MusicXML 3.1 Tutorial

MusicXML is a digital sheet music interchange and distribution format. The goal is to create a universal format for common Western music notation, similar to the role that the MP3 format serves for recorded music. The musical information is designed to be usable by notation programs, sequencers and other performance programs, music education programs, and music databases.

The goal of this tutorial is to introduce MusicXML to software developers who are interesting in reading or writing MusicXML files. MusicXML has many features that are required to support the demands of professional-level music software. But you do not need to use or understand all these elements to get started.

MusicXML FAQ

Why did we need a new format? What's behind some of the ways that MusicXML looks and feels? What software tools can I use? Is MusicXML free?

"Hello World" in MusicXML

Here you will find your simplest MusicXML file - one part, one measure, one note.

The Structure of MusicXML Files

There are two ways of structuring MusicXML files - measures within parts, and parts within measures. This section describes how to do it either way, and how to switch back and forth between them. It also discusses the descriptive data that goes at the start of a MusicXML file.

The MIDI-Compatible Part of MusicXML

What parts of MusicXML do I need to represent a MIDI sound file? The MIDI equivalents in MusicXML are described here.

Notation Basics

Here we discuss the basic notation features that go beyond MIDI's capabilities, including stems, beams, accidentals, articulations, and directions.

Chord Symbols and Diagrams

MusicXML provides a rich representation for harmonies, both for harmonic analysis and for chord symbols. Here we discuss how to create the chord symbols and diagrams found in much contemporary sheet music, including lead sheets, piano/vocal/guitar arrangements, and big-band charts.

Tablature

Here we describe the basics of tablature notation: specifying strings, frets, string tunings, and guitar-specific notations like hammer-ons and pull-offs.

Percussion

Here we discuss the steps needed to represent unpitched percussion parts such as drum kits. Some of these techniques apply to other types of music, such as the use of multiple instruments, alternate noteheads, and different measure styles.

Compressed .MXL Files

MusicXML 2.0 added a compressed zip-based format that greatly reduces MusicXML file sizes. Here we discuss the structure of the compressed .mxl format.

Table of Contents

MusicXML FAQ

Why did we need a new music notation format?

There are many fine computer music programs in the world. Unfortunately, sharing music between them used to be difficult. This was a real problem since no one program can do everything equally well. Having to reenter musical data for each program you want to use is a big inconvenience to everyone who uses more than one music software program.

Before MusicXML, the only music notation interchange format commonly supported was MIDI. MIDI is a wonderful format for performance applications like sequencers, but it is not so wonderful for other applications like music notation. MIDI does not know the difference between an F-sharp and a G-flat; it does not represent stem direction, beams, repeats, slurs, measures, and many other aspects of notation.

People had recognized for years that a new interchange format was needed, but no prior attempt at creating a new standard format had succeeded. NIFF had probably been the most successful attempt to date, but its use was very limited and the format was not being maintained. SMDL was the most ambitious attempt, but no software actually used it.

An Internet-friendly standard format was necessary for the growth of the Internet music publishing market. Before MusicXML, it is like using the Internet before HTML was invented, or using synthesizers before MIDI was invented.

Why not use an existing format like NIFF or SMDL?

NIFF and SMDL were noble efforts to solve the same type of interchange problem that MusicXML addresses. So why don't we use them rather than inventing something new?

NIFF represents music using a graphical format. There is no concept of a "C" in NIFF: instead, you determine pitch by its placement on a staff. This type of graphical music representation has a long and distinguished history. It works well for the scanning programs that were the focus of NIFF's work. But it works poorly for many other types of applications, such as sequencing and musical databases. For both of these applications, MIDI works much better than NIFF; for notation, though, NIFF is more complete than MIDI. MusicXML is designed to meet the interchange needs for all these types of applications.

A graphical format like NIFF really does not work well for general interchange, which is one of the main reasons NIFF has not been adopted by more programs. Another major impediment is that NIFF is a binary format, rather than a text format like XML. It is much easier to write and debug programs that read and write to a text format vs. a binary format.

SMDL suffered from the problem of trying to be a general-purpose solution to the problem of representing all possible musics of the past, present, and future. In addition, the project was not grounded in practical implementation experience. The result was something so complicated that practically nobody can understand it. The overwhelming complexity greatly inhibited SMDL's adoption - it certainly inhibited us! - and no commercial software supports it.

