Introduction to MusicXML

Jacques Menu

February 18, 2022, revision 1.2

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. We use a combination of formal definitions from the MusicXML DTD and free text explanations.

All the examples mentioned can be downloaded from https://github.com/jacques-menu/musicformats/tree/master/files/musicxml. They are grouped by subject in subdirectories, such as basic/HelloWorld.xml.

The contents of this document is verbose... because MusicXML itself is!

Contents

Listings

1 Software tools used

MusicXML files have been named with a '`.xml`' suffix for years, until it was decided rather recently that this should be changed to '`.musicxml`'. There are GUI applications that filter the file names in their '`open`' or '`import`' dialogs and don't know that change yet, though. We will thus stick to the '`.xml`' suffix convention.

The scores fragments shown in this document have been produced by translating the '`.xml`' file 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 Foberat Grame, Lyon, France.
The home page to `libmusicxml2` is https://github.com/jacques-menu/musicformats, and `xml2ly` is in the '`lilypond`' branch.

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

Other score editing applications are mentioned in this document, namely Sibelius™ 7.1.3, Finale™ 2014 and MuseScore 3.3.4 (https://musescore.org), which is open-source. This author doesn't own licenses for other commercial applications such as Dorico™ or Capella™.

`musicxml2ly` is mentioned too: this converter of MusicXML to LilyPond is supplied with LilyPond. The design goals of `xml2ly` are to perform at least as well as `musicxml2ly`, while providing as many options as needed to avoid too much editing of the LilyPond code generated.

2 Overview of MusicXML

2.1 What is MusicXML?

MusicXML (*Music eXtended Markup Language*, https://www.musicxml.com) is a specification language meant to represent western notation music scores by texts, readable both by humans and computers, and to help sharing music score files between applications, through export and import mechanisms..
It has been invented by Michael Good and initially developed by Recordare LLC, which was bought by MakeMusic in 2011, and finally transferred to the W3C Music Notation Community Group (https://www.w3.org/community/music-notation/) in 2015. See https://en.wikipedia.org/wiki/MusicXML for more historical details.

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

2.2 Part-wise vs. measure-wise 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 formats.

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.

An *XML DTD defines:

- *elements*, used to structure the data since they can be nested;
- *attributes*, that attach named values to the elements;
- *entities*, which are names of groups of elements, such as '`layout-tenths`' and '`start-stop`'. There are used to structure the DTD itself when those elements occur at multiple places in the DTD, and make the latter more readable and easier to update.

For example, consider:

```
  <part id="P1">
    <measure number="1" width="464.29">
    <!-- ... ... ... ... ... -->
```

We see that:

- the `<part/>` element contains a nested `<measure/>` element;
- the `<part/>` element has an '`"id"`' attribute containing the name of the part, '`"P1"`';
- the `<measure/>` element's attributes contain measure number '`1`'and width '`464.29`'.

Some elements contain a single unnamed data item, such as durations and voice and staff numbers:

```
        <duration>4</duration>
        <voice>2</voice>
        <staff>1</staff>
```

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?)>
```

An example of their use is:

Listing 2: `<backup/>` and `<forward/>` example

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

In DTDs, sub-elements can be followed by one of these characters, which mean, as is usual in computer science:

- '?': 0 or 1 occurence, i.e. optional;
- '*': 0 or more occurrences;
- '+': 1 or more occurrences.

One can see in the definition of the '`<<forward>/>`' element that the '`<<duration>/>`' element is mandatory, while the '`<<staff>/>`' element is optional. The text in the DTD tells that staff 1 in implied if it not specified.

In a DTD, '`<CDATA/>`' means *Character Data*. Such data is not analyzed by the software that reads the MusicXML data, it is merely passed over verbatim to whoever asked the data to be read in.

In the same vein, '`<PCDATA/>`' means *Parsed Character Data*, that is, mixed content XML data that are analyzed by software tools.

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 (the 'M' in 'XML'), delimited by an *start-tag* and a *stop-tag*.

The start-tag is introduced by a '<' and closed by a '>', as in '<<part-list>/>'. The stop-tag is introduced by a '</' and closed by a '>', as in '<</part-list>/>'.

Markups go by pairs, as in:

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

The spaces and end of lines between markups are ignored.

It is possible to contract an element that contains nothing between its start-tag and stop-tag, such as:

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

which can be written:

```
1         <dot />
```

We shall call *sub-element* and element nested into another one. '<<part-name>/>' is thus a sub-element of element '<<score-part/>' above.

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.

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

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

Comments can span several lines.

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 16, page 37.

The syntax of MusicXML data quite regular and simple, and it is easy to program lexical/syntactical analyzers for it.

2.5 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 score header containing:
  - an optional '<work>' element, containing sub-elements such as '<work-number>', '<work-title>' and '<opus>';
  - optional '<movement-number>', '<movement-title>', '<identification>' and '<defaults>' sub-elements;
  - 0 or more '<credit>' elements;

  - 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.

Here is how the score header is actually defined in score.mod:

```
<!--
  The score-header entity contains basic score metadata
  about the work and movement, score-wide defaults for
  layout and fonts, credits that appear on the first page,
  and the part list.
-->
<!ENTITY % score-header
  "(work?, movement-number?, movement-title?,
    identification?, defaults?, credit*, part-list)">
```

2.6 What is the semantics of MusicXML data?

We have seen in section section 2.3, page 5, that not specifying the staff number in a '<forward>' element implies a value of one.

It is very difficult 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.

3 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>' sub-element whose attribute 'number' contains '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 3: `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">
  <work>
    <work-title>Hello World!</work-title>
    </work>
  <!-- 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>
```

4 Measurements

4.1 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 it allows all lengths to be represented independantly of the actual size of the score.

In `common.mod` we find:

Listing 4: Relative lengths

```
<!--
  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`:

Listing 5: Abolute lengths

```
<!--
  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 6: Scaling example

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

4.2 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' divisions 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 section 13, page 29.

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 prolongation 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;
>
```

4.3 Graphics and sound

MusicXML has to account for the possible difference between the drawn head note and the duration of that note, as is the case in tuplets.

In a tuplet containing 3 sixteenth notes, the duration of each such note is one third of that of an eigth note, but the drawn head note's graphical duration is half of that of the latter. See section section 13, page 29, for an example of how this is represented.

Some elements in MusicXML data are specifically meant for MIDI support: they refer to the sound durations only.

5 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 "the purist way", with '0' as their number and the 'implicit' attribute set to 'yes', which specifies that this measure number should not be printed. One sees cases where the number is '1' for anacruses, though:

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

Measures can be irregular, i.e. with less total duration as the current time signature, or much longer that the usual time signatures, see section 15, page 34, for an example.

6 Elements attachment decisions

The MusicXML designers had 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 in the data 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, at the measure level:

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

The elements attached to notes are placed inside a '<notations>' element, itself placed inside a '<note>' element. Notations are defined in note.mod:

Listing 7: Notations 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;
>
```

7 Score description structure

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

7.1 Identification, rights and credits

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?)>
```

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

Listing 8: 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>
```

The '<credit>' element, defined in score.mod, represents various legal informations about the score. It contains placement indication such as page number and alignment, as well as fonts information.

