Updated January 30, 2006
[Thomas Weber was so kind to investigate positioning attributes defined by MusicXML and applied by the current Dolet for Finale. Here follows the exchange].

Thomas,
Thanks for working this out. Some of this I can use in the next major update of SipXML2Score. I have marked the thoughts in this text. Let me remind you of the update policy that I apply:

- Program errors reported by users on licensed utilities will be fixed with the highest priority – in practice this means same day service. If a workaround is possible, the priority is lowered (For example, one utility does not work on W2000 but the user also has an XP so that is considered a workaround. He has been waiting since October 2005…). Such issues are handled when I have nothing to do.
- Program errors reported by users on free utilities get the next lower priority. This is the story of the gift horse.
- Feature enhancements on licensed utilities take priority number 3, on free utilities I may do it or I may not. The latter has never happened but I keep this as an option.
- Sometimes I get requests for a new utility. An example is SipText for Cyrillic and I consider it with priority if the customer commits to buying it.
- There is also a random factor in that sometimes I do things because I feel like doing them. Right now I work on two things: SipSplit (a new part extractor replacing the DOS version) and the rewrite of the experimental SipScore2XML as I have picked up some signals that this will become a future need.

If you're interested in the mathematics of MusicXML and Score, read on.
If not, beware!

Jan de Kloe schrieb:

>However, most of this information is unusable in the conversion to Score.
>In this example, the x value in <note default-x="12"> serves no useful
>purpose.
>
I'm now sure that it's the offset from the preceding barline. How large the offset really is can be derived from the "defaults" section. (I now refer to the Beethoven "An die ferne Geliebte" example that can be found in the xmlsamples.zip file that can be downloaded from musicxml.org.) Here are the first lines from the example file:

```
<defaults>
  <scaling>
    <millimeters>6.35</millimeters>
    <tenths>40</tenths>
  </scaling>
  <page-layout>
    <page-height>1760</page-height>
    <page-width>1360</page-width>
    <page-margins type="both">
      <left-margin>80</left-margin>
      <right-margin>80</right-margin>
      <top-margin>80</top-margin>
      <bottom-margin>80</bottom-margin>
    </page-margins>
    [...]
</defaults>
```

The scaling tag is the important thing here. This tag is not very intuitive to understand. It does NOT mean, one internal unit corresponds with 6.35 millimeters, it means one millimeter corresponds with 6.35 internal units. The page width in this example is 1360 units, that means 1360 / 6.35 = 214.17 (Adobe Reader says 215.9 mm for the pdf version included in xmlsamples.zip). The length of a line of music is the width minus the margins (1360 - 2*80 = 1200). If you add the widths of the measures of one line you always get about 1200 (sometimes not exactly, probably due to rounding errors). In the first line the sum is 1050, adding the indent of 150 given in the first lines makes 1200.

```
<measure number="1" width="324">
  <print>
    <system-layout>
      <system-margins>
        <left-margin>150</left-margin>
        <right-margin>0</right-margin>
      </system-margins>
      <top-system-distance>300</top-system-distance>
    </system-layout>
  </print>
  [...]
</measure>
```

Suggested program change:
When the <scaling> and <page-layout> commands are in the source (not all XML generators generate these) and the user has selected the option (to be provided) to honor XML positioning information (I must make this a user option as users may prefer private settings), then the width of the margin will be derived from this, as well as the width of the staff if this is not too complicated.

The first note in this example (which is actually a rest) has an x value of 136:

```
<note default-x="136">
  <rest/>
  <duration>24</duration>
  <voice>1</voice>
  <type>quarter</type>
</note>
```

The exact horizontal distance from the left border of the page is 80 + 150 + 136 = 366 (left margin + indent + x position). In millimeters this is 57.63.

In this example the length of a line is 1200, which corresponds with 200 units in Score, therefore all values have to be divided by 6 to translate them into Score units. The left margin of 80 units must be subtracted. The recipe for translating the positions of a line of music is the following:

- Add all widths of measures and indents in the line (1200 here).
Calculate the "scaling factor" by dividing that value by 200 (you get 6 here).
- All x values are relative to a reference value that increases when entering a new measure. First we set it to 0. As the first line has an indent of 80, we add 80 for the first bar. (For the second measure we add 324, the width of the first measure, and get 404. But let's stay with the first measure (with reference value 80) for now.)
- The first note (rest) in the first line has an x value of 136, that means its position is at 80 + 136 = 216. Divided with 6 we get 36, and that's the p3 value in Score.

