MusicXML: An Internet-Friendly Format for Sheet Music

Michael **Good**

Abstract

The downloadable sheet music market has been constrained by reliance on proprietary binary formats. MusicXML is a new Internet-friendly format for representing sheet music and music notation. We introduce the basics of MusicXML design, using examples from current software to illustrate MusicXML's advantages over previous music interchange formats. We conclude by discussing the new types of music use that MusicXML makes possible, and how this fits in with ongoing XML development efforts.

1. The Need for a New Music Interchange Format

Common Western music notation is a symbolic method of representing music for performers and listeners. Besides its use in publishing sheet music, musical scores and parts, it has been encoded in different computer formats for over 30 years. The book *Beyond MIDI* describes over 20 of these published musical codes. There are also many more unpublished, proprietary codes. Given the high costs of traditional music publication, many companies have seen that Internet distribution has the potential to increase both the size and profitability of the sheet music market.

To date, the Internet sheet music market has been hobbled by its reliance on a proliferation of proprietary binary formats. The most common format, Portable Document Format (PDF) , contains no musical semantics and can only be viewed on screen or printed on paper. Companies like Sunhawk, MusicNotes, Sibelius, and Noteheads all have different proprietary music formats for their sheet music players. If you buy sheet music from them or their partners today, you can play it, view it, or print it only with that single proprietary player. This provides little value-added to consumers when compared to printed music on paper. We believe this has fragmented the online sheet music market and contributed significantly to its disappointing sales through 2001.

Electronic musical instruments such as synthesizers faced a similar problem in the 1980s, when there was no way to get musical instruments from different vendors to work or play together. The invention of the Musical Instrument Digital Interface (MIDI) format [MIDI 1996]solved this problem, and led to the rapid growth of the electronic musical instrument market. The introduction of General MIDI led to even further levels of compatibility, interchange, and growth in the instrument market.

Today, MIDI remains the only symbolic music interchange format in wide use today. But MIDI, designed to solve problems in music performance, cannot represent much of what is found in sheet music. MP3 and other audio formats represent music recordings, not music notation.

Except for very simple music, computers cannot automatically derive accurate music notation from a music recording, despite decades of research.

Music interchange formats have been developed in the past, but none besides MIDI has been successful. Notation Interchange File Format (NIFF) is based on the binary Resource Interchange File Format (RIFF) format. It has been used to interchange music between scanning and notation applications. NIFF contains more notation data than MIDI, but its highly graphical representation is inferior to MIDI for performance and analysis applications. Its adoption outside of scanning software has been very limited. Standard Music Description Language (SMDL) , based on Standard Generalized Markup Language (SGML), was an attempt to create a general-purpose, formal specification for music, and was designed without the guidance of implementation experience. The result was a complex, difficult-to-understand specification that, to our knowledge, has not been implemented in any commercial product.

2. MusicXML's Approach to Music Interchange

MusicXML attempts to do for online sheet music and music software what MIDI did for electronic musical instruments. MusicXML represents common Western music notation from the 17th century onwards. By using XML, it is more Internet-friendly that proprietary binary formats. MusicXML serves as an interchange format for applications in music notation, music analysis, music information retrieval, and musical performance. Therefore it augments, but does not replace, existing specialized formats for individual applications. It is designed to be adequate, not optimal, for these diverse applications.

XML has obvious appeal as a technology to help solve the music interchange problem. It is designed to represent complex, structured data in a standardized way. The same things that make XML appeal in other application areas - including straightforward usability over the Internet, ease of creating documents, and human readability - apply to musical scores as well. Castan et al. [Castan 2001]discuss several approaches to using XML to represent music, and many more have been introduced on the Web. These alternative XML formats tend to be much simpler than MusicXML, do not represent as many aspects of music notation, and lack software that works with commercial music applications.

To circumvent the problems of past interchange formats, the design of MusicXML followed two main strategies:

1. The design of MusicXML was based on two of the most powerful academic music formats for music notation: MuseData and Humdrum. Both formats have large music repertoires available, and have been used for diverse music applications. A format that learns from these successes would have a solid technical grounding.