For example, one finds in xmlsamples3.1/ActorPreludeSample.xml:

Listing 9: Credits example

```
  <credit page="1">
    <credit-type>title</credit-type>
    <credit-words default-x="1447" default-y="3477" font-size="19.5"
   justify="center" valign="top">Prelude to a Tragedy</credit-words>
  </credit>
  <credit page="1">
    <credit-type>composer</credit-type>
    <credit-words default-x="2718" default-y="3387" font-size="7.8"
   justify="right" valign="top">Lee Actor (2003)</credit-words>
  </credit>
  <credit page="1">
    <credit-type>rights</credit-type>
    <credit-words default-x="1447" default-y="45" font-size="7.8"
   justify="center" valign="bottom" xml:space="preserve">© 2004
   Polygames.     All Rights Reserved.</credit-words>
```

```
  </credit>
  <credit page="2">
    <credit-type>page number</credit-type>
    <credit-words default-x="1412" default-y="45" font-size="7.8"
   halign="center" valign="bottom">- 2 -</credit-words>
  </credit>
  <credit page="3">
    <credit-type>page number</credit-type>
    <credit-words default-x="1447" default-y="45" font-size="7.8"
   halign="center" valign="bottom">- 3 -</credit-words>
  </credit>
  <credit page="4">
    <credit-type>page number</credit-type>
    <credit-words default-x="1412" default-y="45" font-size="7.8"
   halign="center" valign="bottom">- 4 -</credit-words>
  </credit>
```

We see the '<credit-words>' element in the example above. In MusicXML, 'words' means text, as defined in direction.mod:

```
<!--
  The words element specifies a standard text direction.
  Left justification is assumed if not specified.
  Language is Italian ("it") by default. Enclosure
  is none by default.
-->
<!ELEMENT words (#PCDATA)>
<!ATTLIST words
    %text-formatting;
    %optional-unique-id;
>
```

7.2 Score geometry

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

Listing 10: 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>
```

7.3 Part groups and parts

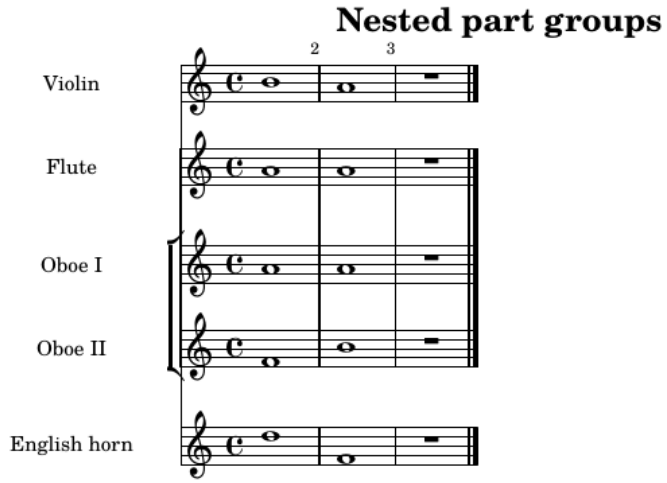
Part groups are used to structure complex scores, mimicking the way large orchestras are organized. For example, there can be a winds group, containing several groups such as flutes, oboes, horns and bassoons.

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 11: 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 author suspects that this is only because Finale™ doesn't create MusicXML markups in a strict first-in, last-out order.

Various applications handle lilypond-ignored/OverlappingPartGroups.xml their own way. `xml2ly` rejects such data for the time being, with this message:

Listing 12: Overlapping groups `xml2ly` error message

```
  ### MusicXML ERROR ### lilypond-ignored/OverlappingPartGroups.xml:39:
There are overlapping part groups, namely:
  '1' -=> PartGroup_2 ('1', partGroupName "Part group 1"), lines 15..39
and
  '2' -=> PartGroup_3 ('2', partGroupName "Part group 2"), lines 27..43

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

7.4 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 as such in MusicXML however – that is, not 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 staff 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;

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

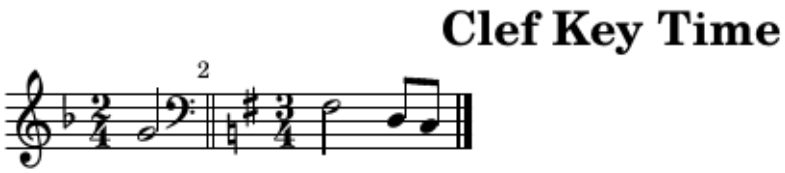
This author has found MusicXML files in which the voice numbers are not contiguous, such as '`1`', '`5`' and '`9`'. The DTD doesn't preclude this, and the applications handle example multistaff/NonContiguousVoiceNumbers.xml their own way.

7.5 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 13: Clef, key and time signature example

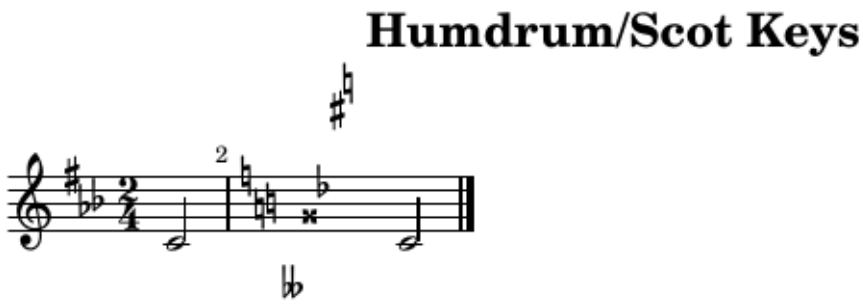
```
      <attributes>
        <divisions>2</divisions>
        <key>
          <fifths>-1</fifths>
          </key>
        <time>
          <beats>2</beats>
          <beat-type>4</beat-type>
          </time>
        <clef>
          <sign>G</sign>
          <line>2</line>
          </clef>
        </attributes>
      <!-- ... ... ... ... ... -->
      <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:



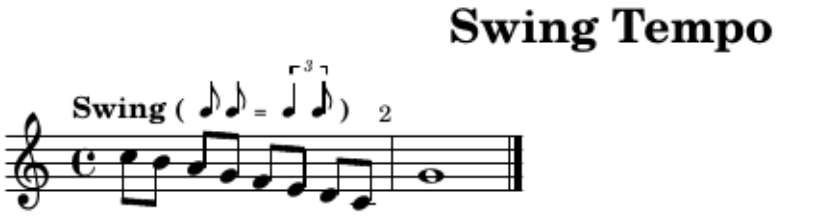
Listing 14: Humdrum/Scot non-traditional key example

```
        <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.

