XML & MusicXML

craig@ccrma.stanford.edu
24 February 2015

XML Development

- eXtensible Markup Language

Version 0 :: 1996
Version 1.0 :: 1998
Version 1.1 :: 2004
Version 1.1.5 :: 2008

XHTML 1.0 2000
1.1 2001
XHTML5 2008

http://en.wikipedia.org/wiki/XML

- Predecessor: SGML (Standardized Generalized Markup Language)

1970's – 1980's

HTML 1.0 1991
2.0 1995
4.0 1997
5.0 2008

http://en.wikipedia.org/wiki/Standard_Generalized_Markup_Language

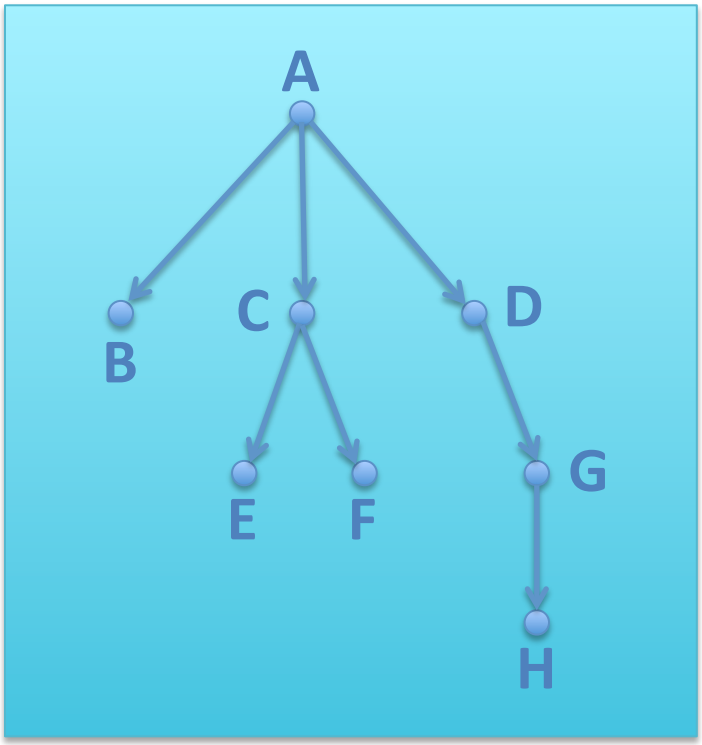
- Predecessor: GML (Generalize Markup Language)

1960's

http://en.wikipedia.org/wiki/IBM_Generalized_Markup_Language

XML data structure

- XML describes a tree structure:



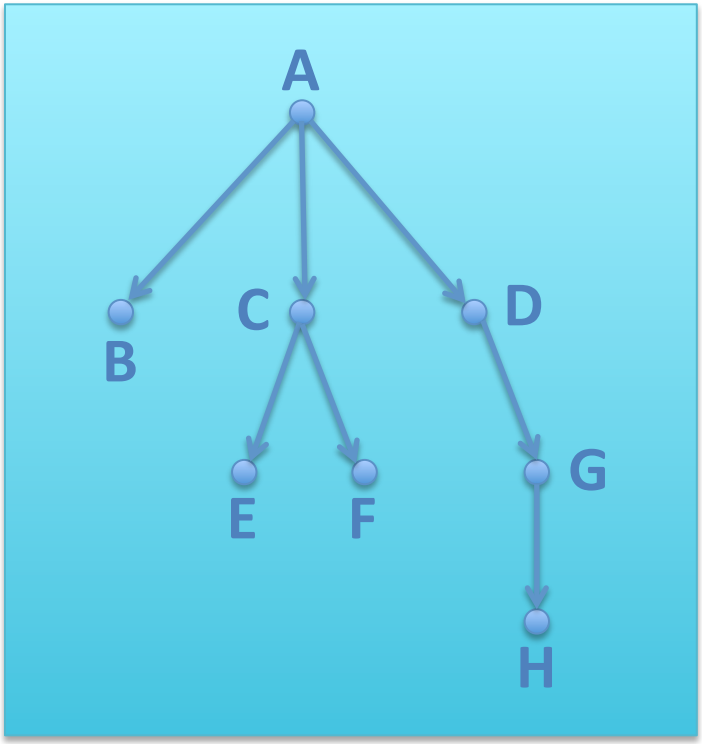
- Serialization:

```
<A>
   <B/>
   <C>
      <E/>
      <F/>
   </C>
   <D>
      <G>
         <H/>
      </G>
   </D>
</A>
```

- Equivalent serialization: <A><B/><C><E/><F/></C><D><G><H/></G></D></A>

XML data structure

- XML describes a tree structure:



- Same data structure as directories/folders on a hard disk

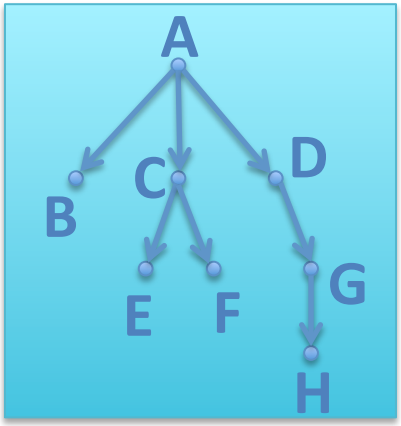
- Same conceptualization as LISP code:

(A B (C E F) (D (G (H))))

- SharpEye's internal format is a tree structure (but not XML)

- JSON data format is also a tree structure. (with a simpler syntax than XML).

XML Terminology




```
<A>
   <B/>
   <C>
      <E/>
      <F/>
   </C>
   <D>
      <G>
         <H/>
      </G>
   </D>
</A>
```

- <C>…</C> is an ***element*** (tree node)
- C is the element's ***name***
- <C> is a *start* ***tag***
- </C> is an end tag
- <E/> and <F/> are *element* ***content*** of <C>
- Plain text inside of an element is *text* ***content***

- <H/> is an element without contents (terminal node)
- <H/> is equivalent to <H></H>
- Start tags must be followed by matching end tag, or the shorthand <xxx/> must be used.