Where did the design of MusicXML come from?

MusicXML was based primarily on two academic music formats:

- The MuseData format, developed by Walter Hewlett at the Center for Computer Assisted Research in the Humanities (CCARH), located at Stanford University
- The Humdrum format, developed by David Huron, based at Ohio State University.

Eleanor Selfridge-Field from CCARH edited a magnificent book called *Beyond MIDI: A Handbook of Musical Codes*. Studying this volume made it clear that, as we had been advised, MuseData and Humdrum were the most comprehensive starting points for the type of language we wanted to build.

The first beta version of MusicXML was basically an XML updating of MuseData, with the addition of some key concepts from Humdrum. Since both formats have been used primarily for work in classical and folk music, MusicXML was extended beyond those boundaries to better support contemporary popular music. In recent releases, MusicXML's capabilities have been greatly expanded so it can serve as a distribution format for digital sheet music as well as an interchange format. Many features have been added in order to make MusicXML near-lossless when it comes to capturing how music is formatted on the printed page.

Why do you use XML?

XML was designed to solve exactly the type of problem we face today in music software. Say you have 100 music applications, each with its own format. For each application to communicate with the other, 10,000 separate programs would need to be written without a common interface language! With a common interface language, each application writes only one program, so only 100 separate programs would be required. Consumers gain enormous value at relatively small cost to the software developer.

XML builds on decades of experience with its predecessor SGML, as well as the experience with the explosive growth of HTML and the Web. XML hits the magic "sweet spot" between simplicity and power, just as HTML and MIDI have done in the past. It is designed to represent complex, structured data, and musical scores fit that description.

Using XML frees us from worrying about the basic syntax of the language, instead letting us worry about the semantics - what to represent, versus the syntax of how to represent it. Similarly, there is no need to write a low-level parser to read the language: XML parsers exist everywhere. Basing a music interchange language on XML lets music software, a relatively small community, leverage the investment in tools made by much larger markets such as electronic commerce.

Is MusicXML free?

Yes. MusicXML 3.1 is developed by the W3C Music Notation Community Group and licensed under the W3C Community Final Specification Agreement.

Is MakeMusic's software open source?

No. It is important for the MusicXML format itself to be free and open source. MakeMusic transferred responsibility for the MusicXML format to the W3C Music Notation Community Group to help ensure its continued success within the music industry.

MakeMusic does not see a similar advantage to making our own Finale MusicXML software open source. Our Sibelius plug-in is not open source, but we may decide to change that in the future. Many open source applications include MusicXML support.

Who is using MusicXML?

Many companies, music software developers, and scholars are using MusicXML in their products and research. People use MusicXML with many different types of notation software, such as:

- Desktop notation editing programs including Finale, Sibelius, Dorico, MuseScore, capella, Encore, and Overture.

- Web-based notation programs for editing, playback, and education, including Noteflight, SmartMusic, Soundslice, Flat, Gustaf, and Scorio.
- Digital audio workstation programs including Cubase, Digital Performer, Logic, Reaper, and Sonar.
- Music scanning programs, including SmartScore, PhotoScore, SharpEye Music Reader, capella-scan, ScoreMaker, PlayScore, and Audiveris.
- Tablature editors, including Guitar Pro, Progression, Fandango, TablEdit, TaBazar, and TuxGuitar.
- iOS notation programs for iPhone and iPad, including Komp, Notion, Symphony Pro, and SeeScore.
- Android notation programs, including Ensemble Composer, Notate, and NotateMe.
- Electronic music stands, including Newzik, Blackbinder, and OrganMuse.
- Musicology applications and toolkits, including music21 and MelodicMatch.

See our MusicXML software page at **www.musicxml.com/software/** for a more complete list, including links to each application.

What software tools are available?

One of the great things about basing our work on XML is that there are tools available for practically every computer platform.

At Recordare, we used Microsoft tools for our first generation of Dolet software. The original Dolet for Finale plug-in was written in a mix of Visual Basic 6.0 and Visual C++ 6.0, using Microsoft's MSXML parser. The Microsoft parser was designed to be easy to use within Visual Basic and succeeded admirably. We had great success with it.