7.6 Metromone and tempo

MusicXML has rich support for metronome specifications. Example tempos/SwingTempo.xml contains:



Listing 15: Swing tempo example

```
    <direction placement="above">
       <direction-type>
         <words>Swing</words>
       </direction-type>
       <direction-type>
         <metronome parentheses="yes" default-y="30" halign="left"
relative-x="26">
           <metronome-note>
             <metronome-type>eighth</metronome-type>
             <metronome-beam number="1">begin</metronome-beam>
           </metronome-note>
```

```
            <metronome-note>
              <metronome-type>eighth</metronome-type>
              <metronome-beam number="1">end</metronome-beam>
            </metronome-note>
            <metronome-relation>equals</metronome-relation>
            <metronome-note>
              <metronome-type>quarter</metronome-type>
              <metronome-tuplet bracket="yes" show-number="actual" type
    ="start">
                <actual-notes>3</actual-notes>
                <normal-notes>2</normal-notes>
                <normal-type>eighth</normal-type>
              </metronome-tuplet>
            </metronome-note>
            <metronome-note>
              <metronome-type>eighth</metronome-type>
              <metronome-tuplet type="stop">
                <actual-notes>3</actual-notes>
                <normal-notes>2</normal-notes>
                <normal-type>eighth</normal-type>
              </metronome-tuplet>
            </metronome-note>
          </metronome>
        </direction-type>
      </direction>
```

8 Notes

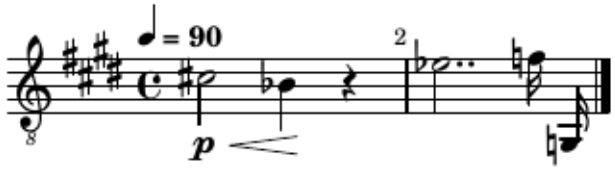
A note is described by a 'note' element, defined in note.mod:

Listing 16: Note definition

```
<!--
  Notes are the most common type of MusicXML data. The
  MusicXML format keeps the MuseData distinction between
  elements used for sound information and elements used for
  notation information (e.g., tie is used for sound, tied for
  notation). Thus grace notes do not have a duration element.
  Cue notes have a duration element, as do forward elements,
  but no tie elements. Having these two types of information
  available can make interchange considerably easier, as
  some programs handle one type of information much more
  readily than the other.
-->
<!ELEMENT note
  (((grace, ((%full-note;, (tie, tie?)?) | (cue, %full-note;))) |
    (cue, %full-note;, duration) |
    (%full-note;, duration, (tie, tie?)?)),
   instrument?, %editorial-voice;, type?, dot*,
   accidental?, time-modification?, stem?, notehead?,
   notehead-text?, staff?, beam*, notations*, lyric*, play?)>
```

Consider basic/MinimalScore.xml:

Minimal score



The first note in measure 2 in this example is described by:

Listing 17: Minimal score 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 section 13, page 29, for the latter.

Note elements can have '<stem>' and '<beam>' sub-elements attached to them, as in the following example. See section 8.6, page 24 for a score containing some:

```
      <note>
        <pitch>
          <step>A</step>
          <octave>2</octave>
        </pitch>
        <voice>3</voice>
        <type>16th</type>
        <stem>up</stem>
        <staff>2</staff>
        <beam number="1">begin</beam>
        <beam number="2">begin</beam>
      </note>
```

Before showing an example, we shall look into more detail in the elements that are attached to notes in the forthcoming sections.

8.1 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.
  ... ... ... ... ...
-->
<!ELEMENT accidental (#PCDATA)>
<!ATTLIST accidental
    cautionary %yes-no; #IMPLIED
    editorial %yes-no; #IMPLIED
    %level-display;
    %print-style;
    %smufl;
>
```

8.2 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;
>
```

8.3 Ornaments

Ornaments are defined in note.mod:

```
<!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;
>
```

8.4 Dynamics

MusicXML dynamics are defined in common.mod:

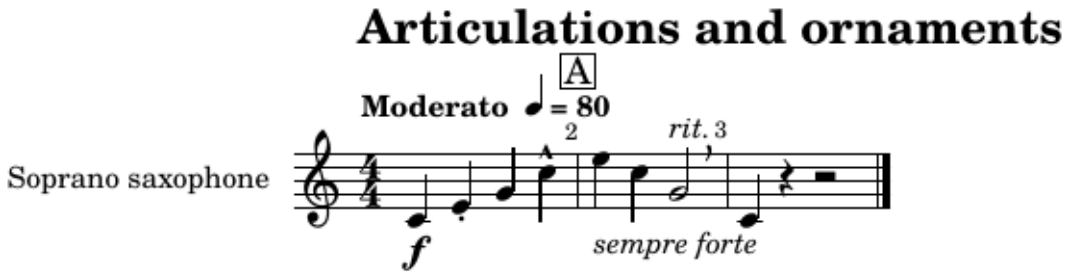
```
<!ELEMENT dynamics ((p | pp | ppp | pppp | ppppp | pppppp |
  f | ff | fff | ffff | fffff | ffffff | mp | mf | sf |
  sfp | sfpp | fp | rf | rfz | sfz | sffz | fz |
  n | pf | sfzp | other-dynamics)*)>
<!ATTLIST dynamics
    %print-style-align;
    %placement;
    %text-decoration;
    %enclosure;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

Other dynamics can also 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/>.
```

8.5 An example of articulations and dynamics

The reader can see various such in articulations/ArticulationsAndOrnaments.xml:

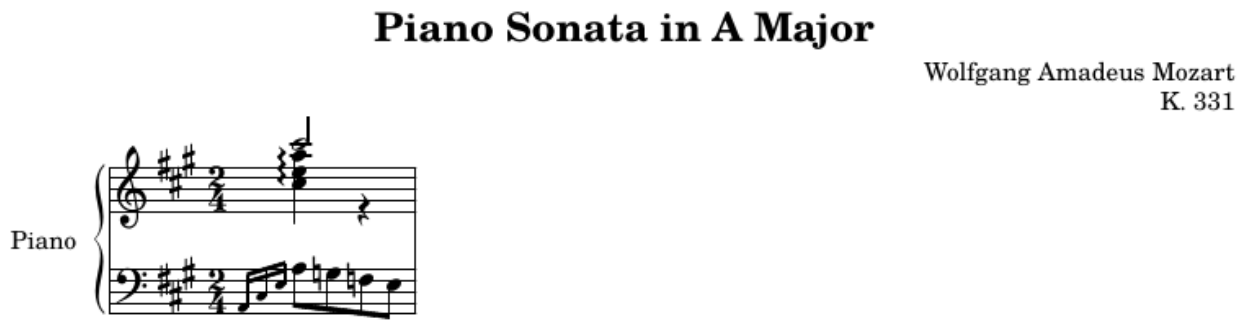


8.6 Grace notes

The '<grace>' element is defined in note.mod:

```
<!--
  The grace element indicates the presence of a grace note.
  The slash attribute for a grace note is yes for slashed
  eighth notes. The other grace note attributes come from
  MuseData sound suggestions. The steal-time-previous attribute
  indicates the percentage of time to steal from the previous
  note for the grace note. The steal-time-following attribute
  indicates the percentage of time to steal from the following
  note for the grace note, as for appoggiaturas. The make-time
  attribute indicates to make time, not steal time; the units
  are in real-time divisions for the grace note.
-->
<!ELEMENT grace EMPTY>
<!ATTLIST grace
    steal-time-previous CDATA #IMPLIED
    steal-time-following CDATA #IMPLIED
    make-time CDATA #IMPLIED
    slash %yes-no; #IMPLIED
>
```

