Introduction to MusicXML

Notensatz-Konferenz, Salzburg Mozarteum, January 17-18, 2020

Jacques Menu *

December 21, 2019 version

Abstract

This document presents a basic view of MusicXML and a couple of short examples illustrating how MusicXML represents a music score. Our goal is to give a flavor of what MusicXML definitions and data look like from a musician's point of view.

All the examples mentioned can be downloaded from https://github.com/grame-cncm/libmusicxml/tree/lilypond/files/samples/musicxml. They are grouped by subject in sub-directories, such as basic/HelloWorld.xml.

1 Software tools used

The scores fragments shown in this document have been produced by translating the '.xml' files to LilyPond syntax, and then creating the graphical score with LilyPond.

The translations have been done by xml2ly, a prototype tool developed by this author. xml2ly and some of the specific examples presented in this document are this author's contribution to libmusicxml2, an open-source C++ library created and maintained by Dominique Fober at Grame, Lyon, France. The home page to libmusicxml2 is https://github.com/grame-cncm/libmusicxml.

The reader is invited to handle the '.xml' file examples with their own software tools to compare the results with the ones herein.

Tests with other score editing applications are mentioned in this document, namely Sibelius™, Finale™ and MuseScore, which is open-source. musicxml2ly is mentioned too: this translator is supplied with LilyPond. This author doesn't own licenses for other commercial applications such as Dorico™ or Capella™.

2 Overview of MusicXML

2.1 What is MusicXML?

MusicXML (*Music eXtended Markup Language*) is a specification language meant to represent music scores by texts, readable both by humans and computers. It has been designed by the W3C Music Notation Community Group (https://www.w3.org/community/music-notation/) to help sharing music score files between applications, through export and import mechanisms.

The homepage to MusicXML is https://www.musicxml.com.

MusicXML data contains very detailed information about the music score, and it is quite verbose by nature. This makes creating such data by hand quite difficult, and this is done by applications actually.

*Former lecturer in computer science at Centre Universitaire d'Informatique, University of Geneva, Switzerland

2.2 Part-wise vs. measurewise descriptions

MusicXML allows the score to be represented as a sequence of parts, each containing a sequence of measures, or as a sequence of measures, each containing a sequence of parts, i.e. data describing the contents of the corresponding measure in a part.

It seems that measure-wise descriptions have been very little used and then abandoned, and we shall stick to part-wise MusicXML data in this document.

As a historical note, an XSL/XSLT script was supplied in the early days of MusicXML to convert between part-wise and measure-wise representations.

2.3 MusicXML's formal definition

As a member of the *XML family of languages, MusicXML is defined by a DTD (*Document Type Definition*), to be found at https://github.com/w3c/musicxml/tree/v3.1.

The dtds/3.1/schema subdirectory contains '*.mod' text files defining the various concepts. The file common.mod contains definitions used in other '*.mod' files:

```
<!--
  This file contains entities and elements that are common
  across multiple DTD modules. In particular, several elements
  here are common across both notes and measures.
-->
```

For example, note.mod defines the '<backup>' and <forward>' markups this way:

Listing 1: <backup> and <forward> definition

```
<!--
  The backup and forward elements are required to coordinate
  multiple voices in one part, including music on multiple
  staves. The forward element is generally used within voices
  and staves, while the backup element is generally used to
  move between voices and staves. Thus the backup element
  does not include voice or staff elements. Duration values
  should always be positive, and should not cross measure
  boundaries or mid-measure changes in the divisions value.
-->
<!ELEMENT backup (duration, %editorial;)>
<!ELEMENT forward
  (duration, %editorial-voice;, staff?)>
```

and an example of their use:

Listing 2: <backup> and <forward> example

```
      <forward>
        <duration>4</duration>
        <voice>2</voice>
        <staff>1</staff>
      </forward>
      <backup>
        <duration>8</duration>
      </backup>
```

The current version of the MusicXML DTD is 3.1, and there are discussions about version 3.2.

The syntactical aspects of MusicXML are quite simple and regular, which makes it easy to handle MusicXML data with algorithms.

2.4 Markups

MusicXML data is made of so-called markups, delimited by an opener and a closer. The opener is introduced by a '<' and closed by a '>', as in '<part-list>'. The closer is introduced by a '</' and closed by a '>', as in '</part-list>'.

Markups go by pairs, which allows markups to be nested, such as:

```
        <duration>4</duration>
```

and:

```
        <clef>
          <sign>G</sign>
          <line>2</line>
        </clef>
```

Markups can have attributes such as the part name 'P1' in:

```
        <score-part id="P1">
          <part-name>Music</part-name>
        </score-part>
```

The values of attributes can be double-quoted characters strings and integer or floating point numbers.

Some attributes are mandatory such as 'id' in '<score-part>', while others are optional.

It is possible to contract an element that contains nothing between its opener and closer, such as:

```
        <dot></dot>
```

this way:

```
        <dot />
```

Comments can be used in MusicXML data. They start with a '<!--' opener and end with a '-->' closer, as in:

```
<!-------------------------------------------------------------->
    <measure number="1">
  <!-- A very minimal MusicXML example, part P1, measure 1 -->
```

Comments can span several lines.

The spaces and end of lines between markups are ignored.

MusicXML is a representation of HOW TO DRAW a score, which has implications on the kind of markups available, in particular '<forward>' and '<backup>', which are presented at section 6.8

All this makes the syntax of MusicXML data quite regular and simple, and it is easy to program lexical/syntactical analyzers it.

Markups are called 'elements' in the MusicXML DTD, and we shall use that terminology in the remainder of this document.

2.5 What is the semantics of MusicXML data?

It is very difficult though to define the semantics – the meaning of the sentences – of an artificial language in a complete and consistent way, i.e. without omitting anything and without contradictions.

MusicXML is no exception to this rule: there are things unsaid in the DTD, which leaves room to interpretation by the various applications that create or handle MusicXML data.