We then rewrote the Dolet for Finale plug-in in Java and C++ so that our software could run on macOS as well as Windows. Many other projects are using Java for multiple platforms, including Linux. We used the Xerces parser from the Apache group, as are most of the other Java-based MusicXML projects that we know. We have also heard good reports about the Xerces C++ parser. Other projects use the built-in XML support provided by Python, JavaScript, and other programming languages.

If all you want to do is write MusicXML files, not read them, you really don't need a parser, though you may find it useful. A scanner like SharpEye Music Reader, for instance, does not have any great need to read MusicXML files. Its MusicXML support is written in C, like the rest of the program, without using any special XML tools. The pae2xml translator is written in Perl. The Dolet 6 for Sibelius plug-in is written in ManuScript. You can also use XSLT to read and transform MusicXML files.

There are many XML sites that can guide you to XML tools beyond what we list here. If you are using an XML parser, in most cases you will probably be using the XML DOM (Document Object Model) to handle MusicXML files. Good DOM support is likely to be more important than SAX support for MusicXML programs.

Many tools today will automatically generate classes from an XSD schema definition. MusicXML's schema has some complexity that can require manual edits of the automated output, but many developers find these tools to save a lot of time in their MusicXML development.

Why did you release an XSD for MusicXML 2.0?

When we started developing MusicXML, a DTD (Document Type Definition) was the only official W3C recommendation for defining an XML document. Since that time, XSD (XML

Schema Definition) has become an official W3C recommendation, and alternative standards like RELAX NG have also become available.

When XSDs were first released, neither the supporting XSD software nor the state of MusicXML application usage was sufficiently advanced to take advantage of this technology. XSDs are more complex than DTDs, so it was clear there would be a cost to creating a MusicXML XSD schema. At the time, it was also not clear how support would evolve for XSD, RELAX NG, and other schema alternatives. It thus made sense to stay with DTDs for the releases of MusicXML 1.0 and 1.1.

XSD and MusicXML technology have both matured significantly in the past few years. Developer tool support for XSDs is now as pervasive as for DTDs. New XML tools such as XQuery and XML data binding tools can work much better with XSDs than DTDs. While RELAX NG has some advantages over XSDs, its software support is not yet as pervasive as for XSDs.

In addition, MusicXML's move from an interchange to a distribution format makes it increasingly important to provide the most powerful tools possible for automated quality assurance. Validating against an XSD can catch many more errors in XML document creation than validating against a DTD.

These combined customer needs and opportunities regarding XSDs led us to develop a MusicXML 2.0 XSD that was released in September 2008. The XSD puts much more of MusicXML's semantics into the language definition itself, rather than just in the documentation. Thus there are many documents that validate against the DTD that do not validate against the XSD, but these reflect MusicXML file errors that you do want to catch as early and automatically as possible.

DTDs remain more readable than XSDs for many people. To learn MusicXML, you may find it easiest to read the DTDs, the XSDs, or both in combination. This will depend on your experience and comfort level with the different technologies. The XSDs will generally offer more precise definitions than the DTDs since the format is now much more strongly typed.

Why do you use all these elements instead of attributes?

This is mainly a stylistic decision. Several XML books advise representing semantics in elements rather than attributes where possible. One advantage of doing this is that elements have structure, but attributes do not. If you find that what you are representing really has more than one part, you can create a hierarchical structure with an element. With attributes, you are limited to an unordered list. For information retrieval applications, it can also be easier to search directly for elements rather than for attribute/element combinations.

In MusicXML, attributes are generally limited to a few uses:

- To indicate if this is where an element starts or stops, such as for slurs and tuplets.
- To identify elements, as in measure numbers or beam levels.
- To suggest how an element would best be printed.
- To suggest how an element would best be converted into MIDI or other sound formats.

In both MusicXML 1.1 and 2.0, the third category - suggesting how an element would best be printed - grew enormously. So you will find that MusicXML 3.1 files make much more use of attributes than do MusicXML 1.0 files. The principles determining when to use elements and when to use attributes have remained the same.

In summary, we follow common recommended practice by using elements for data and attributes for metadata. We find this works well. Certainly there are DTDs and XSDs in other domains that use attributes much more extensively, and they work well too. Either way can do the job as long as it is applied consistently.