For example, in gracenotes/LilyPondIssue34.xml, the three grace notes at the beginning of the lower staff are described by:



Listing 18: Grace notes example

```
      <note>
        <grace/>
        <pitch>
          <step>A</step>
          <octave>2</octave>
        </pitch>
        <voice>3</voice>
        <type>16th</type>
        <stem>up</stem>
        <staff>2</staff>
        <beam number="1">begin</beam>
        <beam number="2">begin</beam>
        <notations>
          <slur type="start" placement="above" number="1"/>
        </notations>
      </note>
      <note>
        <grace/>
        <pitch>
          <step>C</step>
          <alter>1</alter>
          <octave>3</octave>
```

```
        </pitch>
        <voice>3</voice>
        <type>16th</type>
        <stem>up</stem>
        <staff>2</staff>
        <beam number="1">continue</beam>
        <beam number="2">continue</beam>
      </note>
      <note>
        <grace/>
        <pitch>
          <step>E</step>
          <octave>3</octave>
        </pitch>
        <voice>3</voice>
        <type>16th</type>
        <stem>up</stem>
        <staff>2</staff>
        <beam number="1">end</beam>
        <beam number="2">end</beam>
      </note>
```

9 Ties

MusicXML makes the distinction between graphics and sound, and this applied to ties: the '`<tie>`' element indicates sound, and the '`<tied>`' element indicates notation. In a '`<tie>`', the '`start`' and '`stop`' values in the '`type`' attribute are used to indicate the beginning and end of the tie.

These two elements are defined in `note.mod`:

```
<!ELEMENT tie EMPTY>
<!ATTLIST tie
    type %start-stop; #REQUIRED
    %time-only;
```

```
<!ELEMENT tied EMPTY>
<!ATTLIST tied
    type %tied-type; #REQUIRED
    number %number-level; #IMPLIED
    %line-type;
    %dashed-formatting;
    %position;
    %placement;
    %orientation;
    %bezier;
    %color;
    %optional-unique-id;
>
```

10 Slurs

The '`<slur>`' element is placed inside a '`<notations>`' element, itself placed inside a '`<note>`' element. It is defined in `note.mod`:

```
<!ELEMENT slur EMPTY>
<!ATTLIST slur
    type %start-stop-continue; #REQUIRED
```

```
    number %number-level; "1"
    %line-type;
    %dashed-formatting;
    %position;
    %placement;
    %orientation;
    %bezier;
    %color;
    %optional-unique-id;
>
```

11 Tie and slur example

This example is in basic/TieAndSlur.xml:

Listing 19: Tie and slur example

```
      <note>
        <pitch>
          <step>A</step>
          <octave>3</octave>
        </pitch>
        <duration>2</duration>
        <type>quarter</type>
        <voice>1</voice>
      </note>
      <note>
        <pitch>
          <step>B</step>
          <alter>-1</alter>
          <octave>3</octave>
        </pitch>
        <duration>1</duration>
        <type>eighth</type>
        <voice>1</voice>
        <accidental>flat</accidental>
        <tie type="start" />
        <notations>
          <tied type="start" />
        </notations>
      </note>
      <note>
        <pitch>
          <step>B</step>
          <alter>-1</alter>
          <octave>3</octave>
        </pitch>
        <duration>1</duration>
        <type>eighth</type>
        <voice>1</voice>
        <accidental>flat</accidental>
        <tie type="stop" />
        <notations>
          <tied type="stop" />
```

```
          <slur number="1" type="start" />
        </notations>
      </note>
      <note>
        <pitch>
          <step>C</step>
          <octave>4</octave>
        </pitch>
        <duration>1</duration>
        <type>eighth</type>
        <voice>1</voice>
      </note>
      <note>
        <pitch>
          <step>B</step>
          <octave>3</octave>
        </pitch>
        <duration>1</duration>
        <type>eighth</type>
        <voice>1</voice>
        <notations>
          <slur number="1" type="stop" />
        </notations>
      </note>
```

```
%<!--
% The harmony elements are based on Humdrum's **harm
% encoding, extended to support chord symbols in popular
% music as well as functional harmony analysis in classical
% music.
%
% If there are alternate harmonies possible, this can be
% specified using multiple harmony elements differentiated
% by type. Explicit harmonies have all note present in the
% music; implied have some notes missing but implied;
% alternate represents alternate analyses.
%
% The harmony object may be used for analysis or for
% chord symbols. The print-object attribute controls
% whether or not anything is printed due to the harmony
% element. The print-frame attribute controls printing
% of a frame or fretboard diagram. The print-style entity
% sets the default for the harmony, but individual elements
% can override this with their own print-style values.
%
% A harmony element can contain many stacked chords (e.g.
% V of II). A sequence of harmony-chord entities is used
% for this type of secondary function, where V of II would
% be represented by a harmony-chord with a V function
% followed by a harmony-chord with a II function.
%-->
%<!ENTITY % harmony-chord "((root | function), kind,
% inversion?, bass?, degree*)">
%
%<!ELEMENT harmony ((%harmony-chord;)+, frame?,
% offset?, %editorial;, staff?)>
%<!ATTLIST harmony
%    type (explicit | implied | alternate) #IMPLIED
%    %print-object;
%    print-frame  %yes-no; #IMPLIED
%    %print-style;
```

```
%    %placement;
%    %optional-unique-id;
%>
%
```

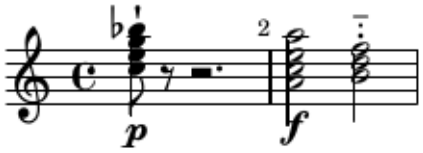
```
%<!--
% Figured bass elements take their position from the first
% regular note (not a grace note or chord note) that follows
% in score order. The optional duration element is used to
% indicate changes of figures under a note.
%
% Figures are ordered from top to bottom. A figure-number is
% a number. Values for prefix and suffix include plus and
% the accidental values sharp, flat, natural, double-sharp,
% flat-flat, and sharp-sharp. Suffixes include both symbols
% that come after the figure number and those that overstrike
% the figure number. The suffix values slash, back-slash, and
% vertical are used for slashed numbers indicating chromatic
% alteration. The orientation and display of the slash usually
% depends on the figure number. The prefix and suffix elements
% may contain additional values for symbols specific to
% particular figured bass styles. The value of parentheses
% is "no" if not present.
%-->
%<!ELEMENT figured-bass (figure+, duration?, %editorial;)>
%<!ATTLIST figured-bass
%    %print-style;
%    %printout;
%    parentheses %yes-no; #IMPLIED
%    %optional-unique-id;
%>
%<!ELEMENT figure
% (prefix?, figure-number?, suffix?, extend?, %editorial;)>
%<!ELEMENT prefix (#PCDATA)>
%<!ATTLIST prefix
%    %print-style;
%>
%<!ELEMENT figure-number (#PCDATA)>
%<!ATTLIST figure-number
%    %print-style;
%>
%<!ELEMENT suffix (#PCDATA)>
%<!ATTLIST suffix
%    %print-style;
%>
%
```

12 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 *only upon its second note*.