For example, clefs are defined in attributes.mod, starting with:

```
<!--
  Clefs are represented by the sign, line, and
  clef-octave-change elements. Sign values include G, F, C,
  percussion, TAB, jianpu, and none. Line numbers are
  counted from the bottom of the staff. Standard values are
  ... ... ... ... ...
```

What is a 'none' clef? Is the clef currently in use still to be used from now on, merely hiding the 'none' clef, or should an implicit, default treble clef be used? As it turns out, various applications don't agree on the answer to this question, see the next-to-last measure of clefs/Clefs.xml.

This author has found MusicXML files that contain 'PERCUSSION': is this to be accepted and handled as 'percussion'? This point is not mentioned in the DTD either.

2.6 Overall structure of MusicXML data

MusicXML data consists of:

- a '<?xml>' element indicating the characters encoding used;
- a '<!DOCTYPE>' element telling that the contents is in 'score-partwise' mode;
- a '<score-partwise>' element indicating the MusicXML DTD number that the forthcoming data complies to, and that contains:
  - a '<part-list>' element containing the various '<score-part>'s in the score;
  - a sequence of '<part>' elements in the order they appear in the score, each one containing the measures in the given part, in order.

3 Measurements

3.1 Geometrical lengths

MusicXML represents lengths by 10^{th} of an interline space, i.e. the distance between lines in staves. This relative measure unit has the advantage that if does not change if the score is scaled by some factor.

In common.mod we find:

```
}
<!--
  The tenths entity is a number representing tenths of
  interline space (positive or negative) for use in
  attributes. The layout-tenths entity is the same for
  use in elements. Both integer and decimal values are
  allowed, such as 5 for a half space and 2.5 for a
  quarter space. Interline space is measured from the
  middle of a staff line.
-->
<!ENTITY % tenths "CDATA">
<!ENTITY % layout-tenths "(#PCDATA)">
```

In order to obtain absolute lengths for drawing, MusicXML specifies how many tenths are equal to how many millimeters in the '<scaling>' element, defined in layout.mod:

```
<!--
  Version 1.1 of the MusicXML format added layout information
  for pages, systems, staffs, and measures. These layout
  elements joined the print and sound elements in providing
  formatting data as elements rather than attributes.

  Everything is measured in tenths of staff space. Tenths are
  then scaled to millimeters within the scaling element, used
  in the defaults element at the start of a score. Individual
  staves can apply a scaling factor to adjust staff size.
  When a MusicXML element or attribute refers to tenths,
  it means the global tenths defined by the scaling element,
  not the local tenths as adjusted by the staff-size element.
-->

<!-- .................. -->
```

```
<!--
  Margins, page sizes, and distances are all measured in
  tenths to keep MusicXML data in a consistent coordinate
  system as much as possible. The translation to absolute
  units is done in the scaling element, which specifies
  how many millimeters are equal to how many tenths. For
  a staff height of 7 mm, millimeters would be set to 7
  while tenths is set to 40. The ability to set a formula
  rather than a single scaling factor helps avoid roundoff
  errors.
-->
<!ELEMENT scaling (millimeters, tenths)>
<!ELEMENT millimeters (\#PCDATA)>
<!ELEMENT tenths %layout-tenths;>
```

This leads for example to:

Listing 3: Scaling example

```
        <scaling>
          <millimeters>7.05556</millimeters>
          <tenths>40</tenths>
        </scaling>
```

3.2 Notes durations

MusicXML uses a quantization of the duration with the '`<divisions>`' element, which tells how many divisions there are in a quarter note:

```
       <divisions>2</divisions>
```

This example means that there are 2 division in a quarter note, i.e. the duration measure unit is an eigth note. Let's borrow from physics and MIDI terminology and call this a quantum.

Any multiple of this quantum can be used in the MusicXML data after that specification, but there's no way to express a duration less than an eigth node.

The quantum value has to be computed from the shortest note in the music that follows this element, taking tuplets into account, see section 10.2.

Is it possible to set the quantum to other values in multiple places in the MusicXML data at will if needed? The DTD doesn't mentions that, and in practice, all applications support this feature.

Notes prolongations dots are specified with as many '`<dot>`' elements as needed:

```
<!--
  One dot element is used for each dot of prolongation.
  The placement element is used to specify whether the
  dot should appear above or below the staff line. It is
  ignored for notes that appear on a staff space.
-->
<!ELEMENT dot EMPTY>
<!ATTLIST dot
    %print-style;
    %placement;
>
```

4 Elements attachments decisions

It is not always easy to decide what element a given element should be attached to. Should a '`<dynamics>`' element or '`<metronome>`' element be attached to a note or be placed at the '`<measure>`' level? Is so, should it occur before or after the note over or below which it should be displayed?

MusicXML defines a *direction* as a musical indication that is not necessarily attached to a specific note. Two or more directions may be combined to indicate the start and stop of wedges, dashes, and so on.

For example, '`<dynamics>`' elements are placed outside of '`<note>`' elements in a '`<direction>`' element:

```
      <direction placement="below">
        <direction-type>
          <dynamics>
            <ffff/>
          </dynamics>
        </direction-type>
        <staff>1</staff>
      </direction>
```

5 A complete example

As is usual in computer science, this minimal example is named basic/HelloWorld.xml. It is displayed below, together with the resulting graphic score.

The first line specifies the character encoding of the contents below, here UTF-8. Then the '`!DOCTYPE`' element at lines 2 to 4 tells us that this file contains partwise data conforming to DTD 3.0.

Then the '`<part-list>`' element at lines 7 to 11 contains a list of '`<score-part>`'s with their '`id`' attribute, here '`P1`' alone.

After this, we find the sequence of '`part`'s with their '`id`' attribute, here '`P1`' alone, and, inside it, the single '`<measure>`' element with attribute '`number`' 1.