>The LJ command needs to be executed

>anyway on a conversion result and I haven't seen anything better than that.
>
You're certainly right that anyone who converts MusicXML to Score will rejustify the music in Score, so there is in fact no real need for exact translation of positioning in this case.

Suggested program change:
None. The mechanism I have works well for XML input which has or does not have default-x values for notes/rests. There is no point to use the information as it significantly complicates the program (thousands of lines of code). It would make more sense to include the LJ-algorithm but I do not want to reinvent the wheel. Tom Brodhead did exactly that in his justification program.
It is beyond doubt that if the first implementation of Dolet would have had these MusicXML 1.1 positioning features, I would have used it because basically it is easier than the way I had to resort to.

>When I get a <stem default-y="-45">down</stem> to convert this -45 is
>ignored because if would result in irregular output.
>
The system for the stem length is totally different from what we know from Score. The values are not in scale steps, as would be reasonable, but in the units also used for horizontal positioning. The reference y value is not the default endpoint (seven scale steps from the middle of the notehead) but the top staff line. So the conversion is much more difficult in this case.

In this example the height of a stave (distance from bottom to top line) is 40 units. One can see that e.g. from the first note in measure 3, an e", for which the stem ends at the bottom line:

```
<stem default-y="-40">down</stem>
```

The stave height must be in proportion with the music font size, which is according to line 45 of the file 18 points:

```
<music-font font-family="Maestro" font-size="18"/>
```

Point size 18 means the font height is 18/72". In millimeters this is 18/72" * 25.4 = 6.35. By the way, this is exactly the value that was given as scaling factor between internal points and millimeters, only that 6.35 millimeters do not make one unit but vice versa. Who can solve this riddle? The other way round makes more sense to me.
However, to get internal units we must multiply with 6.35, so this makes
6.35 * 6.35 = 40.3225 internal units. This is pretty close to the 40 units stave height we derived from the stem position above. So, how can this finally be translated into Score's code 1 p8?

First of all it is important to know, how to convert units into scale steps. A stave is 8 scale steps high, in MusicXML units it is
  (points / 72) * 25.4 * scaling factor.
That makes
  (points / 72) * 25.4 * scaling factor / 8 units, per scale step, or to simply this
  points * scaling factor * 127/2880

(Remark: "points" refers to the point size of the music font.)

Let's assume we already have the p4 value of the note. The calculation looks as follows for upstemmed notes:
p8 = p4 - y / (points * scaling factor * 127/2880) - 4 and for downstemmed notes:
p8 = p4 - y / (points * scaling factor * 127/2880) - 18.

Here a test: The first "real note" in measure is a b' (p4 = 7) and has the following stem information:

```
<stem default-y="-55">down</stem>
```

p8 would be 7 - (-55) / (18 * 6.35 * 127/2880) - 18 = -0.08798...

This is roughly 0, the stem length is as usual.

Suggested program change:
None. The generation of stem length is working to satisfaction in the converter.

>Where we NEED the x/y coordinates is on text. So far I have not figured
>out how this works. Either because what is generated by Dolet is
>erroneous or I misinterpret the definition. If anyone can suggest how I
>should use those coordinates to influence Par3 and Par4 in a conversion
>and supply testing material I will take this on with priority.
>
For text conversion probably not only the x/y coordinates need to be translated but also the point size. Score's point size for text is always dependent on the staff size. So first of all, let's find out how to translate the staff size accurately. As we have seen, the height of a staff is equal to the point size of the used music font which has to be multiplied with 1/72 * 25.4 to get the height in mm. In Score, the height of a staff is p6 * 8.89. The following "translates" the point size into a p6 value:
p6 = points * 1/72 *25.4 / 8.89
or simplified:
p6 = points * 5/126
As the proportions line width / staff height are to be kept, we are not finished yet. Score's width is 190.5, MusicXML's width is width (in units) / scaling factor. This results in the following formula:
p6 = points * 5/126 * 190.5 / (XML-width / scaling factor) and simplified:
p6 = points * (scaling factor / XML-width) * 635/84

Now we can calculate the correct p6 for the text item using Tom's formula:
p6 = ((TextPoints / 13.885955) / (StavePoints * (scaling factor /
XML-width) * 635/84)
simplified:
p6 = (TextPoints * XML-width) / (104.9712 * scaling factor * StavePoints)

Suggested program change:
This is usable information and will result in a major improvement on text placement.