Element Attributes

- Elements can contain a list of attributes within the start tag

<A a="1" b="two" c="1 and 2">

- Element **A** has three *attributes*: a, b, and c.
- A is the *name* of the attribute, 1 is its value.
- Attributes must have values. c="" represents an attribute without a value.
- Attributes are optional (similar to key values in LISP).
- The value of a is 1, the value of b is two and the value of c is 1 and 2.
- XML Attribute values *must* be enclosed in double or single quotes.
- Only one attribute of a given name allowed. Bad example: <A a="1" a="2">
- Attributes are considered unordered:

  <A a="1" b="two"> is identical to <A b="two" a="1">

HTML attributes do not need to be enclosed in quotes:
<table cellpadding=10> is equivalent to <table cellpadding="10">
XHTML is does not allow the first case since quotes are always needed.

Elements vs. Attributes

- Elements can contain subelements
- Attributes cannot contain subattributes

- Two similar (but not identical) ways of expressing the same data:

```
<A a="1" b="two" c="1 and 2"/>
```

```
<A>
  <a>1</a>
  <b>two</b>
  <c>1 and 2</c>
</A>
```

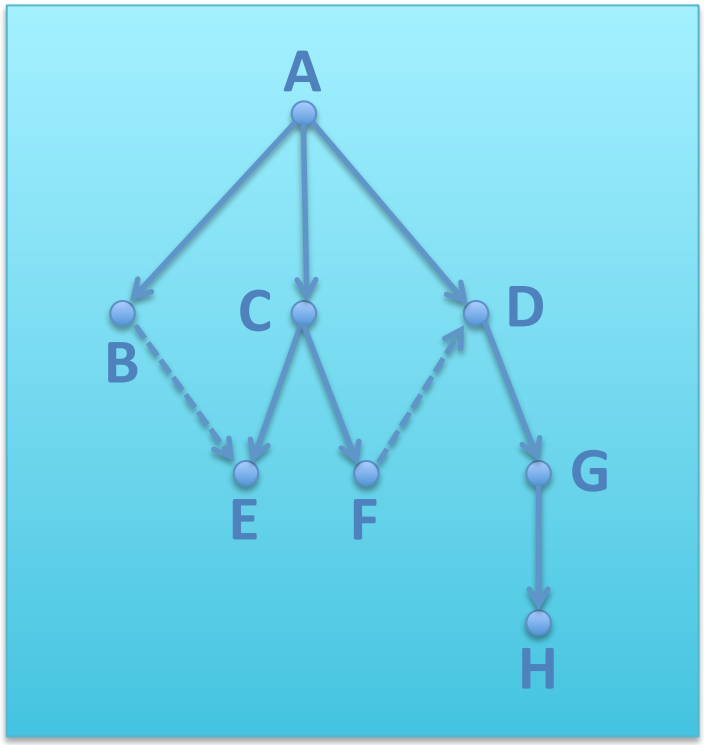
Informal shorthand for attribute a of element A (but not in data):

A@a

- Attribute a in the first example cannot be expanded later into subattributes
- Element a in the second example can be expanded later to include element contents

XML for non-tree structured data

- non-tree data can be shoe-horned into XML data structure



- Tree-like portions encoded as XML elements
- Non-tree connections handled by specialized id/idref/idrefs attributes.

```
<A>
   <B idref="e"/>
   <C>
      <E id="e"/>
      <F idref="d"/>
   </C>
   <D id="d">
      <G>
         <H/>
      </G>
   </D>
</A>
```

DTD:

```
<!ATTLIST B
   id      ID      #IMPLIED
   idref   IDREF   #IMPLIED>
```

- Similar to pointers in C.

XML declaration

- Used to indicate that the following data is XML data
- First characters in file must be “<?xml” (see UTF-16 below).

<?xml version=“1.0” encoding=“UTF-8” standalone=“yes”?>

Three attributes which *must* be in this order (but optional):

@version = version of XML being used (1.0 or 1.1).
@encoding = character set being used in data. (also UTF-16 which requires two endian bytes before opening <?)
* UTF-8 is backwards compatible with 7-bit ASCII
* UTF-16 is not.
@standalone = “yes” if no external definition file, “no” if DTD (Document Type Definition).

XML complete data file

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<A>
    <B idref="e"/>
    <C>
        <E id="e"/>
        <F idref="d"/>
    </C>
    <D id="d">
        <G>
            <H/>
        </G>
    </D>
</A>
```

Even more complete data file

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<!DOCTYPE A [
  <!ELEMENT A (B,C,D)>
  <!ELEMENT C (E,F)>
  <!ELEMENT D (G)>
   <!ELEMENT G (H)>
   <!ATTLIST B idref IDREF #IMPLIED>
   <!ATTLIST E id    ID    #IMPLIED>
   <!ATTLIST D id    ID    #IMPLIED>
]>
<A>
    <B idref="e"/>
    <C>
        <E id="e"/>
        <F idref="d"/>
    </C>
    <D id="d">
        <G>
            <H/>
        </G>
    </D>
</A>
```

Element A can have subelements B, C & D.

Element B can have an attribute named idref which can be set to a value which is the type IDREF.