While there are a lot of MusicXML elements, we have tried to limit them to elements that directly represent musical concepts. We have tried to avoid using elements that would introduce concepts not found in musical scores. For example, there is no "timeslice" element as found in formats like NIFF. We have generally found that we can represent what applications need without having to introduce artificial elements.

Why is MusicXML so verbose? Isn't that inefficient?

MusicXML is both an interchange and distribution format. In both cases, ease of comprehension is much more important than terseness. Musical representation is complex enough without trying to figure out an abbreviated code. The XML specification advises that "terseness in XML markup is of minimal importance", and we believe this is really true. Somebody who understands the domain should be able to figure out the basics of an XML file simply by reading it. We believe that MusicXML meets this goal, where several other musical XML proposals do not.

There is of course a place for terse text formats like abc or Plaine and Easie, which allow easy and fast text entry of musical data. There are converters for both abc and the Plaine and Easie codes to MusicXML.

Uncompressed MusicXML files do take up a lot of space, which can be problematic for digital sheet music distribution. MusicXML 2.0 added a compressed zip format with a .mxl suffix that can make files roughly 20 times smaller than their uncompressed version. As an example, a four-page Joplin rag takes up 518K as an uncompressed .musicxml file, but only 19K as a compressed .mxl file. This is even smaller than the MIDI representation, which is 21K, and the MXL file contains much more data about the music.

Why do I see text instead of music when I look at a MusicXML file in my browser?

In order to see a MusicXML file as music within a browser, you need to use a web application that can understand, display, and playback MusicXML files. In the past these applications often used Flash, but nowadays they usually use HTML5.

MusicXML 3.1 has two registered media types:

- application/vnd.recordare.musicxml for compressed .mxl files
- application/vnd.recordare.musicxml+xml for uncompressed .musicxml files

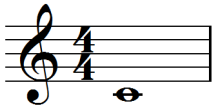
Perhaps in the future, browsers will be able to automatically respond to these media types so that files outside of web pages will be displayed and played back automatically.

It would be possible to build an XSLT stylesheet to convert MusicXML into a web-readable format, and a proof of concept for this was done by one MusicXML user. However, building a professional quality XSLT for music notation would be a difficult task.

"Hello World" in MusicXML

Brian Kernighan and Dennis Ritchie popularized the practice of writing a program that prints the words "hello, world" as the first program to write when learning a new programming language. It is the minimal program that tests how to build a program and display its results.

In MusicXML, a song with the lyrics "hello, world" is actually more complicated than we need for a simple MusicXML file. Let us keep things even simpler: a one-measure piece of music that contains a whole note on middle C, based in 4/4 time:



Here it is in MusicXML:

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<!DOCTYPE score-partwise PUBLIC
    "-//Recordare//DTD MusicXML 3.1 Partwise//EN"
    "http://www.musicxml.org/dtds/partwise.dtd">
<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>
```

Let's look at each part in turn:

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
```

This is the XML declaration required of all XML documents. We have specified that the characters are written in the Unicode encoding "UTF-8". This is the version of Unicode that has ASCII as a subset. Setting the value of standalone to "no" means that we are defining the document with an external definition in another file.

```
<!DOCTYPE score-partwise PUBLIC
    "-//Recordare//DTD MusicXML 3.1 Partwise//EN"
    "http://www.musicxml.org/dtds/partwise.dtd">
```

This is where we say that we are using MusicXML, specifically a partwise score where measures are contained within parts. We use a PUBLIC declaration including an Internet location for the DTD. The URL in this declaration is just for reference. Most applications that read MusicXML files will want to install a local copy of the MusicXML DTDs on the user's machine. Use the entity resolver in your XML parser to validate against the local copy, rather than reading the DTDs slowly over the network.