And there is one more hurdle for positioning the text. In this example:

```
    <credit-words default-x="680" default-y="1678" font-size="24"
```

```
justify="center" valign="top">An die ferne Geliebte </credit-words>
```

the text is to be centered at x=680 (which is (680 - 80) / 6 = 100 in Score, which does not surprise us). This requires knowledge of the font metrics and the overall length of the line - I am not going to work that out here.

Suggested program change:
The converter knows the font metrics and uses these to avoid text overlapping, centering, left and right justification. What will be adapted is the automatic setting of lyrics height which subject came up in our earlier exchange.

The above was hard enough. Does anybody feel like examining all this for mistakes? Not that Jan includes faulty calculations in his future versions of SipXML2Score!

Good night!

Thomas Weber

Added on January 30, 2006:

Daniel Walsh schrieb:

> Hey Jan!
>
> <The discussion about the details of converting MusicXML 1.1
> positioning attributes will only interest a small segment of this
> list.>
>
> That is a reason for leaving it here!!
>
> Please do not move it. Merely copy it.
> It will be easier to find it here and there -- in both places.
>
> This list is for discussions like this. Please help keep the list alive.
>
> Daniel.

I agree. What we discuss here is not only relevant for (potential) SipXML2Score users (I am NO SipXML2Score user), it's about a music exchange format that is likely to become the future standard for music typesetting programs. And this certainly is of interest for a much larger group of Score users.

A transparent file format is what makes Score so powerful. While other companies keep their file format definitions secret, in Score we know exactly what is possible with Score files, and Score allows us to change virtually every byte by hand (by editing parameters directly in Score or editing a pmx file "out of Score"). MusicXML basically offers the same flexibility because you do not depend on the music editor which may not allow you everything that MusicXML may allow you. To make changes to a file, you can do it by hand (just like with Score) or use any MusicXML ready program you like. As we have seen, the MusicXML format does not yet offer all the fine tuning features that the Score format offers, but it's only version 1.1, there are certainly many to follow. And what Frank suggested ("E.g. using Sibelius for part extraction, finding page turns and making a rough layout and exporting that datas to Score to do the fine tuning.") could really be the future for an efficient music engraving workflow, couldn't it? In the Score world we already have a number of specialized programs that operate on the Score format, which is another strong point of Score. MusicXML could open a whole new universe of tools.

And, last but not least, MusicXML could be a blessing for Score's future. If it has reached a stage which allows really precise positioning of all elements and handling of all the special situations that the upper class programs can handle, then publishers will maybe accept MusicXML files as standard instead of insisting on Sibelius or Finale files (which, concluded from some mailings on this list, appears to be more and more the case), so they would not bother whether you are using Sibelius, Finale, Score or whatever program. This could even result in generally better quality of computer engraved music published in print. The engraver has the choice to use that program(s) that are best suited for a certain kind of music, that are most efficient, that offer the best results etc. Therefore the concurrence between music typesetting programs should increase.

I am no professional, so what do the professionals say? Is this wishful thinking? There certainly are things that MusicXML will never be able to handle like Score, such as all the PostScript features (_99-Text and code 15 items) or the individual changes to the symbol library using draw. But this is a feature of the Score program, not of the file format, which must not be confused. MusicXML allows explicit declaration of different music fonts, so in this point it is even a bit more precise than a Score file which, handed on to another Score user, includes no information about the actual shape of the symbols.

And we may not overlook the potential risks of MusicXML. Recordare, the company that introduced and develops MusicXML, indisputably has its own economical interests. Unlike the W3C (which is responsible for html and general XML) it is not a more or less neutral consortium. They may not include things in the MusicXML standards that they don't see a need for while others do. This may result in the introduction of "proprietary" elements like in the past of html. Some browsers (namely Netscape) introduced their own tags for handling multimedia content which was not possible before. Microsoft in return introduced its own tags that were incompatible with Netscape's. The MS Internet Explorer e.g. is the only browser that supports ActiveX-tags that have been introduced by Microsoft which requires Microsoft's ActiveX technology and therefore MS Windows. A similar development in MusicXML would wreck all its gains.
An exchange format that can only be interpreted correctly by a single application is not an exchange format any more.

Thomas Weber