The nesting of elements, such as '`<key>`' containing a '`<fifths>`' element, leads the structure of a MusicXML representation to be a tree. The way the specification is written conforms to the computer science habit of drawing trees with their root at the top and their leaves at the bottom.

Listing 4: HelloWorld.xml

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<!DOCTYPE score-partwise PUBLIC
    "-//Recordare//DTD MusicXML 3.0 Partwise//EN"
    "http://www.musicxml.org/dtds/partwise.dtd">
<score-partwise version="3.0">
  <!-- A very minimal MusicXML example -->
  <part-list>
    <score-part id="P1">
      <part-name>Music</part-name>
    </score-part>
  </part-list>
  <part id="P1">
<!--========================================================-->
    <measure number="1">
  <!-- A very minimal MusicXML example, part P1, measure 1 -->
      <attributes>
        <divisions>1</divisions>
        <key>
          <fifths>0</fifths>
        </key>
        <time>
          <beats>4</beats>
          <beat-type>4</beat-type>
```

```
        </time>
        <clef>
          <sign>G</sign>
          <line>2</line>
        </clef>
      </attributes>
  <!-- A very minimal MusicXML example, part P1, measure 1, before
   first note -->
      <note>
        <pitch>
          <step>C</step>
          <octave>4</octave>
        </pitch>
        <duration>4</duration>
        <type>whole</type>
      </note>
    </measure>
<!--================================================================-->
  </part>
</score-partwise>
```

6 Score structure

MusicXML data contains a mix of legal informations, score geometry and musical contents. Some aspects of this are presented in this section.

6.1 Identification and rights

The '<identification>' element is defined in identity.mod:

```
<!--
  Identification contains basic metadata about the score.
  It includes the information in MuseData headers that
  may apply at a score-wide, movement-wide, or part-wide
  level. The creator, rights, source, and relation elements
  are based on Dublin Core.
-->
<!ELEMENT identification (creator*, rights*, encoding?,
  source?, relation*, miscellaneous?)>

<!--
  The creator element is borrowed from Dublin Core. It is
  used for the creators of the score. The type attribute is
  used to distinguish different creative contributions. Thus,
  there can be multiple creators within an identification.
  Standard type values are composer, lyricist, and arranger.
  Other type values may be used for different types of
  creative roles. The type attribute should usually be used
  even if there is just a single creator element. The MusicXML
  format does not use the creator / contributor distinction
  from Dublin Core.
-->
<!ELEMENT creator (#PCDATA)>
<!ATTLIST creator
    type CDATA #IMPLIED
>
```

For example, xmlsamples3.1/ActorPreludeSample.xml contains:

Listing 5: Identification and rights example

```
  <identification>
```

```
    <creator type="composer">Lee Actor</creator>
    <rights>© 2004 Polygames.     All Rights Reserved.</rights>
    <encoding>
      <software>Finale v25 for Mac</software>
      <encoding-date>2017-12-12</encoding-date>
      <supports attribute="new-system" element="print" type="yes" value
    ="yes"/>
      <supports attribute="new-page" element="print" type="yes" value="
    yes"/>
      <supports element="accidental" type="yes"/>
      <supports element="beam" type="yes"/>
      <supports element="stem" type="yes"/>
    </encoding>
  </identification>
```

6.2 Score geometry

The dimensions and margins of the graphics score can be specified with the '<page-layout>' element, as in basic/ClefKeyTime.xml:

Listing 6: Page layout example

```
  <defaults>
    <scaling>
      <millimeters>7.05556</millimeters>
      <tenths>40</tenths>
      </scaling>
    <page-layout>
      <page-height>1683.36</page-height>
      <page-width>1190.88</page-width>
      <page-margins type="even">
        <left-margin>56.6929</left-margin>
        <right-margin>56.6929</right-margin>
        <top-margin>56.6929</top-margin>
        <bottom-margin>113.386</bottom-margin>
        </page-margins>
      <page-margins type="odd">
        <left-margin>56.6929</left-margin>
        <right-margin>56.6929</right-margin>
        <top-margin>56.6929</top-margin>
        <bottom-margin>113.386</bottom-margin>
        </page-margins>
      </page-layout>
    <word-font font-family="FreeSerif" font-size="10"/>
    <lyric-font font-family="FreeSerif" font-size="11"/>
    </defaults>
```

6.3 Part groups and parts

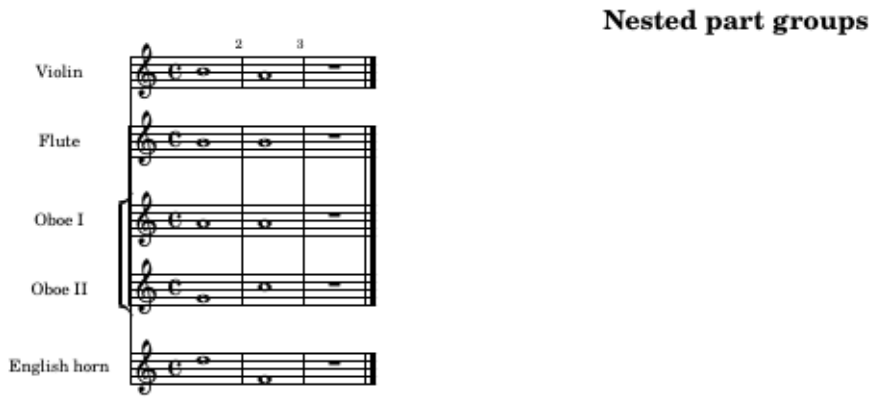
Part groups are used to structure complex scores, mimicking the way large orchestras are organized. For example, there can be a winds group, containing several groups such as flutes, oboes, horns and bassoons.

A '<part-group>' element has a 'type' attribute, whose value can be 'start' or 'stop'. A part group is thus delimited by a pair of '<part-group>' elements, the first one of type 'start', and the second one of type 'stop'.

The 'id' attribute of the '<score-part>' element is used to reference the part later in the MusicXML data. Often, is has the form 'Pn', where 'n' is a number.

Part groups can be nested, leading to a hierarchy of groups. This is done with the 'number' attribute of the '<part-group>' element, which indicates how 'start' and 'stop' attributes are paired together.

For example, partgroups/NestedPartGroups.xml contains:



Listing 7: Nested part groups example

