microtuning unit definition (`mudef`)

The architectural form **microtuning unit definition** (`mudef`) specifies the basic (smallest) unit of pitch used to specify a detuning. It can be set to cents, or to a finer or coarser value as necessary. The value is specified as a division of an arbitrary interval.

The attribute **microtuning unit count** (`mucount`) specifies the size of the division of the interval.

To determine the interval size of one microtuning unit, divide the base interval (as specified by the microtuning unit definition) by the microtuning unit count.

```
<!element mudef     -- Microtuning unit definition --
                    - O (interval)
>
<!attlist mudef
```

```
        id          ID          #REQUIRED
        mucount     -- Number of MUs in the interval --
                    NUMBER      #REQUIRED
    >
    <!attlist mudef
        SMDL        NAME        mudef
    >
```

8.8.5 Just intonation

Just intonation is the use of intervals which are describable as ratios of small integers, as opposed to the mathematically complex equal-tempered intervals found in most music.

Just intonation methodology is fully supported by this International Standard. A pitch can be specified as an exact interval from the any other pitch. This allows context-dependent harmonic development to any arbitrary degree of complexity.

NOTE 48 Due to a phenomenon sometimes called the "Pythagorean comma," it is quite possible for the perceived tonal center of a just-intoned piece to change its frequency. Therefore, in general, a piece using a mixture of gamut-based pitches and just-intoned pitches may pose challenging esthetic and technical difficulties for both composers and performers.

8.8.5.1 Just intonation pitch (`jipit`)

The architectural form **just intonation pitch** (`jipit`) specifies a pitch in just intonation as a base pitch and an optional interval. The base pitch can be specified in the same way as a non-just nominal pitch, or as a reference to another just intonation pitch.

The resultant pitch is determined by offsetting the base pitch by the interval. If no interval was specified, the resultant pitch is the base pitch.

The architectural form **just intonation pitch reference** (`jipitref`) is a reference to a just intonation pitch.

```
                        <!-- Just Intonation Pitch -->
    <!element jipit     -- Just intonation nominal pitch --
                        O O ( ( gampitch | freqspec | jipitref),
                            interval?)
    >
    <!attlist jipit
        SMDL        NAME        jipit
        id          ID          #IMPLIED
    >
    <!element jipitref  -- Just intonation nominal pitch reference --
                        - O EMPTY
    >
    <!attlist jipitref
        SMDL        NAME        jipitref
        idr         IDREF       #REQUIRED
    >
```

8.8.5.2 Just intonation interval (`jipitint`)

The architectural form **just intonation interval** (`jipitint`) specifies a just intonation interval as the relationship between two just intonation pitches. The resulting interval is derived by dividing the second pitch by the first.

```
                    <!-- Just Intonation Interval -->
<!element jipitint  -- Just intonation nominal pitch interval --
                    O - ( jipit, jipit)
>
<!attlist jipitint
    SMDL        NAME        jipitint
>
```

8.8.5.3 Common chord symbol (`ccschord`)

Found only in the content of `chordchg`-form elements (see Section 8.3.3.4), the architectural form **common chord symbol** (`ccschord`) specifies a chord change by means of a reference pitch, a chord body, and an optional bass note. The reference pitch represents the root of the chord. The chord body can be specified directly or via a chord body name from a chord body gamut.

A polychord can be represented by specifying more than one pairing of reference pitch and chord body. They must be specified in descending order by reference pitch.

The attribute **chord body gamut** (`chordgam`) identifies the chord body gamut that will be used to look up chord body names. The selected gamut will remain in use until changed by another occurrence of this attribute.

The architectural form **bass note** (`bassnote`) identifies the lowest note that should be played for this chord change. It is specified as an interval with reference to the first (or only) reference pitch.

```
                    <!-- Common Chord Symbol -->
<!element ccschord  -- Common chord symbol chord change --
                    - O ( ( refpitch, ( chordnm | chordbdy))+,
                        bassnote?)
>
<!attlist ccschord
    SMDL        NAME        ccschord
    chordgam    -- Current chord body gamut --
                IDREF       #IMPLIED        -- Default: no change --
>
<!element bassnote  -- Bass note (may also be in body of chord).  If
                       polychord, interval refers to first
                       refpitch. --
                   - O (gamintvl)
>
<!attlist bassnote
    SMDL        NAME        bassnote
>
```

8.8.5.4 Figured bass (thoroughbass) (`figbass`)

Found only in the content of `chordchg`-form elements (see Section 8.3.3.4), the architectural form **figured bass (thoroughbass)** (`figbass`) represents a thoroughbass chord by means of a reference pitch (the bass) and a chord body that is specified directly as a set of figured bass intervals.