If your application wants to validate against the MusicXML XSD rather than a DTD, you can use an entity resolver in your XML parser to do this. When writing MusicXML files, writing the DOCTYPE makes it easier for all applications - DTD or XSD based - to validate MusicXML files.

```
<score-partwise version="3.1">
```

This is the root document type. The <score-partwise> element is made up of parts, where each part is made up of measures. There is also a <score-timewise> option which is made up of measures, where each measure is made up of parts. The version attribute lets programs distinguish what version of MusicXML is being used more easily. Leave it out if you are writing MusicXML 1.0 files.

```
  <part-list>
    <score-part id="P1">
      <part-name>Part 1</part-name>
    </score-part>
  </part-list>
```

Whether you have a partwise or timewise score, a MusicXML file starts off with a header that lists the different musical parts in the score. The above example is the minimal part-list possible: it contains one score-part, the required id attribute for the score-part, and the required part-name element.

```
  <part id="P1">
```

We are now beginning the first (and only, in this case) part within the document. The id attribute here must refer to an id attribute for a score-part in the header.

```
    <measure number="1">
```

We are starting the first measure in the first part.

```
      <attributes>
```

The attributes element contains key information needed to interpret the notes and musical data that follow in this part.

```
        <divisions>1</divisions>
```

Each note in MusicXML has a duration element. The divisions element provided the unit of measure for the duration element in terms of divisions per quarter note. Since all we have in this file is one whole note, we never have to divide a quarter note, so we set the divisions value to 1.

Musical durations are typically expressed as fractions, such as "quarter" and "eighth" notes. MusicXML durations are fractions, too. Since the denominator rarely needs to change, it is

represented separately in the divisions element, so that only the numerator needs to be associated with each individual note. This is similar to the scheme used in MIDI to represent note durations.

```
<key>
  <fifths>0</fifths>
</key>
```

The key element is used to represent a key signature. Here we are in the key of C major, with no flats or sharps, so the fifths element is 0. If we were in the key of D major with 2 sharps, fifths would be set to 2. If we were in the key of F major with 1 flat, fifths would be set to -1. The name "fifths" comes from the representation of a key signature along the circle of fifths. It lets us represent standard key signatures with one element, instead of separate elements for sharps and flats.

```
<time>
  <beats>4</beats>
  <beat-type>4</beat-type>
</time>
```

The time element represents a time signature. Its two component elements, beat and beat-type, are the numerator and denominator of the time signature, respectively.

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

MusicXML allows for many different clefs, including many no longer used today. Here, the standard treble clef is represented by a G clef on the second line of the staff (e.g., the second line from the bottom of the staff is a G).

```
</attributes>
<note>
```

We are done with the attributes, and are ready to begin the first note.

```
<pitch>
  <step>C</step>
  <octave>4</octave>
</pitch>
```

The pitch element must have a step and an octave element. Optionally it can have an alter element, if there is a flat or sharp involved. These elements represent sound, so the alter element must always be included if used, even if the alteration is in the key signature. In this case, we have no alteration. The pitch step is C. The octave of 4 indicates the octave the starts with middle C. Thus this note is a middle C.

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

Our divisions value is 1 division per quarter note, so the duration of 4 is the length of 4 quarter notes.

```
<type>whole</type>
```

The <type> element tells us that this is notated as a whole note. You could probably derive this from the duration in this case, but it is much easier to work with both notation and performance applications if the notation and performance data is represented separately.

In any event, the performance and notation data do not always match in practice. For example, if you want to better approximate a swing feel than the equal eighth notes notated in a jazz chart, you might use different duration values while the type remains an eighth note. Bach's music contains examples of shorthand notation where the actual note durations do not match the

standard interpretation of the notes on the page, due to his use of a notational shorthand for certain rhythms.

The duration element should reflect the intended duration, not a longer or shorter duration specific to a certain performance. The note element has attack and release attributes that suggest ways to alter a note's start and stop times from the nominal duration indicated directly or indirectly by the score.

```
      </note>
```

We are done with the note.

```
    </measure>
```

We are done with the measure.

```
  </part>
```

We are done with the part.

```
</score-partwise>
```

And we are done with the score.

One limitation of XML's document type definitions is that if you want to limit the number of elements within another element, you generally must also restrict how they are ordered as well. In the attributes, for instance, we want no more than one divisions element; for the note's pitch, we want one and only one step element and octave element. In order to do this, the order in which these elements appear must be constrained as well.

Thus the order in which elements appear in these examples does matter. The DTD definitions should make it clear what ordering is required; we will not spell that out in detail during the tutorial.

