Introduction to MusicXML

Notensatz-Konferenz, Salzburg Mozarteum, January 17-18, 2019

Jacques Menu [*]

December 2, 2019

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 subdirectories, such as '`basic/HelloWorld.xml`'.
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. The reader can handle the '`.xml`' files with their own software tools to compare the results with the ones herein.

1 What MusicXML is

MusicXML (*Music eXtended Markup Language*) is a way to represent a music score by a text, readable both by humans and computers. It has been designed by the W3C Music Notation Community Group to help sharing music score files between applications, using export/import commands provided by the latter.

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

MusicXML data contains very detailed information about the music score, and is quite verbose by nature.

2 MusicXML 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 '`schema`' subdirectory contains '`*.mod`' text files defining the various concepts. '`common.mod`' contains definitions used in other '`*.mod`' files.

For example, here is how the '`backup`' and `forward`' markups:

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

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

Listing 1: <backup> and <forward> example

are defined in 'note.mod':

```
<!--
  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?)>
```

Listing 2: <backup> and <forward> definition

The current MusicXML DTD version 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 these aspects with algorithms.

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, and there are things unsaid in the DTD, which leaves room to interpretation by the various applications that create or handle MusicXML data.

3 part-wise vs. measure-wise

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 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.

4 Markups

MusicXML data is made of so-called markups, composed of two parts. The opener is introduced by a '`<`' and closed by a '`>`', as in '`<part-list>`'. The closer and second part is introduced by a '`</`' and closed by a '`>`', as in '`</part-list>`'.

Markups can be self sufficient, as the example above, or go by pairs, which allows nesting markups, 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>
```

Some such 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 '<!--' and end with '-->', 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 below.

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

5 A first example

As is usual in computer science, this example is named 'basic/HelloWorld.xml'. It is displayed in figure 1, 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 part-wise 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.

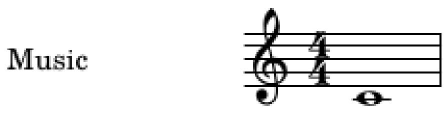
6 Part groups and parts

7 Staves and voices

There's no staff or voice component as such in MusicXML data: one knows they exist because they are mentioned in notes and other elements, such as:

```
```

Figure 1: Contents of 'basic/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>
```

8 Staves and voices

9 Numbering

The various parts in a (part-wise) MusicXML descriptions are usually named from 'P1' on, but any name could be used.

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.

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

10 Measurements

MusicXML represents lengths by 10^{th} of in 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, MusicXML specifies how many tenths there 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:

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

11 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 quantum is an eigth note. Any multiple of this quantum can be used, 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 below.

Is it possible to set the quantum to other values later in the MusicXML data at will if needed.

12 Notes

13 Measures

14 '`<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.

15 Chords

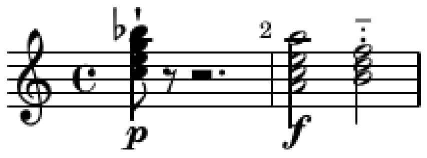
Chords are not evidenced as such in MusicXML data. Instead, the '`<chord>`' element means that the given note is part of a chord after the first note in the chord has be met. Remember: MusicXML is about drawing scores. Put it another way, you know there is a chord upon its second note.

The code for the last three note chord in '`chords/Chords.xml`' is shown in figure 2 .

16 Tuplets

The situation for tuplets is different than that of the chords: each note in the tuplet has a '`<time-modification>`' element, from the first one on, as shown in figure 3. This element contains two elements:

Figure 2: Last chord from 'chords/Chords.xml'



```
      <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>
```

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

One should play '`<actual-notes>`' when there usually only '`<normal-notes>`'. The example above is thus that of a triplet of quarter notes, and the duration of the triplet as a whole is that of a half note.

17 Small element, big effect

In '`harmonies/Inversion.xml`', shown in figure 4, 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.

18 Further reading

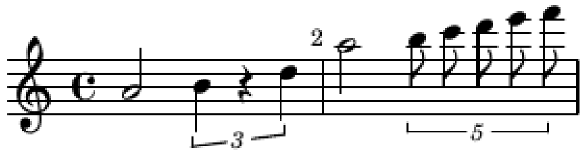
There is a lot of information about MusicXML on the Internet. And of course, plenty of examples can be found at `https://github.com/grame-cncm/libmusicxml/tree/lilypond/files/samples/musicxml`.

Listings

List of Figures

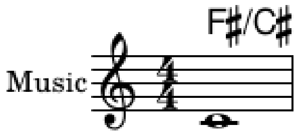
Contents

Figure 3: First tuplet from 'tuplets/Tuplet.xml'



```
      <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>
```

Figure 4: Harmony inversion from 'harmonies/Inversion.xml'



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