Data/Structure definition separation

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<!DOCTYPE A SYSTEM "tree.dtd">
or <!DOCTYPE A SYSTEM "http://somewhere.com/tree.dtd">
or <!DOCTYPE A PUBLIC "-//Owner/Class Description//Language//Version" "tree.dtd">
<A>
    <B idref="e"/>
    <C>
        <E id="e"/>
        <F idref="d"/>
    </C>
    <D id="d">
        <G>
            <H/>
        </G>
    </D>
</A>
```

Formal Public Identifier

tree.dtd:

```
<!ELEMENT A (B,C,D)>
<!ELEMENT C (E,F)>
<!ELEMENT D (G)>
<!ELEMENT G (H)>
<!ATTLIST B idref IDREF #IMPLIED>
<!ATTLIST E id    ID    #IMPLIED>
<!ATTLIST D id    ID    #IMPLIED>
```

Parameters

Fixed — function(int one, int two, int three)
(like MIDI)

C

Optional — function(int one, int two = 2, int three=3)
(like Guido, SCORE)

C++

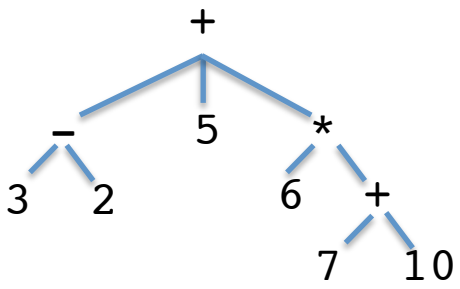
Variable — function(const char* format, …)
http://cc.byexamples.com/2007/01/18/va-list-create-function-like-printf-2

Key — (function :key1 value1 :key2 value2)

Lisp

Tree — (+ (- 3 2) 5 (* 6 (+ 7 10)))

(recursive key system)



MIDI Parameters

- All MIDI protocol parameters are fixed excepts for “system exclusive messages”
- Meta messages (component of MIDI files, not MIDI protocol) are variable.

0x90 60 127 note(channel, key, velocity)

0xE6 0x7f 0x7f bend(channel, LSB, MSB)

- Allows hot-plugging of MIDI cable.
- Limits expandability (function space maximized with fixed parameter commands)

SCORE Parameters

- SCORE items are all variable length fixed parameter lists.
- Similar to MIDI meta message system, but better extensibility
- Identical to Music V (C Sound) parameter system

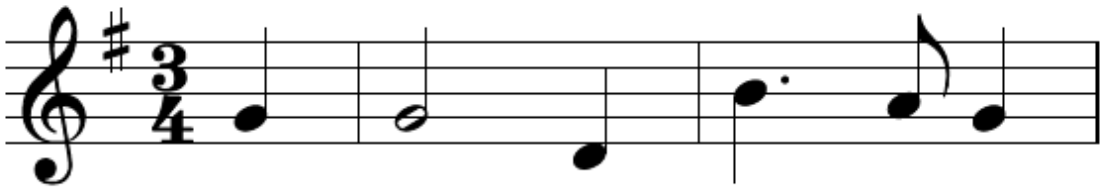
http://www.csounds.com/chapter1/index.html

```
8  1 0 0 0.6 128.146
14 1 0 3
3  1 1.2 0 0.8
17 1 5.997 0 -1
1  1 9.297 7 20 1 2
1  1 20.566 4 10 2 4
1  1 50.64 8 20 1 2
5  1 50.64 8.5 8.5 64.016 1.579 -2
14 1 61.923 1
1  1 64.016 8 20 1 2
1  1 75.291 6 10 1 2
1  1 86.561 5 10 2 4
14 1 109.113 1
1  1 111.206 9 20 2 4
14 1 128.146 1 3
```

- Allows for both forwards and backwards compatibility:
  - New parameters added to end of current list
  - Old program ignores (but preserves) unknown parameters.

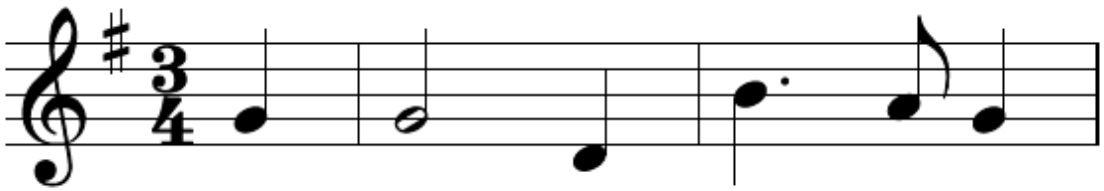
Non-XML data trees

- SharpEye uses a form of tree structure for its data
- LISP-based ENP music editor uses tree structure:



```
(:begin :score
      (:begin :part1
            :staff :treble-staff
            :key-signature :g-major
            :time-signature (3 4)
            (:begin :voice1
                  (
                        :time-signature (3 4 :kind :pickup)
                        (1 ((1 :notes (67))))
                  )
                  (:begin :measure1
                        (2 ((1 :notes (67))))
                        (1 ((1 :notes (74))))
                  )
                  (:begin :measure2
                        (2 (
                              (3 :notes (71))
                              (1 :notes (69))
                        ))
                        (1 ((1 :notes (67))))
                  )
                  (:begin :measure3
                        (2 (
                              (3 :notes (67))
                              (1 :notes (69))
                        ))
                        (1 ((1 :notes (71))))
                  )
```

XML as a container for non-tree data



```
8   1    0.000  0  0 100
3   1    1.500
17 1   9.444   0  1
18 1  13.444  0  3   4
1   1  20.944  5 10   0 1.0
14 1  32.290  1
1   1  35.679  5 10   1 2.0
1   1  52.032  2 10   0 1.0
14 1  63.378  1
1   1  66.767  7 20   0 1.5 0 10
1   1  80.853  6 10   0 0.5 0  1
1   1  88.654  5 10   0 1.0
14 1 100.000 1
```