```
  <part-list>
    <score-part id="P1">
      <part-name>Violin</part-name>
    </score-part>
    <part-group number="1" type="start">
      <group-symbol>line</group-symbol>
      <group-barline>yes</group-barline>
    </part-group>
    <score-part id="P2">
      <part-name>Flute</part-name>
    </score-part>
    <part-group number="2" type="start">
      <group-symbol>bracket</group-symbol>
      <group-barline>yes</group-barline>
    </part-group>
    <score-part id="P3">
      <part-name>Oboe I</part-name>
    </score-part>
    <score-part id="P4">
      <part-name>Oboe II</part-name>
    </score-part>
    <part-group number="2" type="stop"/>
    <part-group number="1" type="stop"/>
    <score-part id="P5">
      <part-name>English horn</part-name>
    </score-part>
  </part-list>
```

The MusicXML DTD states that part groups may *overlap*. This seem to be only because Finale™ doesn't create MusicXML markups in a strict first-in, last-out order.

Here is how partgroups/OverlappingPartGroups.xml is handled by various applications:

- Sibelius™ 7.1.3 loses the last group, i.e. the bassoons;
- Finale™ 2014 loses them too;
- MuseScore 3.3.4 crashes;
- `musicxml2ly` loses the bassoons too;
- `xml2ly` rejects such data for the time being, with the message shown below.

```
  ### MusicXML ERROR ### partgroups/OverlappingPartGroups.xml:169:
There are overlapping part groups, namely:
  '2' -=> PartGroup_6 ('2', partGroupName "1
2"), lines 164..169
and
  '1' -=> PartGroup_2 ('1', partGroupName ""), lines 76..170
```

```
7
8  Please contact the maintainers of libmusicxml2 (see option '-c, -
       contact'):
9    either you found a bug in the xml2ly translator,
10   or this MusicXML data is the first-ever real-world case
11   of a score exhibiting overlapping part groups.
12 Abort trap: 6 (core dumped)
```

6.4 Staves and voices

In MusicXML, a part is composed of one or more staves, each composed of one or more voices. There are no structured staves nor voices in MusicXML however – the way parts and measures are. The '`<stave>`' and '`voice`' element only contain a number.

To be more precise:

- stave numbers start at 1 in every part, which refers to the top-most staff in the part;
- a stave number of 1 is implied by default, i.e. when an optional '`<stave>`' element is missing, as can happen in notes descriptions;
- voice numbers start at 1 in every staff, and a voice number of 1 is implied by default, i.e. when an optional '`<voice>`' element is missing;

This author has found MusicXML files in which the voice numbers are not contiguous, such as 1, 5, 9. The DTD doesn't preclude this. The way `multistaff/NonContiguousVoiceNumbers.xml` is handled depends on the application.

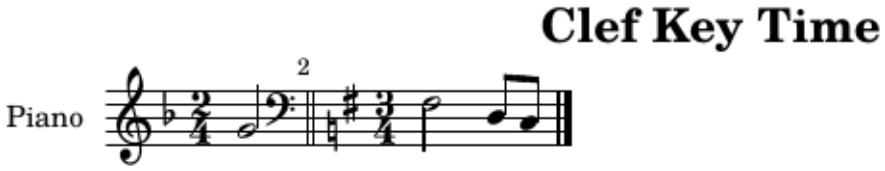
A given voice can change staff and come back to the former one, for example in keyboard scores.

6.5 Clefs, keys and time signatures

MusicXML offers elements to describe the common cases:

- traditional keys are described by a '`<fifths>`' element;
- simple clefs are described by '`<sign>`' and '`<line>`' elements;
- simple time signatures are desribed by '`<beats>`' and '`<beat-type>`' elements.

An example is found in `basic/ClefKeyTime.xml`:



Listing 8: Clef, key and time signature example

```
1        <attributes>
2          <divisions>2</divisions>
3          <key>
4            <fifths>-1</fifths>
5            </key>
6          <time>
7            <beats>2</beats>
8            <beat-type>4</beat-type>
9            </time>
10         <clef>
11           <sign>G</sign>
12           <line>2</line>
13           </clef>
14         </attributes>
15       <!-- ... ... ... ... ... -->
16       <attributes>
```

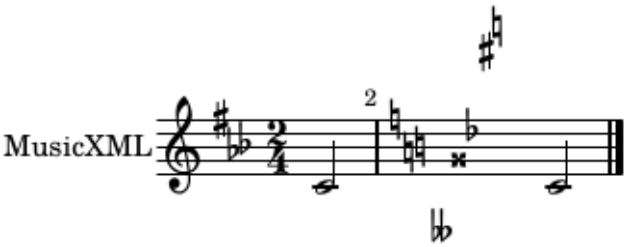
```
        <key>
          <fifths>1</fifths>
          </key>
        <time>
          <beats>3</beats>
          <beat-type>4</beat-type>
          </time>
        <clef>
          <sign>F</sign>
          <line>4</line>
          </clef>
        </attributes>
```

In this example, the various sub-elements are:

Fragment	Meaning
'`<fifths>-1</fifths>`'	the number of fitfhs. A negative number is the number of flats, 0 means C major or A minor, and a positive value is the number of sharps
'`<beats>2</beats>`'	the number of beats per measure
'`<beat-type>4</beat-type>`'	the beat type, i.e. the duration of each beat expressed as a fraction of a whole note
'`<sign>G</sign>`'	the clef sign to be displayed. Sign values include '`G`', '`F`', '`C`', '`percussion`', '`TAB`', '`jianpu`', and '`none`'
'`<line>2</line>`'	the number of the line at which the clef is placed

Composite time signatures such as '`2/4 + 3/8`' and '`3+2/8`' can be specified, as well as '`<senza-misura>`' for cadenzas.

MusicXML also supports non-traditional keys the Humdrum/Scot way. For example, the time signature at the beginning of measure 2 in `keys/HumdrumScotKeys.xml` is described by:



Listing 9: Humdrum/Scot non-traditional key example