The architectural form **figured bass interval** (`figbsint`) specifies an interval in terms of a name step distance from the reference pitch, optionally modified by a fictum adjustment.

The attribute **interval sustain** (`sustain`) indicates whether the interval is sustained beyond the duration of the current figured bass event. If so, the ID of a succeeding figured bass event is specified and the interval is sustained through the duration of that event.

```
                   <!-- Figured Bass (Thoroughbass) -->
   <!element figbass   -- Figured bass (thoroughbass) chord change --
                       - O ( refpitch, figbsint*)
   >
   <!attlist figbass
       SMDL        NAME        figbass
       id          ID          #IMPLIED
   >

   <!element figbsint  -- Figured bass (thoroughbass) interval --
                       - O ( nsdist, fictadj?)
   >
   <!attlist figbsint
       SMDL        NAME        figbsint
       sustain     -- ID of last figbass over which voice sustains --
                   IDREF       #IMPLIED        -- Default: no sustain --
   >
```

8.8.5.5 Reference pitch value (refpitch)

The architectural form **reference pitch value** (`refpitch`) specifies the reference pitch on which chord body intervals are based. It is specified as a gamut-based pitch in the current pitch gamut.

NOTE 49 The reference pitch represents the “root” of a common chord symbol or the “bass” of a figured bass.

```
                      <!-- Reference Pitch Value -->
   <!element refpitch  -- Reference pitch on which chord intervals are
                          based.  "Root" for CCS chord; "bass" for
                          figured bass. --
                       - O ( gampitch)
   >
   <!attlist refpitch
       SMDL        NAME        refpitch
   >
```

8.8.6 Chord body (chordbdy)

The architectural form **chord body** (`chordbdy`) specifies the body of a chord either precisely, as a set of gamut-based intervals with respect to the reference pitch, or imprecisely, as a character string that describes the interval set (like “major seventh”).

NOTE 50

a) The imprecise and precise representations are not mutually exclusive. Both can be present, thereby providing additional information.
b) For consistency, dyads are treated as chord bodies whose precise specification requires only a single gamut-based interval.

The architectural form **chord text** (`chordtxt`) is a character string that describes a chord body.

```
                         <!-- Chord Body -->
<!element chordbdy  -- Intervals of chord with respect to reference
                       pitch --
                    O O ( chordtxt?, gamintvl*)
>
<!attlist chordbdy
    SMDL        NAME        chordbdy
>

<!element chordtxt  -- Imprecise chord body description (without root
                       or bass) --
                   - O (#PCDATA)
>
<!attlist chordtxt
    SMDL        NAME        chordtxt
>
```

8.8.6.1 Chord body gamut (chordgam)

The architectural form **chord body gamut** (`chordgam`) defines a lookup table that maps chord body names to chord bodies.

NOTE 51 For example, the name “ma7” could be associated with a chord body containing the gamut intervals for a major third, perfect fifth, and major seventh. The gamut entry might look like this (with end-tags omitted by markup minimization):

```
<chordnm>ma7
<chordbdy>
<chordtxt>Major 7th
<gamintvl><nsintvl>02<gsintvl>04
<gamintvl><nsintvl>04<gsintvl>07
<gamintvl><nsintvl>06<gsintvl>11
```

The same entry could also look like this, using one possible type of user-defined short reference markup minimization:

```
ma7  "Major 7th"  02:04, 04:07, 06:11
```

The architectural form **chord body name** (`chordnm`) is a character string that names a chord body.

```
                      <!-- Chord Body Gamut -->
<!element chordgam  -- Chord body gamut --
                    - O ( chordnm, chordbdy)+
>
<!attlist chordgam
    SMDL        NAME        chordgam
    id          ID          #IMPLIED
>

<!element chordnm   -- Chord body name --
                    - O (#PCDATA)
>
<!attlist chordnm
    SMDL        NAME        chordnm
>
```

9 Gestural, Visual, and Analytical Domains

The architecture of the gestural, visual, and analytical domains (respectively represented by the `perform`, `score`, and `analysis` architectural forms) is largely undefined by this International Standard, because the generality, power, and flexibility of the HyTime International Standard's information addressing facilities makes a constrained architecture for these domains unnecessary.

The `perform`, `score`, and `analysis` architectural forms are all derived from the HyTime `ilink` architectural form. Each of them links itself to the `cantus` element to which it corresponds.