```
<SCORE version="4">
  <item p1="8" p2="1" p6="100" />
  <item p1="3" p2="1" p3="1.5" />
  <item p1="17" p2="1" p3="9.444" p5="1" />
  <item p1="18" p2="1" p3="13.444" p5="3" p6="4" />
  <item p1="1" p2="1" p3="20.944" p4="5" p5="10" p7="1" />
  <item p1="14" p2="1" p3="32.29" p4="1" />
  <item p1="1" p2="1" p3="35.679" p4="5" p5="10" p6="1" p7="2" />
  <item p1="1" p2="1" p3="52.032" p4="2" p5="10" p7="1" />
  <item p1="14" p2="1" p3="63.378" p4="1" />
  <item p1="1" p2="1" p3="66.767" p4="7" p5="20" p7="1.5" p9="10" />
  <item p1="1" p2="1" p3="80.853" p4="6" p5="10" p7="0.5" p9="1" />
  <item p1="1" p2="1" p3="88.654" p4="5" p5="10" p7="1" />
  <item p1="14" p2="1" p3="100" p4="1" />
</SCORE>
```

XML

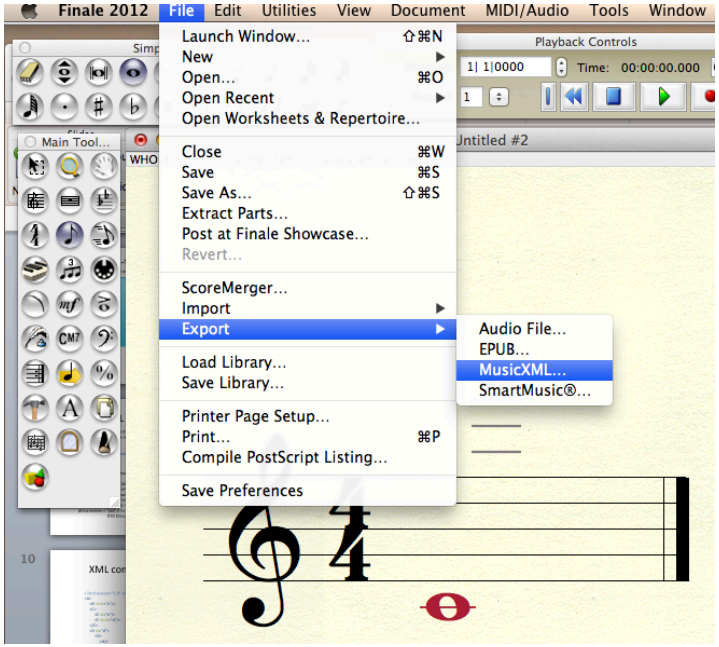
- Advantage: Simple parsing model for data storage
  - Like MIDI, SCORE, LISP, Humdrum
  - Unlike Guido, Lilypond, C, C++, Java, JavaScript (lex/bison type formats)

- Allows for hierarchical structuring of data
  - **Good**: music notation usually fits well into hierarchical model
    - Useful for manipulating music
  - **Bad**: music notation is 2-dimensional, XML is 1-dimensional (superposition of multiple hierarchies)

- Allows for forwards compatibility, and backwards compatibility if careful
  - Possible to add new parameters without altering parsing

MusicXML

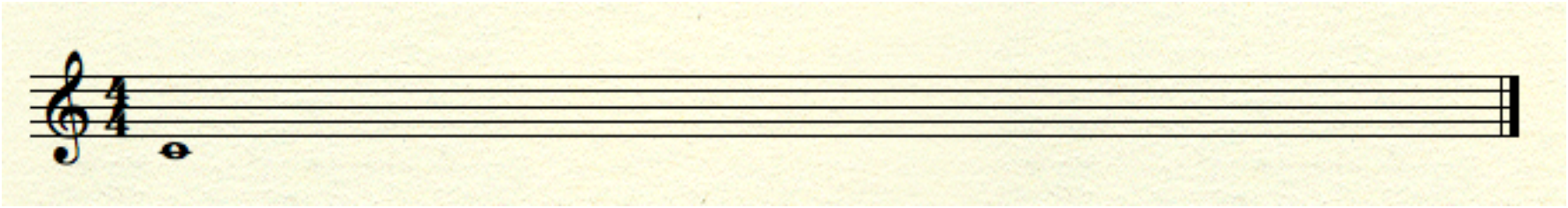
```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<!DOCTYPE score-partwise PUBLIC "-//Recordare//DTD MusicXML 1.0 Partwise//EN"
                "http://www.musicxml.org/dtds/1.0/partwise.dtd">
<score-partwise>
  <identification>
    <encoding>
      <software>Finale 2012 for Mac</software>
      <software>Dolet Light for Finale 2012</software>
      <encoding-date>2013-01-21</encoding-date>
    </encoding>
  </identification>
  <part-list>
    <score-part id="P1">
      <part-name>MusicXML Part</part-name>
      <score-instrument id="P1-I1">
        <instrument-name>Garritan: ARIA Player</instrument-name>
      </score-instrument>
      <midi-instrument id="P1-I1">
        <midi-channel>1</midi-channel>
        <midi-bank>15489</midi-bank>
        <midi-program>1</midi-program>
      </midi-instrument>
    </score-part>
  </part-list>
  <!--======================================================-->
```



<!-- ... --> is a comment in XML

visual barline for readability

MusicXML (2)



```
<part id="P1">
  <measure number="1">
   <print/>
   <attributes>
    <divisions>2</divisions>
    <key>
     <fifths>0</fifths>
     <mode>major</mode>
    </key>
    <time>
     <beats>4</beats>
     <beat-type>4</beat-type>
    </time>
    <clef>
     <sign>G</sign>
     <line>2</line>
    </clef>
   </attributes>
   <sound tempo="120"/>
```

divisions per quarter note