```
        <key>
          <key-step>C</key-step>
          <key-alter>-2</key-alter>
          <key-step>G</key-step>
          <key-alter>2</key-alter>
          <key-step>D</key-step>
          <key-alter>-1</key-alter>
          <key-step>B</key-step>
          <key-alter>1</key-alter>
          <key-step>F</key-step>
          <key-alter>0</key-alter>
          <key-octave number="1">2</key-octave>
          <key-octave number="2">3</key-octave>
          <key-octave number="3">4</key-octave>
          <key-octave number="4">5</key-octave>
          <key-octave number="5">6</key-octave>
        </key>
```

This is another example handled diffently by some applications.

6.6 Metromone and tempo

6.7 Measures

The '<measure>' elements can contain many other elements, depending on the music.

Full measures are usually numbered from '1' up, but these numbers are actually character strings, not integers: this allows for special measure numbers such as 'X1', for example, in the case of cue staves.

Anacruses are best specified with '0' as their number and the 'implicit' attribute set to 'yes':

```
    <measure number="0" implicit="yes" width="129.48">
```

One see cases where the number is 1 for anacruses, though.

Staves have numbers from '1' up, with stave number '1' the top-most one in a given part.

6.8 '<forward>' and '<backup>

,

The '<forward>' element is used typically in a second, third or fourth voice which does not contain notes at some point in time. This element allows drawing to continue a bit further in the voice, without drawing rests in-between.

The '<backup>' is needed to move to the left before drawing the next element. This is necessary where there are several voices in a given staff and one switched drawing from one voice to another, whose next element is not at the right of the last one drawn.

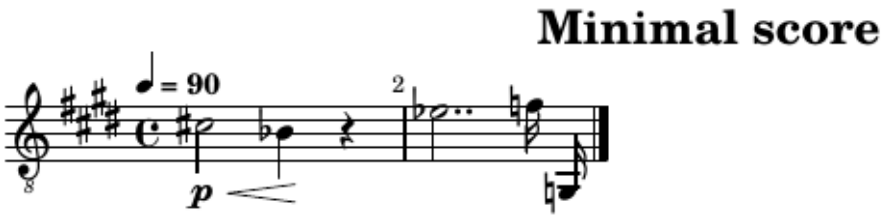
7 Notes

A note is described by a 'note' element, defined in note.mod:

Listing 10: Note definition

```
<!--
  Notations are musical notations, not XML notations. Multiple
  notations are allowed in order to represent multiple editorial
  levels. The print-object attribute, added in Version 3.0,
  allows notations to represent details of performance technique,
  such as fingerings, without having them appear in the score.
-->
<!ELEMENT notations
  (%editorial;,
   (tied | slur | tuplet | glissando | slide |
    ornaments | technical | articulations | dynamics |
    fermata | arpeggiate | non-arpeggiate |
    accidental-mark | other-notation)*)>
<!ATTLIST notations
    %print-object;
    %optional-unique-id;
>
```

The first note in measure 2 in basic/MinimalScore.xml:



is described by:

Listing 11: Note example

```
        <divisions>8</divisions>

      <!-- ... ... ... ... ... -->

        <clef>
          <sign>G</sign>
          <line>2</line>
          <clef-octave-change>-1</clef-octave-change>
        </clef>

      <!-- ... ... ... ... ... -->

      <note>
        <pitch>
          <step>E</step>
          <alter>-1</alter>
          <octave>4</octave>
        </pitch>
        <duration>28</duration>
        <voice>1</voice>
        <type>half</type>
        <dot />
        <dot />
        <accidental>flat</accidental>
      </note>
```

In this example, the various sub-elements are:

Fragment	Meaning
'<step>E</step>'	the diatonic pitch of the note, from A to G
'<alter>-1</alter>'	the chromatic alteration in number of semitones (e.g., -1 for flat, 1 for sharp)
'<octave>4</octave>'	the absolute octave of the note, 0 to 9, where 4 indicates the octave started by middle C
'<duration>28</duration>'	the sounding duration of the note, 28 quanta, which is a double dotted half note with 4 quanta per quarter note (16+8+4)
'<voice>1</voice>'	the voice number of the note, 1
'<type>half</type>'	the display duration of the note, a half note, which determines the note head

Middle C is the one between the left hand and right hand staves in a typical score. Note here: octave numbers are absolute, and the treble clef is octaviated by a '<clef-octave-change>' element!

Voice and staff numbers are optional, in which case the default value is 1.

Having both a sounding and display duration specification is necessary because they do not coincide in the case of dotted notes and tuplets members, see paragraph 10.2 for the latter.

7.1 Accidentals

```
<!--
  Actual notated accidentals. Valid values include: sharp,
  natural, flat, double-sharp, sharp-sharp, flat-flat,
  natural-sharp, natural-flat, quarter-flat, quarter-sharp,
  three-quarters-flat, three-quarters-sharp, sharp-down,
  sharp-up, natural-down, natural-up, flat-down, flat-up,
  double-sharp-down, double-sharp-up, flat-flat-down,
  flat-flat-up, arrow-down, arrow-up, triple-sharp,
  triple-flat, slash-quarter-sharp, slash-sharp, slash-flat,
```

```
  double-slash-flat, sharp-1, sharp-2, sharp-3, sharp-5,
  flat-1, flat-2, flat-3, flat-4, sori, koron, and other.