2. The MusicXML definition was developed iteratively with MusicXML software. The initial prototype software did two-way conversions to MuseData, read from NIFF files, and wrote to Standard MIDI Files (Format 1). Handling these three very different formats demonstrated that MusicXML's basic interchange capabilities were solid. We then moved on to support

interchange with Finale from Coda Music Technology, the market leader in music notation software.

Iterative design and evolutionary delivery techniques have been used since the 1980s to produce more usable and useful computer systems[Gilb 1988] [Good 1988].[Gould 1985]. With MusicXML, we have found that these techniques can also be successful in XML language design.

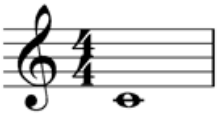
As of October 2001, MusicXML has just begun its public beta test. Recordare provides a MusicXML Finale converter to convert between Finale, MusicXML, and MuseData. The converter runs on Windows with Finale 2000, 2001, and 2002. Visiv has added MusicXML support to their SharpEye Music Reader product, one of the leading music scanner software packages on Windows. SharpEye saves MusicXML files starting with version 2.15. Xemus Software and Middle C Software have announced their plans for MusicXML support in upcoming products. MusicXML is available under a royalty-free license (http://www.recordare.com/dtds/license.html). The complete MusicXML DTD is available at http://www.musicxml.org/xml.html.

With this solid level of acceptance as a result of the MusicXML alpha test, we will be reaching out to more music software developers during the beta test to increase the level of MusicXML usage in the industry. Only after we have greater implementation experience, especially with popular music, do we plan to standardize MusicXML, most likely through Organization for the Advancement of Structured Information Standards (OASIS).

3. Elements of MusicXML Design

To introduce how MusicXML represents musical scores, here is the musical equivalent of C's "hello, world" program for MusicXML. Here we will create about the simplest music file we can make: one instrument, one measure, and one note, a whole note on middle C:

Figure 1. Hello, World in MusicXML



Here is the musical score represented in MusicXML:

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<!DOCTYPE score-partwise PUBLIC
    "-//Recordare//DTD MusicXML 0.5 Partwise//EN"
    "http://www.musicxml.org/dtds/partwise.dtd">
<score-partwise>
  <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>
```

MusicXML can represent scores either partwise (measures within parts) or timewise (parts within measures), with XSLT stylesheets to go back and forth between the two. One of the key insights from the Humdrum format is that musical scores are inherently two-dimensional. Since XML is hierarchical, using XSLT to alternate between the two hierarchies gives us a useful way to simulate the lattice-like structure of a musical score. In our example, we are using a partwise score. The part list is very simple, containing one instrument. The score-part element's id attribute is an ID that is referred to by the part element's id attribute, which is an IDREF.

The attributes element contains musical attributes of a score, such as the key signature, time signature, and clef. Normal key signatures are represented by the number of sharps or flats; the fifths element refers to the position of the key on the circle of fifths. The key of C with no sharps or flats has a value of 0; the key of F with one flat would have a value of -1. The time signature includes the numerator (beats) and denominator (beat-type). The representation of treble clef shows that the second line from the bottom of the staff represents a G.

The first child element of the attributes element, <divisions>, is borrowed from MIDI and MuseData. Musical durations are commonly referred to as fractions: half notes, quarter notes, eighth notes, and the like. While each musical note could have a fraction associated with it, MusicXML instead follows MIDI and MuseData by specifying the number of divisions per quarter note at the start of a musical part, and then specifying note durations in terms of these divisions.

After the attributes, we then have our one and only note in the score: a C in octave 4, the octave that starts with middle C. Since we have one division per quarter note, the duration of 4 indicates a length of 4 quarter notes, or one whole note. The actual printed note type is also included as well as the duration. Notation programs can more easily deal with the written type, while MIDI

programs deal more easily with the duration. In some cases, sounding duration is different than written duration (as in jazz), so having both can be necessary for accuracy, not just program convenience.

Contrast the musical representation of pitch and duration, well-suited for both notation and performance applications, to the graphical representation in NIFF. NIFF is a binary format, but if we do a one-to-one translation of its binary elements to XML syntax, the whole note would be represented like this:

```
<Notehead Code="note" Shape="2" StaffStep="-2">
  <Duration Numerator="1" Denominator="1"/>
</Notehead>
```

The StaffStep attribute tells us that the note is two staff steps, or one line, below the staff. What's its pitch? For that we need to check the clef and key signature, then handle any accidentals that preceded the note in this measure, then look for any accidentals in a note that may be tied to this one. All this computation is needed for one of the two most basic elements of music notation: what pitch is sounding? Fortunately, the other basic element, the timing of the note, is represented much more directly. But the very indirect nature of pitch representation makes NIFF sub-optimal for most performance and analysis applications.

To illustrate how MusicXML gives better results than MIDI for music interchange, let us look at a typical difference in real-life interchange. We scanned in the fourth song of Robert Schumann's *Liederkreis*, Op. 24, on poems by Heinrich Heine, using SharpEye Music Reader version 2.16. We corrected the scanning mistakes within SharpEye: music scanning is not yet as accurate as optical character recognition, especially when scanning older public domain music. We then saved the files from SharpEye to MIDI and MusicXML. We imported the MIDI files into Finale 2002 and Sibelius 1.4 using the default MIDI import settings, and imported the MusicXML file into Finale 2002 using the Recordare MusicXML Finale Converter Beta 1.

This song is not very complicated, so all of its musical features can be captured within SharpEye and saved to MusicXML. Figure 2 shows the last four measures of the song as scanned into SharpEye:

Figure 2. Excerpt from Schumann's Op. 24, No. 4 as scanned into SharpEye

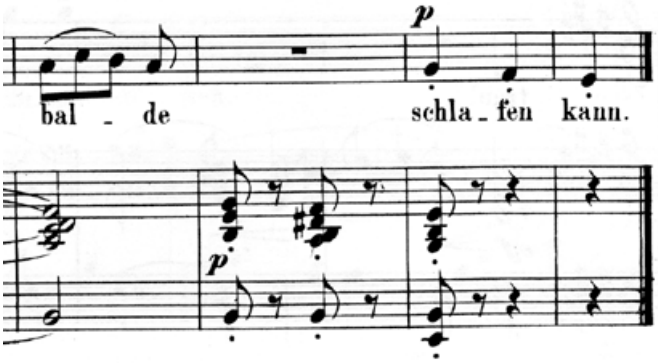


Figure 3 shows what the last four measures of the song look like when imported into Finale using MusicXML:

Figure 3. Importing SharpEye into Finale via MusicXML

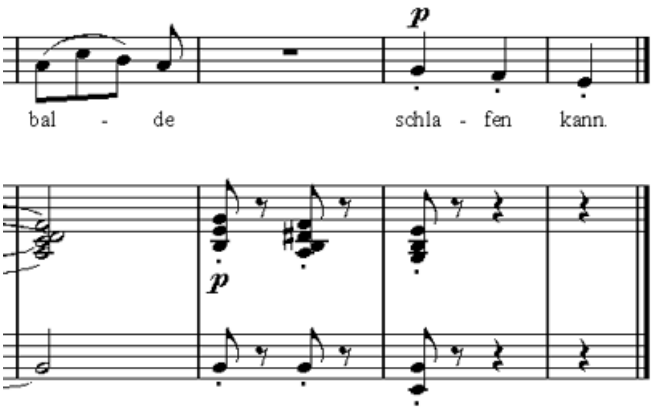
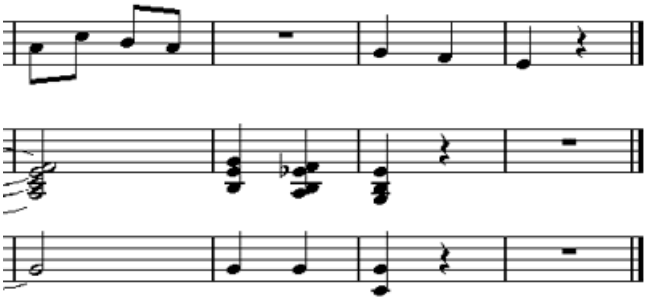


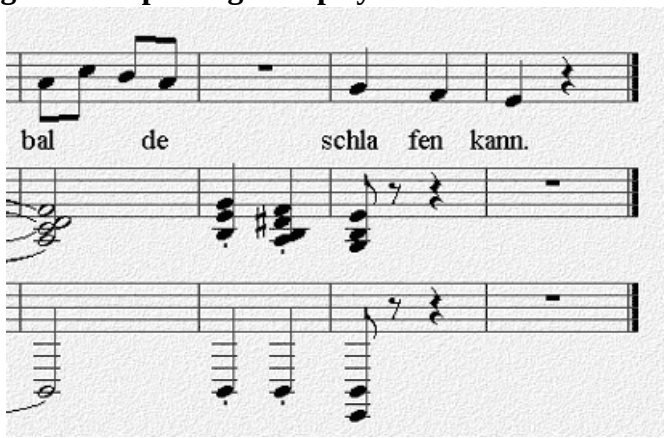
Figure 4 shows the last four measures when imported into Finale using MIDI:

Figure 4. Importing SharpEye into Finale via MIDI



The song lyrics are in the MIDI file, though Finale's reader did not import them. Figure 5 shows the last four measures when imported into Sibelius using MIDI. Sibelius reads the lyrics, but uses treble instead of bass clef for the left hand of the piano part.

Figure 5. Importing SharpEye into Sibelius via MIDI



As you can see, the MIDI imports are much less accurate than the MusicXML import, even with a simple music example. Why is this the case? Let's compare what is represented in the MIDI file vs. the MusicXML file, using an XML version of the binary MIDI format. Let us

look at the second measure of the left hand of the piano part. In the MusicXML file, we set the divisions to 6 to handle some triplets earlier in the song, so our four eighth notes look like:

```
<note>
  <pitch>
    <step>B</step>
    <octave>2</octave>
  </pitch>
  <duration>3</duration>
  <voice>3</voice>
  <type>eighth</type>
  <stem>up</stem>
  <staff>2</staff>
  <notations>
    <articulations>
      <staccato/>
    </articulations>
  </notations>
</note>
<note>
  <rest/>
  <duration>3</duration>
  <voice>3</voice>
  <type>eighth</type>
  <staff>2</staff>
</note>
<note>
 <pitch>
    <step>B</step>
    <octave>2</octave>
  </pitch>
  <duration>3</duration>
  <voice>3</voice>
  <type>eighth</type>
  <stem>up</stem>
  <staff>2</staff>
  <notations>
    <articulations>
      <staccato/>
    </articulations>
  </notations>
</note>
<note>
  <rest/>
  <duration>3</duration>
  <voice>3</voice>
  <type>eighth</type>
  <staff>2</staff>
</note>
```

In the MIDI file, represented using MIDI XML, the measure looks like this:

```
<NoteOn>
  <Delta>0</Delta>
  <Channel>2</Channel>
  <Note>47</Note>
  <Velocity>64</Velocity>
</NoteOn>
<NoteOff>
```

```
    <Delta>48</Delta>
    <Channel>2</Channel>
    <Note>47</Note>
    <Velocity>64</Velocity>
  </NoteOff>
  <NoteOn>
    <Delta>48</Delta>
    <Channel>2</Channel>
    <Note>47</Note>
    <Velocity>64</Velocity>
  </NoteOn>
  <NoteOff>
    <Delta>48</Delta>
    <Channel>2</Channel>
    <Note>47</Note>
    <Velocity>64</Velocity>
  </NoteOff>
```

Consider the differences between the two formats. MIDI has no discrete note element; rather, notes are bounded by NoteOn and NoteOff events. Rests are not represented at all; they are inferred from the absence of notes. This actually works very well for MIDI's intended use with synthesizers and electronic musical instruments. It is not very well suited for music notation. Given how much guessing the notation programs have to do to interpret a Standard MIDI File, you can understand why the results fall short, and fall short in a different way for each MIDI importing program.

MIDI also has no way to distinguish between a D-sharp and an E-flat; the one above middle C has a Note value of 63 in either case. Here Sibelius guessed correctly, while Finale guessed wrong. MIDI has no representation of beams or stem direction, and both programs got the beaming wrong in the voice part. The beaming follows the slur - which is also not represented in MIDI. Clefs are also missing from MIDI, so Sibelius guessed wrong on one part where Finale guessed correctly.

MIDI is the only music interchange format in common use for music notation today. When you can see all the ambiguities and missing data it produces, in this simple 4-bar example of a simple song, you can see why sheet music desperately needs a comprehensive, Internet-friendly interchange format. MusicXML has a tremendous advantage compared to prior efforts like NIFF and SMDL: XML had not been invented yet when the earlier teams did their work.

4. Freedom of Choice for Music Software Developers

One limitation to developing music software has been the tight coupling of music formats to development tools. For instance, Finale plug-ins require C or C++ programming, the Humdrum toolkit requires familiarity with Unix, and MuseData tools run on TenX, a non-standard DOS environment. The tight coupling of programming environment to data representation has limited the freedom and productivity of music software developers.

The promise of XML is that with the widespread availability of XML tools, MusicXML programmers can choose from a much wider range of development tools. We were delighted to

see the promise become a reality during the MusicXML alpha test, where programmers developed MusicXML programs in many different environments, including:

- Visual Basic using the MSXML parser on Windows (Recordare)
- Java using the Xerces parser on Linux, Macintosh OS X, and Windows (Xemus and individual developers)
- C using no parser on Windows (Visiv)