```
    <note default-x="86">
      <pitch>
       <step>C</step>
       <octave>4</octave>
      </pitch>
      <duration>8</duration>
      <voice>1</voice>
      <type>whole</type>
     </note>
     <barline location="right">
      <bar-style>light-heavy</bar-style>
     </barline>
    </measure>
  </part>
  <!--=========================================================-->
</score-partwise>
```

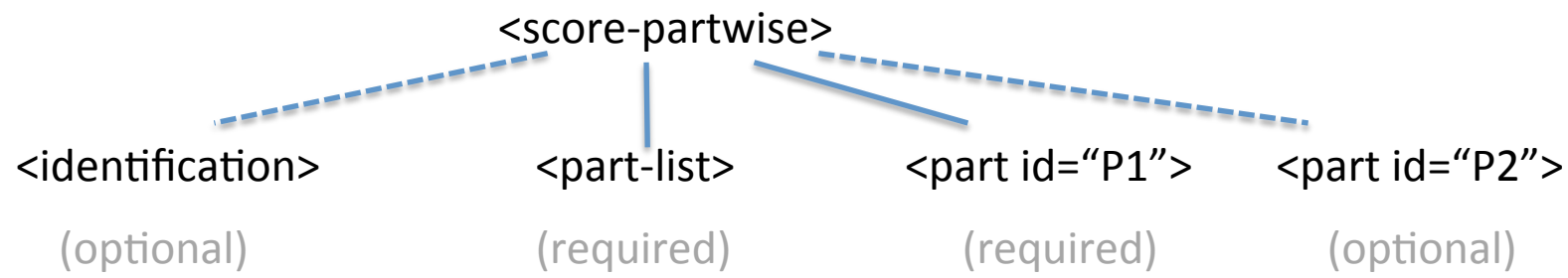
4 quarter notes

looks like a whole note

Compare to GUIDO:
[c/1]

(GUIDO content not separable from structure)

MusicXML Data hierarchy (root)




```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<!DOCTYPE score-partwise PUBLIC "-//Recordare//DTD MusicXML 1.0 Partwise//EN"
                                "http://www.musicxml.org/dtds/1.0/partwise.dtd">
<score-partwise>
```

<score-partwise> is the *root element*

```
<!ELEMENT score-partwise (%score-header;, part+)>

<!ENTITY % score-header
    "(work?, movement-number?, movement-title?,
     identification?, defaults?, credit*, part-list)">
```

DTD/Schema

DTD

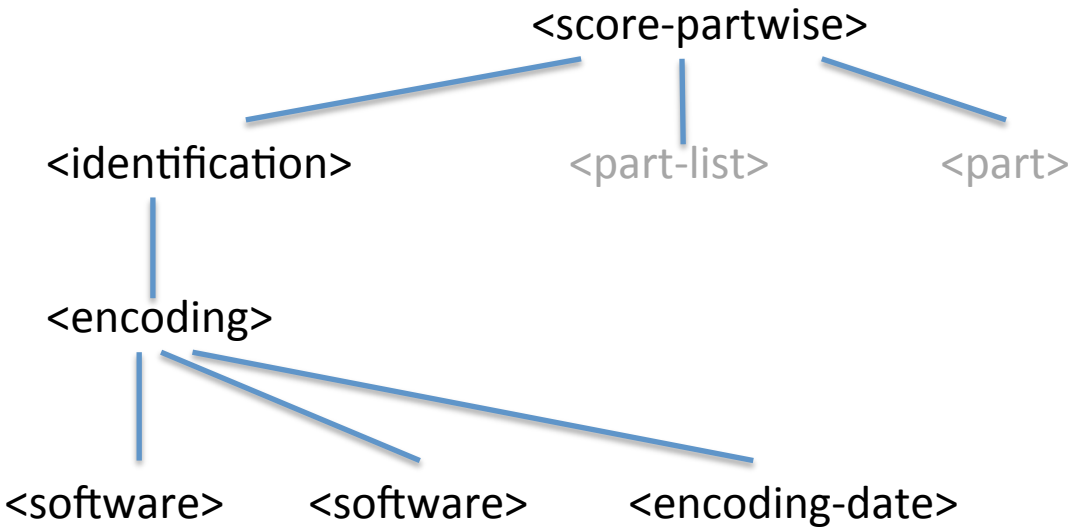
```
<!ELEMENT score-partwise (%score-header;, part+)>

<!ENTITY % score-header
    "(work?, movement-number?, movement-title?,
     identification?, defaults?, credit*, part-list)">
```

Schema