9.1 HyTime **`ilink`** attributes (**`a.ilink`**)

(An SMDL parameter entity, `a.ilink`, is provided below in order to avoid defining some of the HyTime `ilink` architectural form's attributes explicitly and redundantly in the attribute list definitions of `perform`, `score`, and `analysis`.)

```
<!entity % a.ilink "
        -- HyTime ilink attributes, except anchrole. --
    id          ID          #IMPLIED
                                     -- Default: none --
    linkends    -- Link ends.  Constraint: one anchor per anchor
                   role. If one is omitted, ilink element is first
                   anchor.  Constraint: No HyTime reftype constraints,
                   but application designers can constrain element
                   types with reftype attribute. --
                IDREFS      #REQUIRED
    extra       -- External access traversal rule.  Constraint: one
                   per anchor or one for all.  lextype( ( "E" | "I" |
                   "A" | "N" | "P"), ( s+, ( "E" | "I" | "A" | "N" |
                   "P"))*). --
                NAMES       #IMPLIED    -- Default: no HyTime
                                           traversal --
    intra       -- Internal access traversal rule.  Constraint: one
                   per anchor or one for all.  lextype( ( "E" | "I" |
                   "A" | "N" | "P"), ( s+, ( "E" | "I" | "A" | "N" |
                   "P"))*). --
                NAMES       #IMPLIED    -- Default: no HyTime
                                           traversal --
    endterms    -- Link end term information.  Constraint: one per
                   anchor or one for all.  reftype(HyBrid). --
                IDREFS      #IMPLIED    -- Default: none --
    aggtrav     -- Traversal of agglink anchors: agg or members.
                   Constraint: one per anchor or one for all.
                   lextype( ( "AGG" | "MEM" | "COR"), ( s+, ("AGG" |
                   "MEM" | "COR"))*). --
                NAMES       agg
        -- HyTime conloc attribute. --
    conloc      -- ID of element to be used in place of, or as content
                   of, this element (see HyTime International Standard). --
                IDREF       #CONREF
">
```

9.2 Gestural Domain (**`perform`**)

The `perform` element links exactly one performance to one or more corresponding `cantus` elements. The performance can be in any notation, and it must appear in a “performance notation container” element, which may appear in the content of the `perform` element. This “performance notation container” element can (but need not) be in the content of the `perform` element.

NOTE 52 Examples of notations for performances include NIFF, MIDI files and digital audio recordings.

HyTime `ilink`-form elements in the content of the `perform` element can be used to illustrate the correspondences between the performance and the cantus at any level of detail. At the least detailed level, the `perform` architectural form itself is derived from the HyTime `ilink` architectural form, and it forms the bond between itself, the “performance notation container” element, and the `cantus` element(s) to which the performance as a whole corresponds.

The “performance notation container” element need not be in the content of the `perform` element. It must use a HyTime `notation` attribute to indicate the data content notation used, and it must have a unique identifier. The `perform` element specifies the “performance notation container” element as its linkend corresponding to the `performance` token in the value of its `anchrole` attribute.

```
<!element perform
                    - - ANY
>
<!attlist perform
    SMDL        NAME                    "perform"
    anchrole    -- Anchor roles.  Constraint: one per anchor.
                   lextype( ( NAME, s+, ( RNI, "AGG")?), ( s+, NAME,
                   s+, ( RNI, "AGG")?)+). --
                CDATA       #FIXED      "perform notation #AGG cantus #AGG"
    %a.ilink;   -- authorty attributes --
>
```

9.3 Visual Domain (`score`)

The `score` element links exactly one printable and/or displayable edition corresponding to one or more `cantus` elements. The edition can be in any notation, and it must appear in a “edition notation container” element, which may appear in the content of the `score` element. This “edition notation container” element can (but need not) be in the content of the `score` element.

NOTE 53 Examples of notations for editions include NIFF, SPDL, GIF, and JPEG.

HyTime `ilink`-form elements in the content of the `score` element can be used to illustrate the correspondences between the edition and the cantus at any level of detail. At the least detailed level, the `score` architectural form itself is derived from the HyTime `ilink` architectural form, and it forms the bond between itself, the “edition notation container” element, and the `cantus` element(s) to which the edition as a whole corresponds.