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

Chords



Listing 20: 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>
```

13 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:

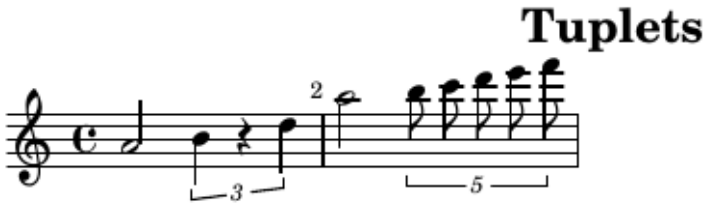
The contents, i.e. the notes in the tuplet, are not nested in the latter: there are placed in sequence between the two '`<tuplet>`' elements that delimitate the tuplet.

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 21: 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>
```

```
41            <normal-notes>2</normal-notes>
42          </time-modification>
43          <notations>
44            <tuplet number="1" type="stop" />
45          </notations>
46        </note>
```

14 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.

14.1 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'.

Listing 22: Existing bar styles

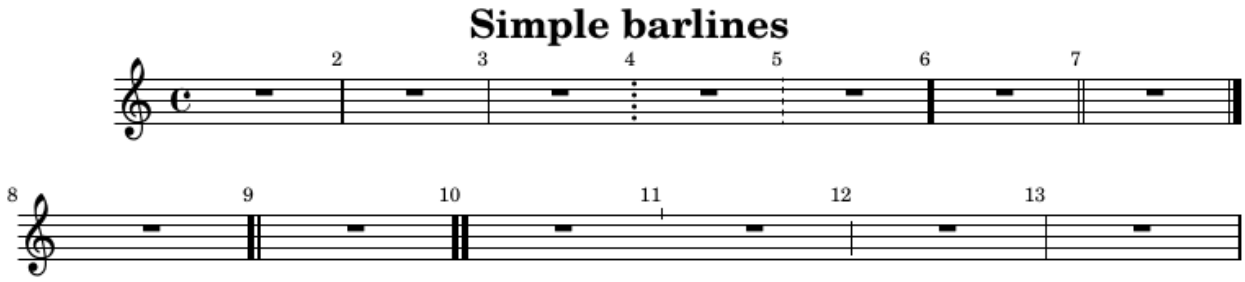
```
1  %<!--
2  % Bar-style contains style information. Choices are
3  % regular, dotted, dashed, heavy, light-light,
4  % light-heavy, heavy-light, heavy-heavy, tick (a
5  % short stroke through the top line), short (a partial
6  % barLine between the 2nd and 4th lines), and none.
7  %-->
8  %<!ELEMENT bar-style (#PCDATA)>
9  %<!ATTLIST bar-style
10 %   %color;
11 %>
12 %
```

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 23: Final barLine

```
1         <barLine location="right">
2           <bar-style>light-heavy</bar-style>
3           </barLine>
```

One can see the various simple barLines in barLines/SimpleBarLines.xml:



14.2 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.

Listing 24: Repeats barLines

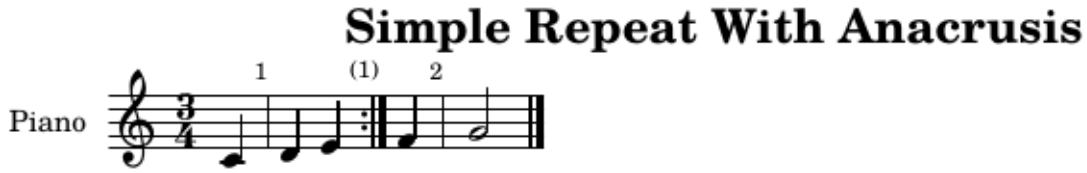
```
%<!--
% Repeat marks. The start of the repeat has a forward direction
% while the end of the repeat has a backward direction. Backward
% repeats that are not part of an ending can use the times
% attribute to indicate the number of times the repeated section
% is played. The winged attribute indicates whether the repeat
% has winged extensions that appear above and below the barLine.
% 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.
%-->
%<!ELEMENT repeat EMPTY>
%<!ATTLIST repeat
%    direction (backward | forward) #REQUIRED
%    times CDATA #IMPLIED
%    winged (none | straight | curved |
%   double-straight | double-curved) #IMPLIED
%>
%
```

```
      <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>
```

14.3 A repeat example

Here is a simple example in repeats/SimpleRepeatWithAnacrusis.xml:




```
<!-- ... ... ... ... ...-->
    <measure number="0" implicit="yes" width="144.60">
      <!-- ... ... ... ... ...-->
      <attributes>
        <divisions>1</divisions>
        <!-- ... ... ... ... ...-->
        </attributes>
      <note default-x="73.07" default-y="-50.00">
        <pitch>
          <step>C</step>
          <octave>4</octave>
          </pitch>
        <duration>1</duration>
        <voice>1</voice>
        <type>quarter</type>
        <stem>down</stem>
        </note>
      </measure>
<!--=========================================================-->
    <measure number="1" width="162.29">
      <note default-x="10.00" default-y="-45.00">
        <pitch>
          <step>D</step>
          <octave>4</octave>
          </pitch>
        <duration>1</duration>
        <voice>1</voice>
        <type>quarter</type>
        <stem>up</stem>
        </note>
      <note default-x="76.96" default-y="-40.00">
        <pitch>
          <step>E</step>
          <octave>4</octave>
          </pitch>
        <duration>1</duration>
        <voice>1</voice>
        <type>quarter</type>
        <stem>up</stem>
        </note>
      <barLine location="right">
        <bar-style>light-heavy</bar-style>
        <repeat direction="backward"/>
        </barLine>
      </measure>
<!--=========================================================-->
    <measure number="2" width="96.76">
      <note default-x="10.00" default-y="-35.00">
        <pitch>
          <step>F</step>
          <octave>4</octave>
```

```
          </pitch>
        <duration>1</duration>
        <voice>1</voice>
        <type>quarter</type>
        <stem>up</stem>
        </note>
      </measure>
<!--=========================================================-->
    <measure number="3" width="143.60">
      <note default-x="10.00" default-y="-30.00">
        <pitch>
          <step>G</step>
          <octave>4</octave>
          </pitch>
        <duration>2</duration>
        <voice>1</voice>
        <type>half</type>
        <stem>up</stem>
        </note>
      <barLine location="right">
        <bar-style>light-heavy</bar-style>
        </barLine>
      </measure>
```

15 Lyrics

15.1 The '<lyric>' element

In MusicXML the '<lyrics>' elements are sub-elements of the '<note>' elements. The definition is in note.mod:

```
<!ELEMENT lyric
  ((((syllabic?, text),
     (elision?, syllabic?, text)*, extend?) |
     extend | laughing | humming),
    end-line?, end-paragraph?, %editorial;)>
<!ATTLIST lyric
    number NMTOKEN #IMPLIED
    name CDATA #IMPLIED
    %justify;
    %position;
    %placement;
    %color;
    %print-object;
    %time-only;
    %optional-unique-id;
>
```