  The quarter- and three-quarters- accidentals are
  Tartini-style quarter-tone accidentals. The -down and -up
  accidentals are quarter-tone accidentals that include
  arrows pointing down or up. The slash- accidentals
  are used in Turkish classical music. The numbered
  sharp and flat accidentals are superscripted versions
  of the accidental signs, used in Turkish folk music.
  The sori and koron accidentals are microtonal sharp and
  flat accidentals used in Iranian and Persian music. The
  other accidental covers accidentals other than those listed
  here. It is usually used in combination with the smufl
  attribute to specify a particular SMuFL accidental. The
  smufl attribute may be used with any accidental value to
  help specify the appearance of symbols that share the same
  MusicXML semantics. The attribute value is a SMuFL canonical
  glyph name that starts with acc.

  Editorial and cautionary indications are indicated
  by attributes. Values for these attributes are "no" if not
  present. Specific graphic display such as parentheses,
  brackets, and size are controlled by the level-display
  entity defined in the common.mod file.
-->
<!ELEMENT accidental (#PCDATA)>
<!ATTLIST accidental
    cautionary %yes-no; #IMPLIED
    editorial %yes-no; #IMPLIED
    %level-display;
    %print-style;
    %smufl;
>
```

7.2 Articulations

The MusicXML articulation elementss are:

```
<!--
  Articulations and accents are grouped together here.
-->
<!ELEMENT articulations
  ((accent | strong-accent | staccato | tenuto |
    detached-legato | staccatissimo | spiccato |
    scoop | plop | doit | falloff | breath-mark |
    caesura | stress | unstress | soft-accent |
    other-articulation)*)>
<!ATTLIST articulations
    %optional-unique-id;
>
```

7.3 Ornaments

Ornaments are defined in note.mod:

```
<!--
  Ornaments can be any of several types, followed optionally
  by accidentals. The accidental-mark element's content is
  represented the same as an accidental element, but with a
  different name to reflect the different musical meaning.
-->
```

```
<!ELEMENT ornaments
  (((trill-mark | turn | delayed-turn | inverted-turn |
     delayed-inverted-turn | vertical-turn |
     inverted-vertical-turn | shake | wavy-line |
     mordent | inverted-mordent | schleifer | tremolo |
     haydn | other-ornament), accidental-mark*)*)>
<!ATTLIST ornaments
    %optional-unique-id;
>
<!ELEMENT trill-mark EMPTY>
<!ATTLIST trill-mark
    %print-style;
    %placement;
    %trill-sound;
>
```

7.4 Dynamics

MusicXML dynamics are defined in common.mod:

```
<!ELEMENT dynamics ((p | pp | ppp | pppp | ppppp | pppppp |
  f | ff | fff | ffff | fffff | ffffff | mp | mf | sf |
  sfp | sfpp | fp | rf | rfz | sfz | sffz | fz |
  n | pf | sfzp | other-dynamics)*)>
<!ATTLIST dynamics
    %print-style-align;
    %placement;
    %text-decoration;
    %enclosure;
    %optional-unique-id;
```

Other dynamics can be specified:

```
The other-dynamics element
  allows other dynamic marks that are not covered here, but
  many of those should perhaps be included in a more general
  musical direction element. Dynamics may also be combined as
  in <sf/><mp/>.
```

8 Slurs and ties

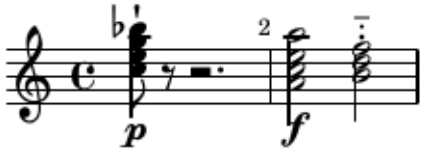
9 Harmony and figured bass

10 Chords and tuplets

10.1 Chords

Chords are not evidenced as such in MusicXML data. Instead, the '<chord>' element means that the given note is part of a chord after the first note in the chord has be met. Remember: MusicXML is about drawing scores. Put it another way, you know there is a chord upon its second note.

The code for the last three note chord in chords/Chords.xml is shown below.



Listing 12: Chord example

```
      <note>
        <pitch>
          <step>B</step>
          <octave>4</octave>
        </pitch>
        <duration>4</duration>
        <voice>1</voice>
        <type>half</type>
        <notations>
          <articulations>
            <staccato />
            <detached-legato />
          </articulations>
        </notations>
      </note>
      <note>
        <chord />
        <pitch>
          <step>D</step>
          <octave>5</octave>
        </pitch>
        <duration>4</duration>
        <voice>1</voice>
        <type>half</type>
      </note>
      <note>
        <chord />
        <pitch>
          <step>F</step>
          <octave>5</octave>
        </pitch>
        <duration>4</duration>
        <voice>1</voice>
        <type>half</type>
      </note>
```

10.2 Tuplets

The situation for tuplets is different than that of the chords: there is a '`<tuplet>`' element, with a '`type`' attribute to indicate the note upon which it starts and stops:

```
        <notations>
          <tuplet number="1" type="start" />
        </notations>
```

The '`number`' attribute can be used to describe nested tuplets:

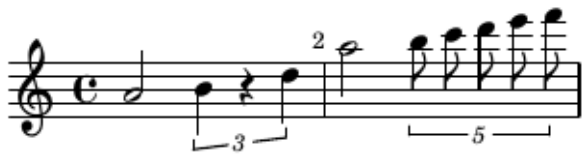
The contents, i.e. the notes in the tuplet, are not nested in the latter: there are placed in sequence between the two '`<tuplet>`' elements that delimitate the tuplet.

Each note in the tuplet has a '`<time-modification>`' element, from the first one on. This element contains two elements:

```
        <time-modification>
          <actual-notes>3</actual-notes>
          <normal-notes>2</normal-notes>
        </time-modification>
```

One should play '`<actual-notes>`' within the time taken by only '`<normal-notes>`'. The example above is thus that of a triplet.

In the case of `tuplets/Tuplets.xml`, shown below, the duration of the tuplets member is 20 quanta, i.e. 2/3 of a quarter note, whose duration is 30, and the 'display' duration is a quarter note. The duration of the triplet as a whole is that of a half note, i.e. 60 quanta.



Listing 13: Tuplet example