The “edition notation container” element need not be in the content of the `score` element. It must use a HyTime `notation` attribute to indicate the data content notation used, and it must have a unique identifier. The `score` element specifies the “edition notation container” element as its linkend corresponding to the `edition` token in the value of its `anchrole` attribute.

```
<!element score
                    - - ANY
>
<!attlist score
```

```
    SMDL        NAME                    "score"
    anchrole    -- Anchor roles.  Constraint: one per anchor.
                   lextype( ( NAME, s+, ( RNI, "AGG")?), ( s+, NAME,
                   s+, ( RNI, "AGG")?)+). --
                CDATA       #FIXED      "score edition #AGG cantus #AGG"
    %a.ilink;   -- authorty attributes --
>
```

9.4 Analytical Domain (analysis)

The structure of the analysis architectural form is completely in the province of the music theorist(s), musicologist, or music critic who create(s) it. Such an author can use the full power of the HyTime addressing and linking facilities to make extremely precise observations and comparisions regarding all of the information in cantuses, performances, and scores. The analysis architectural form itself is derived from the HyTime ilink architectural form, and it specifies in its linkends attribute the cantus element(s) to which the analysis as a whole corresponds.

```
<!element analysis
                    - - ANY
>
<!attlist analysis
    SMDL        NAME                "analysis"
    anchrole    -- Anchor roles.  Constraint: one per anchor.
                   lextype( ( NAME, s+, ( RNI, "AGG")?), ( s+, NAME,
                   s+, ( RNI, "AGG")?)+). --
                CDATA       #FIXED      "analysis #AGG cantus #AGG"
    %a.ilink;
>
```

10 Bibliographic Information (bibinfo)

The architectural form **bibliographic information** (bibinfo) contains bibliographic and discographic data for the cataloging of a piece. Bibliographic information can optionally be associated with the work as a whole, and with each performance, score, or analysis.

NOTE 54 Information architects may wish to include themes elements in the content model of their bibinfo-form elements in actual DTDs.

```
              <!-- Bibliographic Data -->
<!element bibinfo
                -- Bibliographic data --
                - O ANY             -- User must define own
                                       structure. --
>
<!attlist bibinfo
    SMDL        NAME        bibinfo
>
```

10.1 Themes (themes)

The architectural form **themes** (themes) contains links to the cantus that pinpoint key passages (and/or or famous passages) for the purpose of identification of the work.

NOTE 55 A `themes` element allows a cataloging application, for instance, to locate and then display or perform a well known passage, thus making it easy for the user to verify that the correct piece has been accessed.

```
                       <!-- Themes -->
   <!element themes    -- Themes that best identify the work (e.g., incipit)
--
                       - O ( ilink)*
   >
   <!attlist themes
       SMDL        NAME        themes
   >
```

11 Invoking the SMDL Architecture in a Document

11.1 Declaration that the Structure of the Document is a Derived from a Base Architecture

An SMDL document must indicate the fact that its architecture is derived from a base architecture by means of the `APPINFO` parameter of its SGML declaration. The keyword `ArcBase` is specified as a sub-parameter of the `APPINFO` parameter of the SGML declaration. It might appear in the SGML declaration of the document as follows:

```
APPINFO ArcBase
```

NOTE 56 There may be other sub-parameters, in addition to “`ArcBase`.”

This indicates to the document processing system that there may be “`ArcBase`” processing instructions in the document that will specify the base architectures that are in effect.

NOTE 57 In effect, this is an invocation of the “Base Architecture” identification facility found in Annex C of the HyTime International Standard. It does not invoke the HyTime architecture.

11.2 Declaration that the Document’s Architecture is Derived from SMDL

An SMDL document must indicate the fact that its architecture is derived from the SMDL base architecture by means of an `ArcBase` processing instruction. In SGML’s reference concrete syntax, it would appear as follows:

```
<?ArcBase SMDL>
```

This processing instruction establishes that SMDL is the document’s local name for one of its base architectures, and that after this processing instruction, the document processing system can expect to find:

a) An SGML `NOTATION` declaration that identifies the architecture definition document (in this case, this International Standard). Associated with this `NOTATION` declaration must be an attribute definition list (an `ATTLIST`) that declares the support attributes of the architecture. (For SMDL, these are provided below as they would appear in SGML’s reference concrete syntax.)
b) an SGML general `ENTITY` declaration that identifies the meta-DTD of the base architecture (SMDL), and which may specify values for SMDL’s architectural support attributes. (An example using SGML’s reference concrete syntax is given below.)