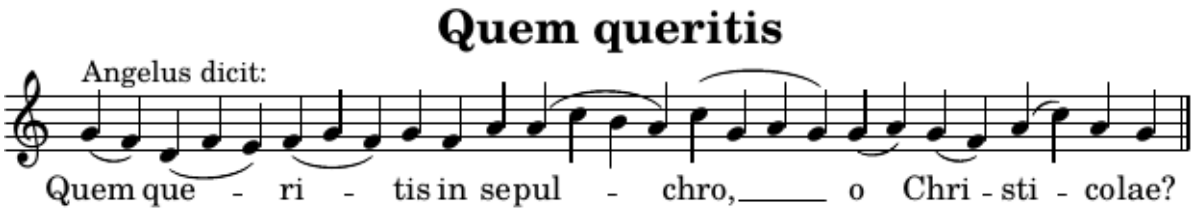
In lyrics:

- word extensions are represented using the '<extend>' element;
- hyphenation is indicated by the '<syllabic>' element, which can be '<single>', '<begin>', '<end>', or '<middle>'. These represent single-syllable words, word-beginning syllables, word-ending syllables, and mid-word syllables, respectively;
- multiple syllables on a single note are separated by '<elision>' elements. A hyphen in the text element should only be used for an actual hyphenated word;
- two text elements that are not separated by an '<elision>' element are part of the same syllable, but may have different text formatting.

The '<text>' sub-element contains the text to be sung. It can have attributes controlling the way it is displayed:

```
<!ELEMENT text (#PCDATA)>
<!ATTLIST text
    %font;
    %color;
    %text-decoration;
    %text-rotation;
    %letter-spacing;
    xml:lang NMTOKEN #IMPLIED
    %text-direction;
>
<!ELEMENT syllabic (#PCDATA)>
```

For example, the first note of lyrics/QuemQueritis.xml contains the single word 'Quem':

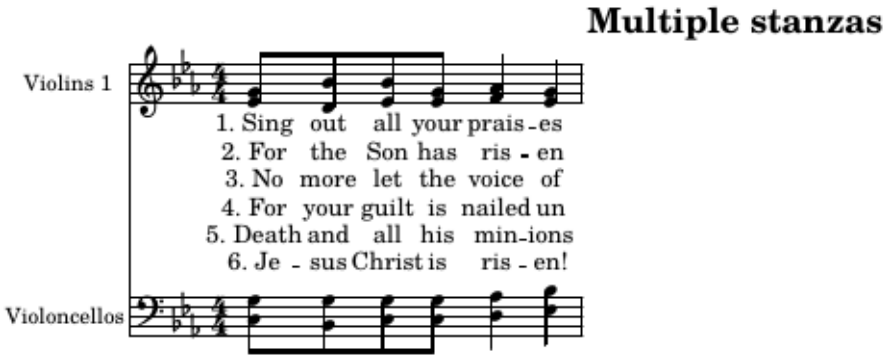



```
      <note>
        <pitch>
          <step>G</step>
          <octave>4</octave>
        </pitch>
        <duration>2</duration>
        <voice>1</voice>
        <type>quarter</type>
        <stem>up</stem>
        <notations>
          <slur type="start" number="1"/>
        </notations>
        <lyric number="1">
          <syllabic>single</syllabic>
          <text>Quem</text>
        </lyric>
      </note>
```

15.2 Stanzas

Stanzas are not represented in MusicXML per se, but implicitly: the 'number' attribute of the '<lyric>' element is used to specify the stanza number.

For example, in lyrics/MultipleStanzas.xml, the first note ($E^{\flat}$) of the first chord in the upper staff contains the lyrics for the first syllable of all the successive stanzas, preceded by the stanza number and a dot:



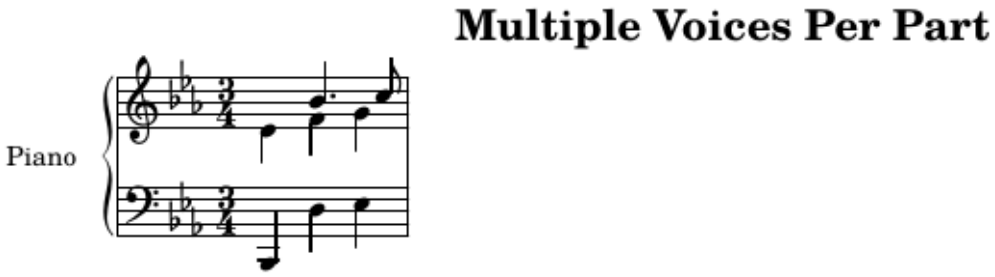
Listing 25: Multiple stanzas example

```
      <note default-x="129.06" default-y="-40.00">
        <pitch>
          <step>E</step>
          <alter>-1</alter>
          <octave>4</octave>
          </pitch>
        <duration>1</duration>
        <voice>1</voice>
        <type>eighth</type>
        <stem>up</stem>
        <beam number="1">begin</beam>
        <lyric number="1">
          <syllabic>single</syllabic>
          <text>1. Sing</text>
          </lyric>
        <lyric number="2">
          <syllabic>single</syllabic>
          <text>2. For</text>
          </lyric>
        <lyric number="3">
          <syllabic>single</syllabic>
          <text>3. No</text>
          </lyric>
        <lyric number="4">
          <syllabic>single</syllabic>
          <text>4. For</text>
          </lyric>
        <lyric number="5">
          <syllabic>single</syllabic>
          <text>5. Death</text>
          </lyric>
        <lyric number="6">
          <syllabic>begin</syllabic>
          <text>6. Je</text>
          </lyric>
        </note>
      <note default-x="129.06" default-y="-30.00">
        <chord/>
        <pitch>
          <step>G</step>
          <octave>4</octave>
          </pitch>
        <duration>1</duration>
        <voice>1</voice>
        <type>eighth</type>
        <stem>up</stem>
```

```
47          </note>
```

16 Multiple voices

Let's look in some detail at the score specified in multistaff/MultipleVoicesPerPart.xml:



The first voice in upper staff '1' has number '1'. The '<forward>' element is used because there no note in this voice upon the first beat, whose duration is '96' divisions. This element allows drawing to continue a bit further in the voice, without drawing rests in-between.:

```
1       <forward>
2         <duration>96</duration>
3         <voice>1</voice>
4         <staff>1</staff>
5       </forward>
6       <note default-x="154">
7         <pitch>
8           <step>B</step>
9           <alter>-1</alter>
10          <octave>4</octave>
11        </pitch>
12        <duration>144</duration>
13        <voice>1</voice>
14        <type>quarter</type>
15        <dot/>
16        <stem default-y="15.5">up</stem>
17        <staff>1</staff>
18      </note>
19      <note default-x="225">
20        <pitch>
21          <step>C</step>
22          <octave>5</octave>
23        </pitch>
24        <duration>48</duration>
25        <voice>1</voice>
26        <type>eighth</type>
27        <stem default-y="18">up</stem>
28        <staff>1</staff>
29      </note>
```