```
        <divisions>30</divisions>

      <!-- ... ... ... ... ... -->

      <note>
        <pitch>
          <step>B</step>
          <octave>4</octave>
        </pitch>
        <duration>20</duration>
        <voice>1</voice>
        <type>quarter</type>
        <time-modification>
          <actual-notes>3</actual-notes>
          <normal-notes>2</normal-notes>
        </time-modification>
        <notations>
          <tuplet number="1" type="start" />
        </notations>
      </note>
      <note>
        <rest />
        <duration>20</duration>
        <voice>1</voice>
        <type>quarter</type>
        <time-modification>
          <actual-notes>3</actual-notes>
          <normal-notes>2</normal-notes>
        </time-modification>
      </note>
      <note>
        <pitch>
          <step>D</step>
          <octave>5</octave>
        </pitch>
        <duration>20</duration>
        <voice>1</voice>
        <type>quarter</type>
        <time-modification>
          <actual-notes>3</actual-notes>
          <normal-notes>2</normal-notes>
        </time-modification>
        <notations>
          <tuplet number="1" type="stop" />
        </notations>
      </note>
```

11 Barlines and repeats

Repeats are not described by high-level elements in MusicXML. Instead, specific barlines containing a '`<repeat>`' element are used to draw the necessary delimiters.

11.1 Simple barlines

The '<barline>' element is defined in barline.mod. It has two main attributes:

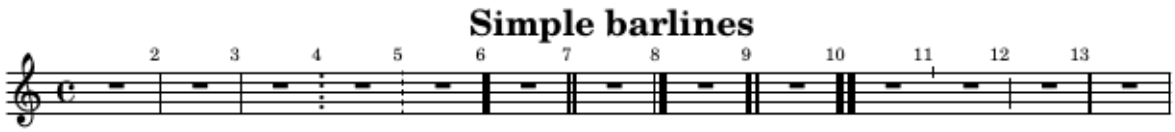
Attribute	Meaning
bar-style	Bar-style contains style information. Choices are 'regular', 'dotted', 'dashed', 'heavy', 'light-light', 'light-heavy', 'heavy-light', 'heavy-heavy', 'tick' (a short stroke through the top line), 'short' (a partial barline between the 2nd and 4th lines), and 'none'. Barlines can occur within measures, as in dotted barlines that subdivide measures in complex meters;
location	If location is 'left', it should be the first element in the measure, aside from the 'print', 'bookmark', and 'link' elements. If location is 'right', it should be the last element, again with the possible exception of the 'print', 'bookmark', and 'link' elements. The value can be 'right', 'left' or 'middle'. If no location is specified, the default value is 'right'.

In the '<bar-style>' element, 'light' is a thin vertical line, and 'heavy'is a thick line. The final barline of a piece is thus represented by:

Listing 14: Final barline

```
<barline location="right">
  <bar-style>light-heavy</bar-style>
  </barline>
```

One can see the various simple barlines in barlines/SimpleBarlines.xml:



11.2 Repeats

The '<repeat>' element in barline can contains these attributes, also defined in barline.mod:

Attribute	Meaning
direction	'forward' is used at the start of a repeat, and 'backward' is used at the end of it;
times	indicates how many times the repeated section as to be played;
winged	indicates whether has winged extensions that appear above and below the barline, to make them easier to see; The 'straight' and 'curved' values represent single wings, while the 'double-straight' and 'double-curved' values represent double wings. The 'none' value indicates no wings and is the default.

```
<barline location="right">
  <bar-style>light-heavy</bar-style>
  <repeat direction="backward" times="5"/>
</barline>
```

```
<barline location="right">
  <bar-style>light-heavy</bar-style>
  <repeat direction="backward" winged="none"/>
</barline>
```

12 Lyrics

13 Creating MusicXML data

This can be done in various ways:

- by hand, using a text editor: possible, but unrealistic for usual scores;
- by exporting the score as an MusicXML text file with a GUI music score editor;
- by scanning a graphics files containing a ready-to-print score, with tools such as PhotoScore Ultimate^{TM};
- by programming an application that outputs MusicXML text.

This author has performed manual text editing on some of the samples supplied with `libmusicxml2` in order to perform tests and debug `xml2ly`, but this is a particular case.

Exporting to MusicXML is probably the most frequent way, and there are applications that do a good job at that. If an application supports say strings instruments scordaturas in scores, then creating a '`<scordatura>`' element is not very difficult.

Scanning graphical scores is a tough problem: how do you tell lyrics from annotations such as '`cresc.`' or tempos such as '`Allegro`'? One usually has to manually fix scanning errors and the category of some text fragments after scanning to get good results. And, then, the scanning application should create quality MusicXML data.

Creating MusicXML by an application is a matter of computer programming, and requires development skill. As an example, `libmusicxml2` supplies the necessary tools, and one can obtain:

```
        <key>
          <fifths>1</fifths>
        </key>
```

with C++ code such as:

Listing 15: Creating a key element in an application

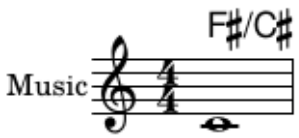
```
Sxmlelement attributes = factory::instance().create(k_attributes);

Sxmlelement key = factory::instance().create(k_key);
key->push (newElement(k_fifths, "1"));
attributes->push (key);
```

14 Importing MusicXML data

Many GUI applications provide a way to import MusicXML data, often with some limitations. We show some of them here.

14.1 Small element, big effect

In harmonies/Inversion.xml, shown below, there is a harmony with an '`<inversion>`' element. A number of applications ignore this element when importing MusicXML data, because it takes a full knowledge of chords structures to compute the bass note of inverted chords.



Listing 16: Harmony inversion