11.2.1 Declaration of SMDL as a Notation

The following SGML NOTATION declaration points to the this International Standard. The ATTLIST that follows it defines the attribute list of the SMDL notation. Both of these declarations (or their equivalents) should appear in the prolog of every SMDL document.

```
<!NOTATION              SMDL    PUBLIC
    "ISO/IEC 10743:1996//NOTATION
     SMDL Architecture Definition Document//EN"
        -- A document architecture conforming to the
           Architectural Form Definition Requirements of
           International Standard ISO/IEC 10743. --
>

<!ATTLIST   #NOTATION   SMDL
    ArcFormA    -- name of attribute that identifies the architectural
                   form to which an SMDL element conforms. --
                NAME        #FIXED      "SMDL"
    ArcNamrA    -- name of attribute that allows aliasing of attribute
                   names defined by SMDL. --
                NAME        #FIXED      "SMDLNms"
    ArcBridA    -- name of attribute that specifies bridging between
                   SMDL elements and non-SMDL elements. --
                NAME        #FIXED      "SMDLBrid"
    ArcDocF     -- name of architectural form that SMDL requires as
                   the document element. --
                NAME        #FIXED      work
    ArcVer      -- version of SMDL to which this document conforms --
                CDATA       #FIXED      1996
    ArcSGML     -- entity containing the SGML declaration applicable
                   to the SMDL meta-DTD --
                ENTITY      #IMPLIED            -- Default: SGML
                                                   declaration is in
                                                   the meta-DTD
                                                   entity.  --
>
```

NOTE 58 The above declarations are set forth in this International Standard for use in SMDL documents, and they may appear in such documents *verbatim*. However, in the prolog of any particular SMDL document, these declarations could conceivably be governed by an SGML declaration that defines a concrete syntax other than SGML's Reference Concrete Syntax. In such a case, they might not look exactly like the above declarations, but they would have the same effect.

11.2.2 Declaration Template for the SMDL Meta-DTD Entity

The following SGML ENTITY declaration points to a file (here named "mysmdl.mtd") containing those formalized SGML aspects of this International Standard (i.e., those portions of the SMDL meta-DTD), which are actually in effect for this application. An ENTITY declaration having the same effect as this one must appear in the prolog of all SMDL documents.

```
<!ENTITY    SMDL
            -- meta-DTD for SMDL architecture --
        SYSTEM "mysmdl.mtd" CDATA SMDL [ArcSGML=mysmdl.sgmldecl]
>
```

NOTE 59 The above `ENTITY` declaration illustrates a scenario in which the SGML declaration governing the meta-DTD is in a separate file named `mysmdl.sgmldecl`.

12 Conformance

12.1 Conforming SMDL document

If an SMDL document

a) complies with all provision of this International Standard; and
b) it is a conforming SGML document as defined in ISO 8879; and
c) it is a conforming HyTime document as defined in ISO/IEC 10744,

it is a conforming SMDL document.

NOTE 60 The provisions of this International Standard allow wide latitude in choosing which constructs SMDL to support.

NOTE 61 Conformance to SMDL is independent of whether the document also conforms to a document architecture.

12.2 Conforming SMDL application

12.2.1 Application conventions

A conforming SMDL application's conventions can affect only areas that are left open by this International Standard to specification by applications.

NOTE 62 E.g., the generic identifiers of element types conforming to SMDL and SMDL architectural forms may be chosen *ad libitum*, but the values of `SMDL` attributes can only be changed if the `SMDLNms` attribute is used as specified in this International Standard.

12.2.2 Conformance of documents

A conforming SMDL application shall require its documents to be conforming SMDL documents, and shall not prohibit any markup that this International Standard would allow in such documents.

NOTE 63 For example, an application markup convention could recommend that only certain minimization functions be used, but could not prohibit the use of other functions if they are allowed by the formal specification.

12.2.3 Conformance of documentation

A conforming SMDL application's documentation shall meet the requirements of this International Standard (see Section 12.5).

12.3 Conforming SMDL system

If an SMDL system

a) meets the requirements of this sub-clause; and
b) it is a conforming SGML system as defined in ISO 8879; and

c) it is a conforming HyTime system as defined in ISO/IEC 10744;

it is a conforming SMDL system.

NOTE 64 Whether a system is a conforming SMDL system is not affected by whether it is also a conforming implementation of a document architecture.

12.3.1 Conformance of documentation

A conforming SMDL system's documentation shall meet the requirements of this International Standard (see Section 12.5).

12.3.2 Conformance to SMDL system declaration

A conforming SMDL system shall be capable of processing any conforming SMDL document that is not inconsistent with the system's SMDL system declaration (see Section 12.6).