```
<xs:element name="score-partwise" block="extension substitution" final="#all">
 <xs:annotation>
  <xs:documentation>
  The score-partwise element is the root element for a partwise
  MusicXML score. It includes a score-header group followed by a
  series of parts with measures inside. The document-attributes
  attribute group includes the version attribute.
  </xs:documentation>
 </xs:annotation>
 <xs:complexType>
  <xs:sequence>
   <xs:group ref="score-header"/>
   <xs:element name="part" maxOccurs="unbounded">
    <xs:complexType>
     <xs:sequence>
      <xs:element name="measure" maxOccurs="unbounded">
       <xs:complexType>
        <xs:group ref="music-data"/>
        <xs:attributeGroup ref="measure-attributes"/>
       </xs:complexType>
      </xs:element>
     </xs:sequence>
     <xs:attributeGroup ref="part-attributes"/>
    </xs:complexType>
   </xs:element>
  </xs:sequence>
  <xs:attributeGroup ref="document-attributes"/>
 </xs:complexType>
</xs:element>
```

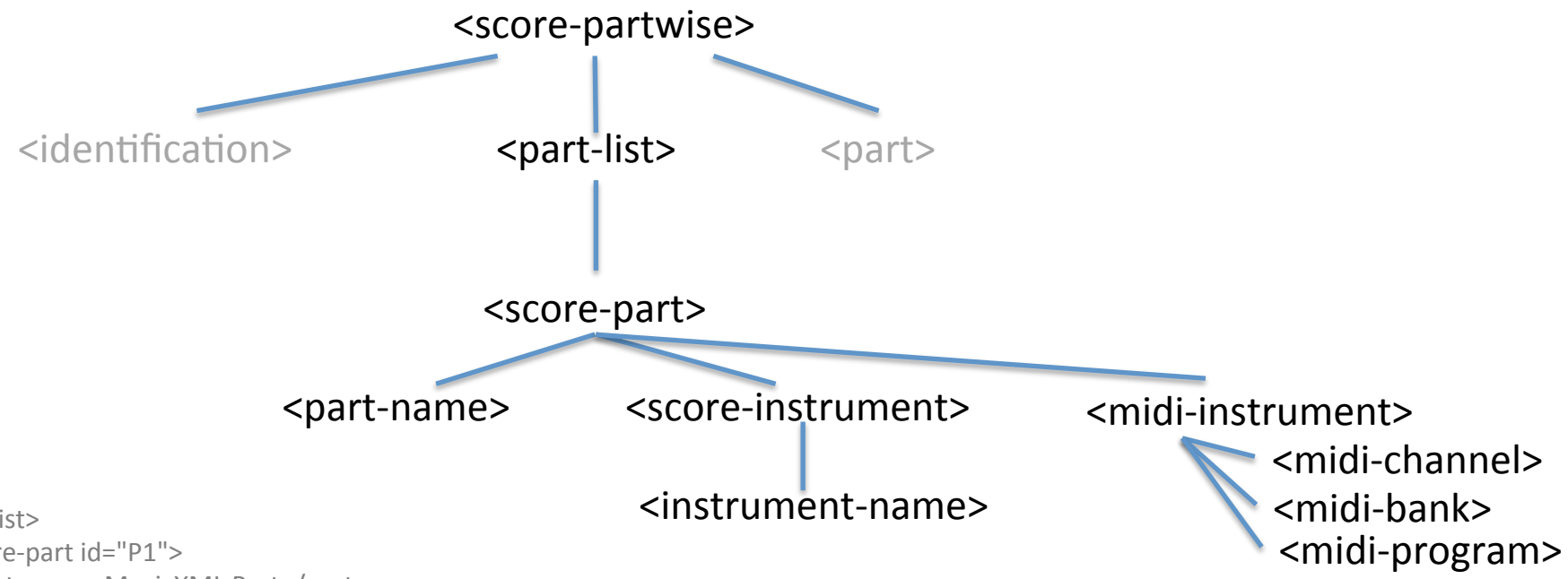
http://www.musicxml.com/UserManuals/MusicXML/MusicXML.htm

MusicXML Data hierarchy (header)



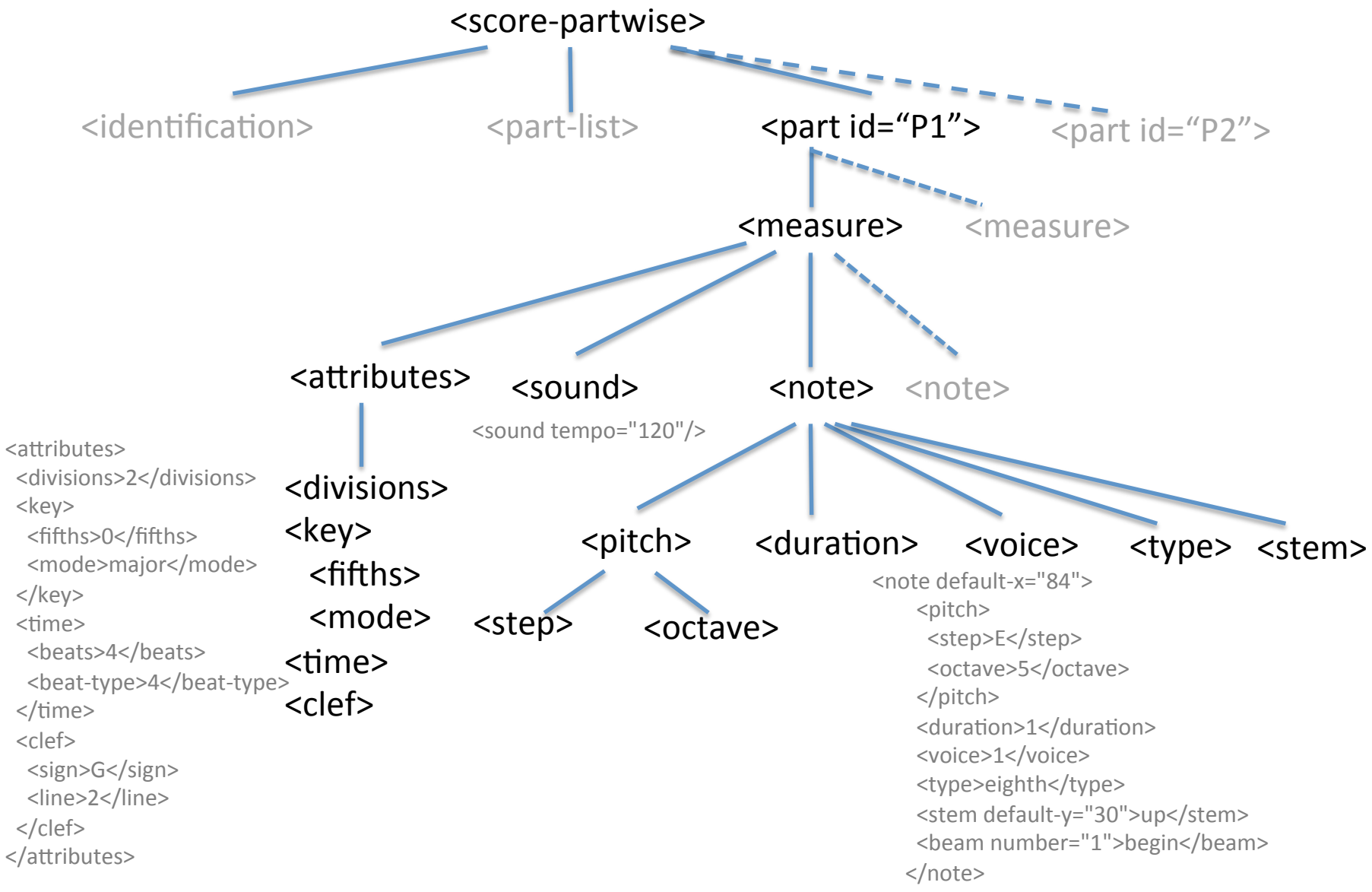

```
<identification>
  <encoding>
   <software>Finale 2012 for Mac</software>
   <software>Dolet Light for Finale 2012</software>
   <encoding-date>2013-02-25</encoding-date>
  </encoding>
 </identification>
```

MusicXML Data hierarchy (header 2)