```
      <harmony>
        <root>
```

```
          <root-step>F</root-step>
          <root-alter>1</root-alter>
        </root>
        <kind>major</kind>
        <inversion>2</inversion>
      </harmony>
```

14.2 Elements handled in different ways

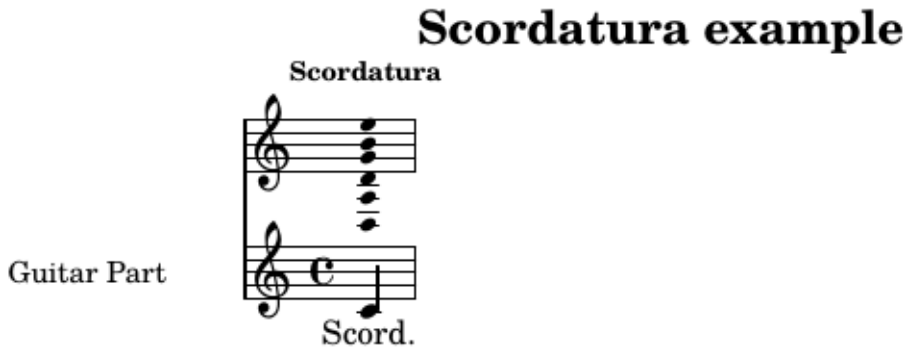
14.3 Elements often not well handled

There are elements that are not displayed in a "standard" way by the usual music score editors. One of them is the '`<beat-repeat>`'.

14.4 Elements usually not handled

There are elements that are not displayed by the usual music score editors, because there is no "standard" way to do so. One of them is the scordatura used on string instrument.

For example, the scordatura in `strings/Scordatura.xml` is the case where the sixth string of the guitar is tuned a tone down to D, which can be described by:



Listing 17: Scordatura example

```
        <scordatura>
            <accord string="6">
              <tuning-step>D</tuning-step>
              <tuning-alter>0</tuning-alter>
              <tuning-octave>3</tuning-octave>
            </accord>
            <accord string="5">
              <tuning-step>A</tuning-step>
              <tuning-alter>0</tuning-alter>
              <tuning-octave>3</tuning-octave>
            </accord>
            <accord string="4">
              <tuning-step>D</tuning-step>
              <tuning-alter>0</tuning-alter>
              <tuning-octave>4</tuning-octave>
            </accord>
            <accord string="3">
              <tuning-step>G</tuning-step>
              <tuning-alter>0</tuning-alter>
              <tuning-octave>4</tuning-octave>
            </accord>
            <accord string="2">
              <tuning-step>B</tuning-step>
              <tuning-alter>0</tuning-alter>
              <tuning-octave>4</tuning-octave>
            </accord>
            <accord string="1">
```

```
                <tuning-step>E</tuning-step>
                <tuning-alter>0</tuning-alter>
                <tuning-octave>5</tuning-octave>
              </accord>
            </scordatura>
```

14.5 A real challenge

The contents of challenging/BeethovenNinthSymphony.xml is over 66 megabytes large. It was created by exporting it from Sibelius™, and contains the whose score for this symphony. One can imagine the amount of work to create the score in the first place, and, of course, there's no way a human could create such MusicXML data by hand.

The interested reader is urged to try and import this file in their favorite score editing sofware. This author's experience is that:

- Sibelius™ 7.1.3 handles it alright;
- Finale™ 2014 finds it well-formed, but too big to be opened;
- MuseScore 3.3 opens it, but then working on the file is extremely slow;
- `musicxml2ly` converts it to LilyPond syntax as of 2.19.83, and the result has some issues that should be fixed rather easily;
- `xml2ly` converts it to LilyPond alright, but the issues in this case show that this converter is still experimental...

15 Conclusion

There is a lot of information about MusicXML on the Internet. And of course, plenty of targeted, ready-to-use examples can be found in files/samples/musicxml.

MusicXML has become a de facto standard for music scores data interchange between applications. As as been shown in this document, the way it is exported and imported by the various applications is quite diverse, and manual editing of the result is to be expected after import.

MusicXML is not the whole story, though. The W3C Music Notation Community Group is working on MNX (https://w3c.github.io/mnx), as a successor to MusicXML. One part of it is MNX-Common, which aims at being less verbose and more semantics-oriented than MusicXML.

For example, consider:

```
<score-partwise version="3.1">
    <part-list>
        <score-part id="P1">
            <part-name>Music</part-name>
        </score-part>
    </part-list>
    <part id="P1">
        <measure number="1">
            <attributes>
                <divisions>1</divisions>
                <key>
                    <fifths>0</fifths>
                </key>
                <time>
                    <beats>4</beats>
                    <beat-type>4</beat-type>
                </time>
                <clef>
                    <sign>G</sign>
                    <line>2</line>
```

```
                </clef>
            </attributes>
            <note>
                <pitch>
                    <step>C</step>
                    <octave>4</octave>
                </pitch>
                <duration>4</duration>
                <type>whole</type>
            </note>
        </measure>
    </part>
</score-partwise>
```

In MNX-Common, this can be written:

Listing 18: MNX-Common example

```
<mnx>
    <score>
        <mnx-common profile="standard">
            <global>
                <measure>
                    <directions>
                        <time signature="4/4"/>
                    </directions>
                </measure>
            </global>
            <part>
                <part-name>Music</part-name>
                <measure barline="regular">
                    <sequence>
                        <directions>
                            <clef sign="G" line="2"/>
                        </directions>
                        <event value="/1">
                            <note pitch="C4"/>
                        </event>
                    </sequence>
                </measure>
            </part>
        </mnx-common>
    </score>
</mnx>
```

Let's conclude with a tribute to the manual score engravers, whose skills have produced so many beautiful scores for centuries! Reaching the quality of their work is still a challenge for current music scoring software.

Listings

Contents