The ability to use rapid application development tools like Visual Basic to work with MusicXML makes it possible to build analysis programs using much more common development skills than the Unix expertise required for Humdrum. Good[Good 2001] illustrates this with some sample visual analysis programs that were written in half a day with Visual Basic, ActiveX controls, and MusicXML.

5. Future Directions

Now that MusicXML 0.5 can handle the basics of interchange between notation and performance applications, there are two main efforts planned to meet the goals of growing the downloadable sheet music market:

1. Test and refine MusicXML with retrieval and analysis applications.
2. Reach out to more music software developers and publishers to broaden MusicXML's reach.

Music information retrieval is complex: the queries are often "fuzzy" (as in the ultimate goal of query by humming or singing), and the data relationships are complicated. We have made some initial attempts to use the June 2001 working draft of XQuery for extremely simple queries of musical melodies, but the results have been discouraging. XQuery's current capabilities for handling queries based on complex, ordered relationships between XML document elements does not seem as strong as its capabilities for Structured Query Language (SQL)-like queries. SQL techniques unfortunately do not get us very far in music information retrieval.

We believe that MusicXML provides the structure that is needed for music information retrieval, but if XQuery will not handle this domain, specialized query tools may need to be developed. This would be unfortunate for music information retrieval. Music software is a small business compared to other software application areas. Much of XML's attraction for music comes from its ability to leverage the investment in XML tools made by larger software markets. If standard XML query tools cannot meet even the most basic needs of music information retrieval, the music community is not likely to be able to take advantage of the optimizations and features provided by new generations of XML database tools.

Music information retrieval likely has years of research ahead on algorithm development. We believe that XML tools and music information retrieval tools can co-evolve together to meet user needs. Music can serve as a useful application area to broaden the scope of standardized

XML query tools. Meanwhile, a standard XML format used in XML databases can let music information retrieval researchers focus on the difficult questions of useful algorithms. The focus needs to move to low-level database representation only when that directly affects these algorithms.

If we really can get query-by-humming to work well for average customers, this could have large commercial implications. But we do not expect to need these breakthroughs to meet the goals of growing the downloadable sheet music market. For this, it should be sufficient to expand the reach of MusicXML to more music software available on people's personal computers. If all you can do with downloadable music is play it and print it with one program, why would you buy it compared to paper? But if you can edit the music, use it as a smart accompaniment, look at the musical score together with the playback of a CD, move the music to an electronic music stand, and write new musical programs yourself, the value of the downloadable sheet music increases dramatically. Once the music you download can be used on most any music program on your PC, downloadable sheet music will start to have more value - or different but complementary value - than paper music. Pervasiveness was a major part of what made the MP3 audio format so popular. We need that type of pervasiveness to make any type of digital sheet music more popular.