```
<part-list>
  <score-part id="P1">
   <part-name>MusicXML Part</part-name>
   <score-instrument id="P1-I1">
    <instrument-name>Garritan: ARIA Player</instrument-name>
   </score-instrument>
   <midi-instrument id="P1-I1">
    <midi-channel>1</midi-channel>
    <midi-bank>15489</midi-bank>
    <midi-program>1</midi-program>
   </midi-instrument>
  </score-part>
 </part-list>
```

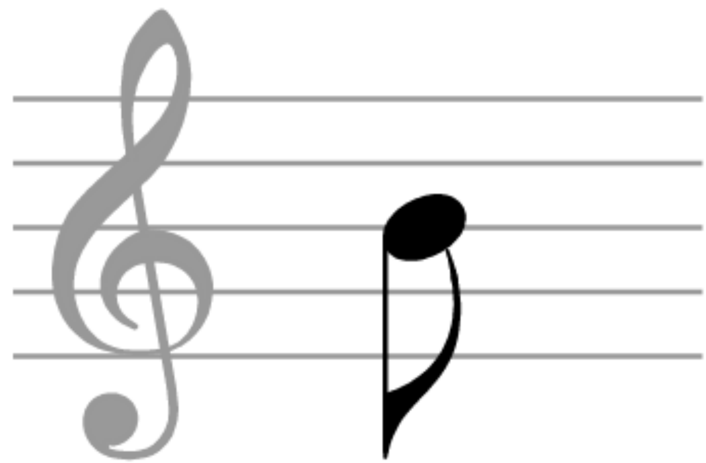
MusicXML Data hierarchy (part)



MusicXML <note>

http://www.musicxml.com/UserManuals/MusicXML/MusicXML.htm#EL-MusicXML-note.htm

```
<note>
    <pitch>
     <step>B</step>
     <octave>4</octave>
    </pitch>
    <duration>16</duration>
    <voice>1</voice>
    <type>eighth</type>
    <stem default-y="-50">down</stem>
 </note>
```



```
                  1         2         3
Column: 123456789012345678901234567890
```

MuseData “<note>”:

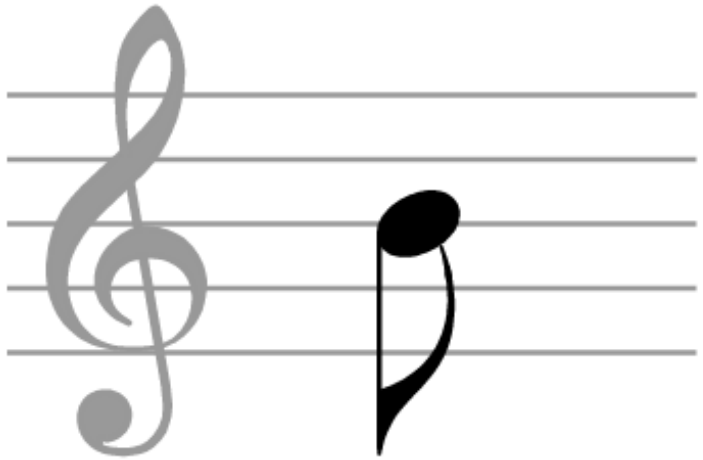
```
B4      1      1 e      d
```

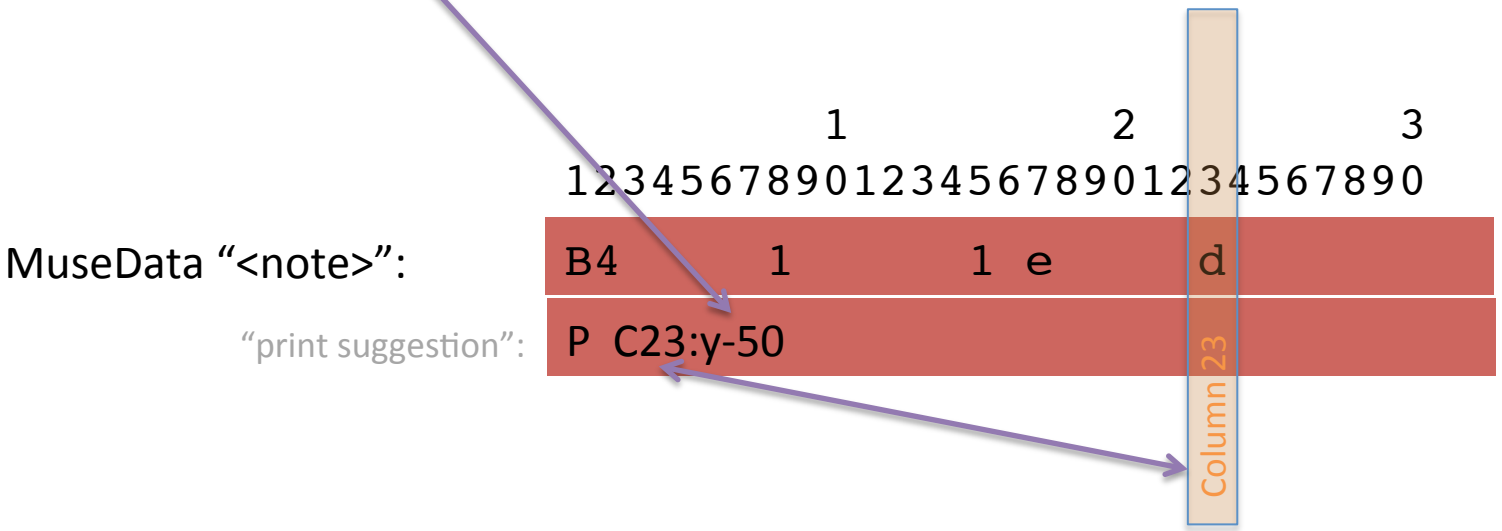
<step> <octave> (<pitch>) — <duration> — <voice> — <type> — <stem>

MusicXML <note>

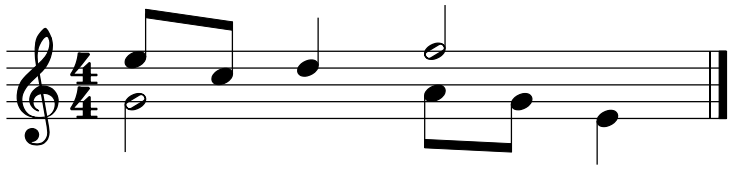
http://www.musicxml.com/UserManuals/MusicXML/MusicXML.htm#EL-MusicXML-note.htm