NOTE 65 A system's inability to process data content notations that are not defined in this International Standard does not affect whether it is a conforming SMDL system.

12.3.3 Application conventions

A conforming SMDL system shall not enforce application conventions as though they were requirements of this International Standard.

NOTE 66 Warning of the violations of application conventions can be given, but they must be distinguished from reports of SMDL errors.

12.4 Validating SMDL engine

If an SMDL engine in a conforming SMDL system meets the requirements of this sub-clause, it is a validating SMDL engine.

NOTE 67 A conforming SMDL system need not have a validating SMDL engine. Implementers can therefore decide whether to incur the overhead of validation in a given system. A user whose multimedia authoring system allows the validation and correction of SMDL documents, for example, will not need to repeat the validation process when the documents are rendered by a processing system.

12.4.1 Error recognition

A validating SMDL engine shall find and report a reportable SMDL error if one exists, and shall not report an error when none exists.

A validating SMDL may optionally report other errors.

NOTE 68 This International Standard does not specify how an SMDL error shold be handled, beyond the requirement for reporting it. In particular, it does not state whether the erroneous information should be treated as data, and/or whether an attempt should be made to continue processing after an error is found.

NOTE 69 This International Standard does not prohibit a validating SMDL engine from reporting an error that this International Standard considers non-reportable if the engine is capable of doing so. Neither does it prohibit an engine from recovering from such errors, not does it require an engine to report them when it is not performing validation.

A validating SMDL engine may warn of conditions that are potentially, but not necessarily, errors.

NOTE 70 SMDL architectural forms do not constrain the construction of document type definitions, only document instances. However, a validating SMDL engine can optionally report DTD constructs that would prevent the creation of a valid conforming instance, or that would allow the creation of a nonconforming instance.

12.4.2 Identification of SMDL messages

Reports of SMDL errors, including optional reports, shall be identified as SMDL messages in such a manner as to distinguish them clearly from all other messages, including warnings of potential SMDL errors.

12.4.3 Content of SMDL messages

A report of an SMDL error, including an optional report, shall state the nature and location of the error in sufficient detail to permit its correction.

NOTE 71 This requirement is worded to allow implementers maximum flexibility to meet their user and system requirements.

12.5 Documentation requirements

The objectives of this International Standard will be met most effectively if users at all levels are aware that SMDL documents conform to an International Standard that is indepenedent of any application or engine. The documentation of a conforming SMDL system or application shall further such awareness.

NOTE 72 These requirements are intended to help users apply knowledge gained from one SMDL system to the use of other systems. They are not intended to inhibit a casual and friendly writing style.

12.5.1 Standard identification

Standard identification shall be in the natural language of the documentation.

Standard identification text shall be displayed prominently:

a) in a prominent location in the front matter of all publications (normally the title page and cover page);
b) on all identifying display screens of SMDL programs; and
c) in all promotional and training materials.

For applications, the identification is:

```
An SMDL application conforming to
International Standard ISO/IEC 10743 --
Standard Music Description Language
```

For systems, the identification is:

```
An SMDL system conforming to
International Standard ISO/IEC 10743 --
Standard Music Description Language
```

The documentation for a conforming SMDL system shall include an SMDL system declaration (see Section 12.6).

12.5.2 Identification of SMDL constructs

The documentation shall distinguish SMDL constructs from application conventions and system functions, and shall identify the SMDL constructs as being part of the Standard Music Description Language.

NOTE 73 The objective of this requirement is for the user to learn which constructs are common to all SMDL systems, and which are unique to this one. This will reduce the experienced user's learning time for a new system or application.

This International Standard shall be cited as a reference for supported SMDL constructs that are not specifically documented for the system or application. For example, if, for simplicity's sake, only a subset of some function is presented (such as by omitting some of the options of the extent reconciliation strategy element type form), it shall be stated clearly that other options exist and can be found in this International Standard.

12.5.3 Terminology

All SMDL constructs shall be introduced using the terminology of this International Standard, translated to the national language used by the publication or program.

Such standard terminology should be used thoughout the documentation. If notwithstanding, a non-standard equivalent is used for a standard term, it must be introduced in context and it shall not conflict with any standard SMDL terms, including terms for unsupported or undocumented constructs.

12.6 SMDL system declaration

An SMDL system declaration specifies the version of SMDL and all of the facilities that an SMDL system can support.

NOTE 74 The system declarataion for a system that supports all of the modules and optional facilities of SMDL is given in Section 11.2.1.