As with MP3 and MIDI, the flip side of pervasiveness is security, or the lack of same. We are hopeful that XML digital signatures can contribute in this area. This is an area that will require careful development before many music publishers are likely to embrace MusicXML or similar technologies.

In the short term, our focus will be to extend MusicXML's reach to many different music software applications, such as music education. Our early successes with MusicXML give us hope that we may at last have a standardized music interchange format, which in turn will enable the growth of the downloadable sheet music and music software markets.

Acknowledgements

Eleanor Selfridge-Field, Walter B. Hewlett, Barry Vercoe, and David Huron provided valuable advice and encouragement, along with their outstanding prior work in music representation. Graham Jones, Ian Carter, William Will, and Craig Sapp were especially helpful during the MusicXML Finale Converter alpha test.

Bibliography

[Castan 2001] Castan, Gerd, Michael Good, and Perry Roland (2001). Extensible Markup Language (XML) for Music Applications: An Introduction. In *The Virtual Score: Representation, Retrieval, Restoration*, ed. Walter B. Hewlett and Eleanor Selfridge-Field (Cambridge, MA: MIT Press), 95-102.

[Gilb 1988] Gilb, Tom (1988). *Principles of Software Engineering Management*. Reading, MA: Addison-Wesley.

[Good 1988] Good, Michael (1988). Software Usability Engineering. *Digital Technical Journal*, No. 6, 125-133. Republished at http://www.recordare.com/good/dtj.html.

[Good 2001] Good, Michael (2001). MusicXML for Notation and Analysis. In *The Virtual Score: Representation, Retrieval, Restoration*, ed. Walter B. Hewlett and Eleanor Selfridge-Field (Cambridge, MA: MIT Press), 113-124.

[Gould 1985] Gould, John D. and Clayton Lewis (1985). Designing for usability: Key principles and what designers think. *Communications of the ACM*, 28 (3), 300-311.

[Grande 1997] Grande, Cindy (1997). The *Notation Interchange File Format*: A *Windows*-Compliant Approach. In *Beyond MIDI: The Handbook of Musical Codes*, ed. Eleanor Selfridge-Field (Cambridge, MA: MIT Press), 491-512.

[Hewlett 1997] Hewlett, Walter B. (1997). *MuseData*: Multipurpose Representation. In *Beyond MIDI: The Handbook of Musical Codes*, ed. Eleanor Selfridge-Field (Cambridge, MA: MIT Press), 402-447.

[Huron 1997] Huron, David (1997). *Humdrum* and *Kern*: Selective Feature Encoding. In *Beyond MIDI: The Handbook of Musical Codes*, ed. Eleanor Selfridge-Field (Cambridge, MA: MIT Press), 375-401.

[MIDI 1996] *The Complete MIDI 1.0 Detailed Specification*. Document version 96.1. Los Angeles: The MIDI Manufacturers Association (1996).

[Sloan 1997] Sloan, Donald (1997). *HyTime* and *Standard Music Description Language*: A Document-Description Approach. In *Beyond MIDI: The Handbook of Musical Codes*, ed. Eleanor Selfridge-Field (Cambridge, MA: MIT Press), 469-490.

Glossary

MIDI	Musical Instrument Digital Interface
NIFF	Notation Interchange File Format
OASIS	Organization for the Advancement of Structured Information Standards
PDF	Portable Document Format
RIFF	Resource Interchange File Format
SGML	Standard Generalized Markup Language
SMDL	Standard Music Description Language

SQL Structured Query Language

Biography

Michael **Good**
CEO
Recordare LLC
Los Altos
U.S.A.

Michael Good, founder and CEO of Recordare LLC, invented the MusicXML system for Internet-friendly music notation. He started his work in music software at the MIT Experimental Music Studio, one of the forerunners of the MIT Media Lab. Before founding Recordare, Michael worked for 20 years in the software usability engineering field at SAP, Xtensory, and Digital Equipment Corporation. He has sung tenor with many musical groups in the Silicon Valley and Boston areas, including the choruses for the Cabrillo Music Festival, San Jose Symphony, Boston Symphony, and Boston Pops. His trumpet playing with the MIT Symphony Orchestra can be heard on Vox CDs.