The notes in voice '2' in staff '1' can now be described, but only after a '<backup>' element that places the "drawing position" back to the beginning of the measure:

```
1       <backup>
2         <duration>288</duration>
3       </backup>
4       <note default-x="108">
5         <pitch>
6           <step>D</step>
7           <octave>4</octave>
```

```
        </pitch>
        <duration>96</duration>
        <voice>2</voice>
        <type>quarter</type>
        <stem default-y="0.5">up</stem>
        <staff>1</staff>
      </note>
      <note default-x="154">
        <pitch>
          <step>F</step>
          <octave>4</octave>
        </pitch>
        <duration>96</duration>
        <voice>2</voice>
        <type>quarter</type>
        <stem default-y="-63">down</stem>
        <staff>1</staff>
      </note>
      <note default-x="201">
        <pitch>
          <step>G</step>
          <octave>4</octave>
        </pitch>
        <duration>96</duration>
        <voice>2</voice>
        <type>quarter</type>
        <stem default-y="-60.5">down</stem>
        <staff>1</staff>
      </note>
```

Then comes the specification of voice '3' in staff '2', again after a '<backup>' element to place the drawing position at the beginning of the measure:

```
      <backup>
        <duration>288</duration>
      </backup>
      <note default-x="108">
        <pitch>
          <step>B</step>
          <alter>-1</alter>
          <octave>1</octave>
        </pitch>
        <duration>96</duration>
        <voice>3</voice>
        <type>quarter</type>
        <stem default-y="5.5">up</stem>
        <staff>2</staff>
      </note>
      <note default-x="154">
        <pitch>
          <step>D</step>
          <octave>3</octave>
        </pitch>
        <duration>96</duration>
        <voice>3</voice>
        <type>quarter</type>
        <stem default-y="-55.5">down</stem>
        <staff>2</staff>
      </note>
      <note default-x="201">
        <pitch>
          <step>E</step>
          <alter>-1</alter>
          <octave>3</octave>
        </pitch>
        <duration>96</duration>
        <voice>3</voice>
        <type>quarter</type>
        <stem default-y="-50.5">down</stem>
        <staff>2</staff>
      </note>
```

17 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™;
- 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, of course, the scanning application should create quality MusicXML data.

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

```
      <attributes>
       <key>
         <fifths>-1</fifths>
         </key>
       </attributes>
```

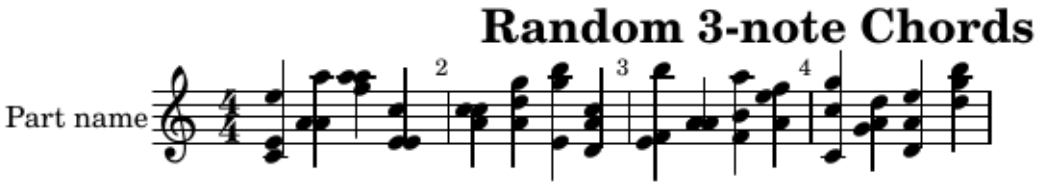
with C++ code such as:

Listing 26: 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);
```

Here is a score containing random 3-note chords created by `libmusicxml/samples/RandomChords.cpp`, a C++ small program provided as a example of using `libmusicxml2` to create MusicXML data:



18 Importing MusicXML data

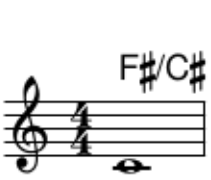
Many GUI applications provide a way to import MusicXML data, often with some limitations. We show some of them below.

It is worth noting that MuseScore 3.3.4 does a good job at issuing warnings if the MusicXML data is not well-formed according to the DTD.

18.1 Small element, big effect

The '`<harmony>`' element can contain an '`<inversion>`' sub-element to indicate the number of the chord inversion. Some applications ignore this element when importing MusicXML data, because it takes full knowledge of chords structures to compute the bass note of inverted chords.

Here is how `xml2ly` handlles the second inversion of the chord in `harmonies/Inversion.xml`:



Listing 27: Harmony inversion

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

18.2 Elements handled in different ways

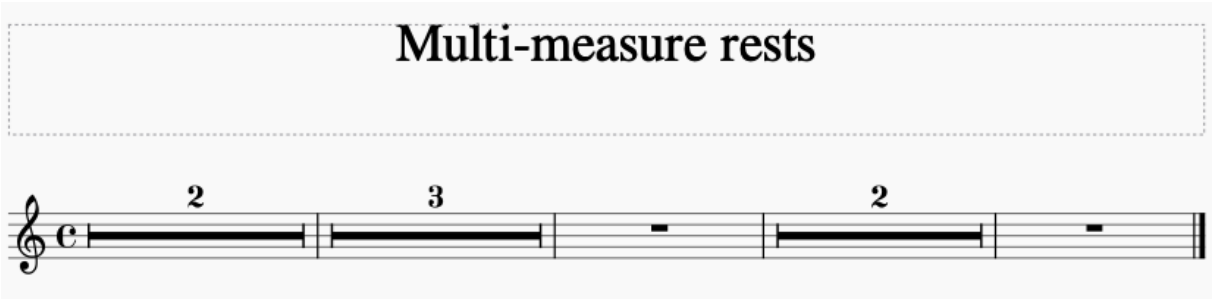
Multi-measure rests are specified in MusicXML with the '`<multiple-rest>`' element. All measures in the sequence have to be explicitly present in the MusicXML data.

For example, the first two measures of `rests/MultiMeasureRests.xml` are a multi-measure rest, described by:

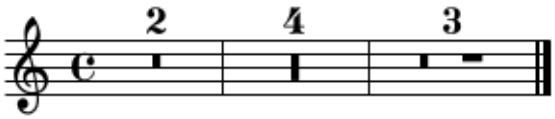
```
  <part id="P1">
    <measure number="1">
      <attributes>
        <divisions>1</divisions>
        <key>
          <fifths>0</fifths>
          <mode>major</mode>
        </key>
        <time symbol="common">
          <beats>4</beats>
          <beat-type>4</beat-type>
        </time>
        <clef>
          <sign>G</sign>
          <line>2</line>
        </clef>
        <measure-style>
          <multiple-rest>2</multiple-rest>
        </measure-style>
      </attributes>
      <note>
        <rest/>
        <duration>4</duration>
        <voice>1</voice>
      </note>
    </measure>
    <!--========================================================-->
    <measure number="2">
      <note>
        <rest/>
        <duration>4</duration>
        <voice>1</voice>
      </note>
    </measure>
    <!--========================================================-->
```

This file is handled differently by various applications, as can be seen below.

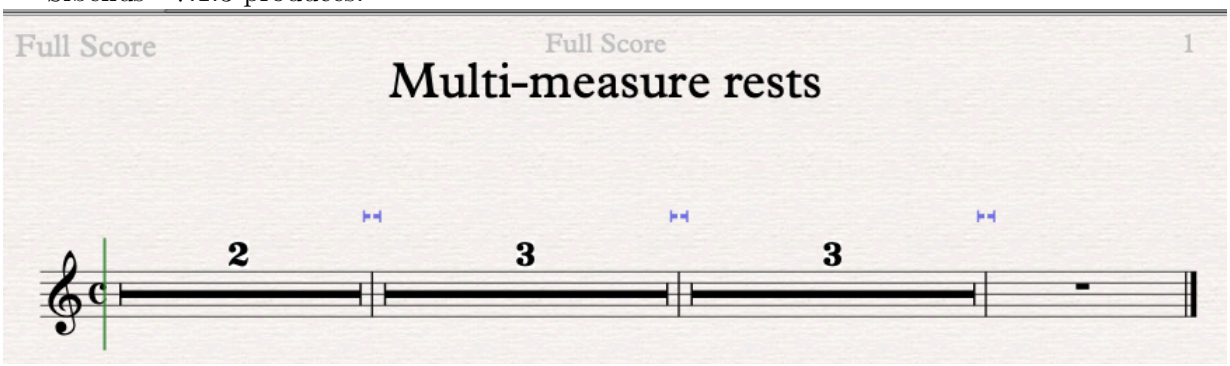
MuseScore 3.3.4 displays it this way:



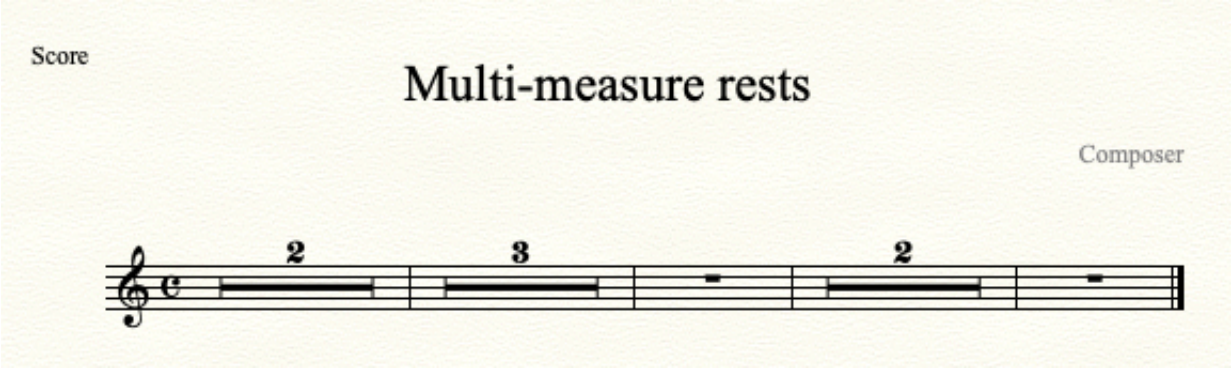
`musicxml2ly` produces:



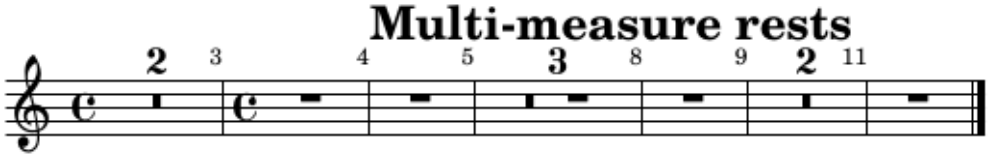
Sibelius™ 7.1.3 produces:



Finale™ 2014 produces:



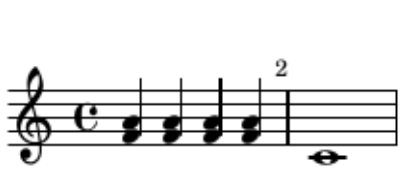
`xml2ly` is still experimental, and currently produces:



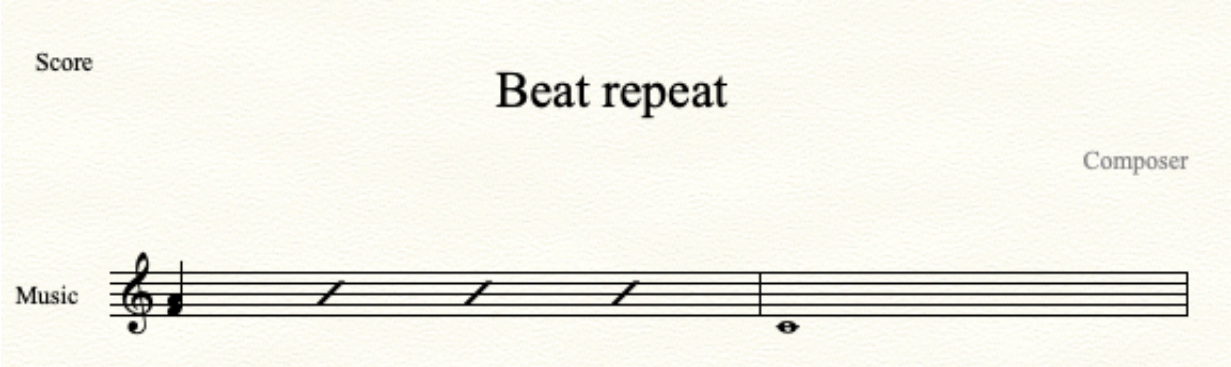
18.3 Elements often not well handled

There are elements that are not displayed in a "standard" way by the usual music score editors. One of them is the '<beat-repeat>', found for example in repeats/BeatRepeat.xml.

MuseScore 3.3.4, musicxml2ly, xml2ly and Sibelius™ 7.1.3 produce the following, i.e. they ignore the beat repeat altogether:



Finale™ 2014 produces:



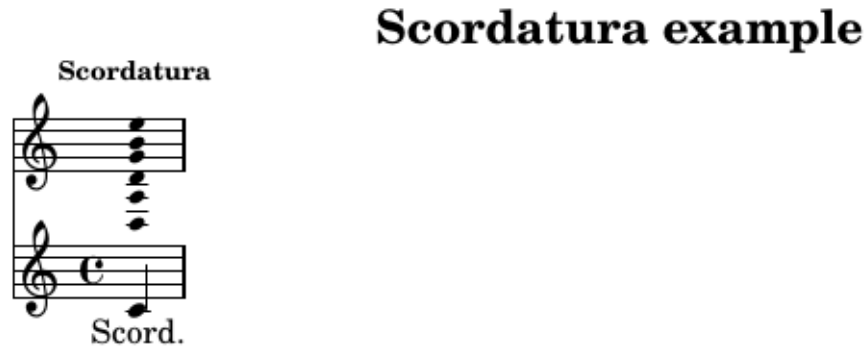
.

And if one exports that score from Finale™ 2014 to MusicXML, the beat repeat information is lost, see repeats/BeatRepeatExportedFromFinale.xml.

18.4 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 28: 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>
```

18.5 A real challenge

The file challenging/BeethovenNinthSymphony.xml is over 66 megabytes large – it contains the whole score for this symphony.

The interested reader is urged to try and import this file into 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 3.3.4 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 the LilyPond code generated show that this converter is still experimental...

19 Conclusion

MusicXML supports other score elements such as harmonies and figured bass, as well as nested repeats. 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/musicxml.

MusicXML has become a de facto standard for music scores data interchange between applications. The way it is exported and imported by the various applications is quite diverse though, 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 in a more concise way:

Listing 29: 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.