```
<note>
      <pitch>
        <step>B</step>
        <octave>4</octave>
      </pitch>
      <duration>16</duration>
      <voice>1</voice>
      <type>eighth</type>
      <stem default-y="-50">down</stem>
  </note>
```





MusicXML Voices/Layers



MusicXML

```
<measure>
  <attributes>
  <note> E5, 1 tick
  <note> C5, 1 tick
  <note> D5, 2 ticks
  <note> F5, 4 ticks
  <backup> 8 ticks
  <note> G4, 4 ticks
  <note> A4, 1 tick
  <note> G4, 1 tick
  <note> E4, 2 tick
</measure>
```

Voice 1:

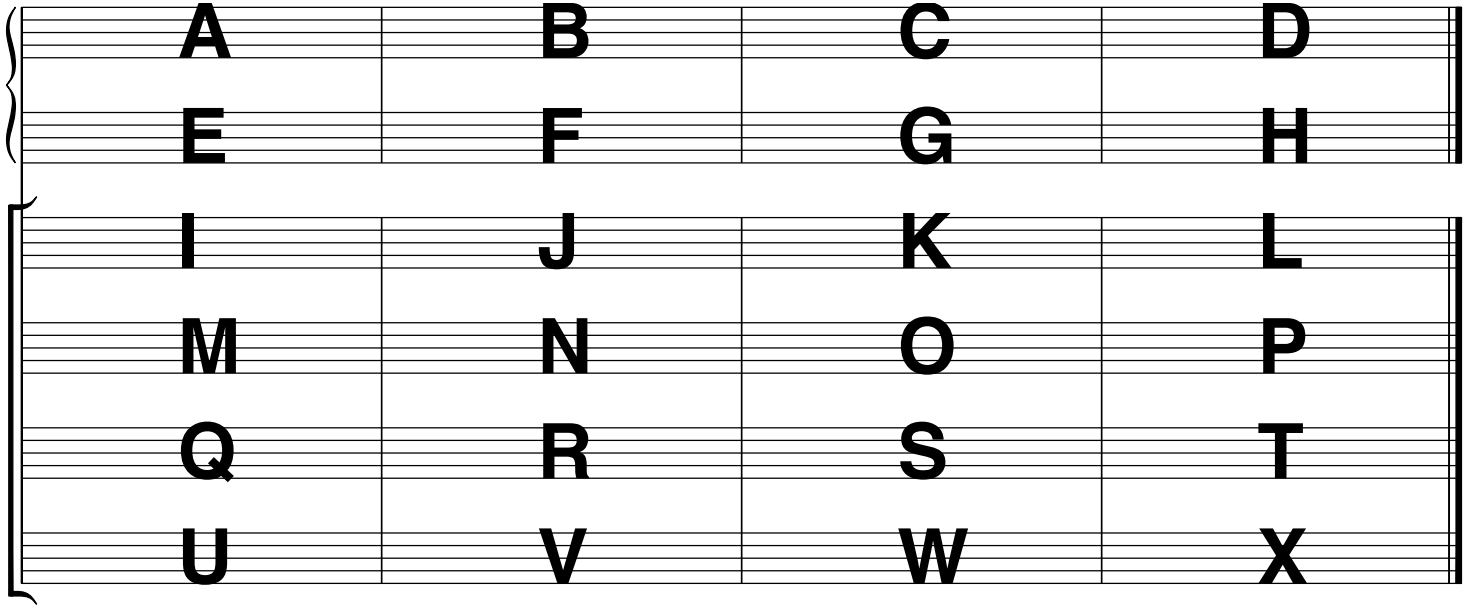
Voice 2:

MuseData

```
$    Q:2 K:0 T:1/1 C:4
E5       1       1 e       u   [
C5       1       1 e       u   ]
D5       2       1 q       u
F5       4       1 h       u
back     8
G4       4       2 h       d
A4       1       2 e       d   [
G4       1       2 e       d   ]
E4       2       2 q       d
mheavy2
```

Partwise/timewise

- <score-partwise> stores score one part (staff) sequentially (part->measure)
- <score-timewise> score each measure sequentially for all parts (measure->part)
- <score-timewise> is about as common as MIDI Type-2 files.
- <score-timewise> is a quasi-realtime encoding (not strictly real-time).



- <score-partwise>: ABCD, EFGH, IJKL, MNOP, QRST, UVWX
- <score-timewise>: AEIMQU, BFJNRV, CGKOSW, DHLPTX

- <opus>: multiple movements of (partwise or timewise).

MusicXML versions

http://www.musicxml.com
http://en.wikipedia.org/wiki/MusicXML

MusicXML 1.0	2004
MusicXML 1.1	2005
MusicXML 2.0	2007
MusicXML 3.0	2011

MusicXML 3.0

- Compressed MusicXML: (.mxl): ZIP file which can include linked material as well as main XML file.

- Standardized list of instruments
  - http://www.musicxml.com/dtds/3.0/sounds.xml
  - http://www.humdrum.org/Humdrum/guide.append2.html

- Jianpu notation, microtonal music (Turkish music), AlphaNotes
  - http://bennyt85erhu.wordpress.com/jianpu
  - http://www.hinesmusic.com/What_Are_Makams.html
  - http://blog.finalemusic.com/post/2011/10/20/Finale-Quick-Tips-AlphaNotes.aspx

- More graphic symbol representations for perussion, handbells, haupt-, nebenstimme
  - http://en.wikipedia.org/wiki/Hauptstimme

Data Interchange Cases