The Structure of MusicXML Files

Adapting Musical Scores to a Hierarchy

Say we have a piece of music for two or more people to play. It has multiple parts, one per player, and multiple measures. XML represents data in a hierarchy, but musical scores are more like a lattice. How do we reconcile this? Should the horizontal organization of musical parts be primary, or should the vertical organization of musical measures?

The answer is different for every music application. David Huron, a music cognition specialist and the inventor of Humdrum, advised us to make sure we could represent music both ways, and be able to switch between them easily.

This is why MusicXML has two different top-level DTDs, each with its own root element. If you use the partwise DTD, the root element is <score-partwise>. The musical part is primary, and measures are contained within each part. If you use the timewise DTD, the root element is <score-timewise>. The measure is primary, and musical parts are contained within each measure. The MusicXML XSD includes both of the top-level document elements in a single XSD file.

Having two different structures does not work well if there is no automatic way to switch between them. MusicXML provides two XSLT stylesheets to convert back and forth between the two document types. The parttime.xsl stylesheet converts from <score-partwise> to <score-timewise>, while the timepart.xsl stylesheet converts from <score-timewise> to <score-partwise>.

An application reading MusicXML can choose which format is primary, and check for that document type. If it is your root element, just proceed. If not, check to see if it is the other MusicXML root element. If so, apply the appropriate XSLT stylesheet to create a new MusicXML document in your preferred format, and then proceed. If it is neither of the two top-level document types, you do not have a MusicXML score, and can return an appropriate error message.

When your application writes to MusicXML, simply write to whichever format best meets your needs. Let the program reading the MusicXML convert it if necessary. If you have a two-dimensional organization in your program so that either format is truly equally easy to write, consider using the score-partwise format. Most of today's MusicXML software uses this format, so if all else is equal, conversion times should be lower overall.

Top-Level Document Elements

The score.mod file defines the basic structure of a MusicXML file. The primary definition of the file is contained in these lines:

```
<![ %partwise; [
<!ELEMENT score-partwise (%score-header;, part+)>
<!ELEMENT part (measure+)>
<!ELEMENT measure (%music-data;)>
]]>
<![ %timewise; [
<!ELEMENT score-timewise (%score-header;, measure+)>
<!ELEMENT measure (part+)>
<!ELEMENT part (%music-data;)>
]]>
```

The %partwise; and %timewise; entities are set in the top-level DTDs partwise.dtd and timewise.dtd. The <![lines indicate a conditional clause like the #ifdef statement in C. So if

partwise.dtd is used, the <score-partwise> element is defined, while if timewise.dtd is used, the <score-timewise> element is defined.

You can see that the only difference between the two formats is the way that the part and measure elements are arranged. A score-partwise document contains one or more part elements, and each part element contains one or more measure elements. The score-timewise document reverses this ordering.

In either case, the lower-level elements are filled with a music-data entity. This contains the actual music in the score, and is defined as:

```
<!ENTITY % music-data
        "(note | backup | forward | direction | attributes |
          harmony | figured-bass | print | sound | barline |
          grouping | link | bookmark)*">
```

In addition, each MusicXML file contains a %score-header; entity, defined as:

```
<!ENTITY % score-header
        "(work?, movement-number?, movement-title?,
          identification?, defaults?, credit*, part-list)">
```

We will now look at the score-header entity in more detail. If the example in the preceding "Hello World" section gave you enough information, you may want to skip ahead to the next section that starts describing music-data.

The Score Header Entity

The score header contains some basic metadata about a musical score, such as the title and composer. It also contains the part-list, which lists all the parts or instruments in a musical score.

As an example, take our MusicXML encoding of "Mut," the 22nd song from Franz Schubert's song cycle *Winterreise*. Here is a sample score header for that work:

```
  <work>
    <work-number>D. 911</work-number>
    <work-title>Winterreise</work-title>
  </work>
  <movement-number>22</movement-number>
  <movement-title>Mut</movement-title>
  <identification>
    <creator type="composer">Franz Schubert</creator>
    <creator type="poet">Wilhelm Müller</creator>
    <rights>Copyright © 2001 Recordare LLC</rights>
    <encoding>
      <encoding-date>2002-02-16</encoding-date>
      <encoder>Michael Good</encoder>
      <software>Finale 2002 for Windows</software>
      <encoding-description>MusicXML 1.0 example</encoding-description>
    </encoding>
    <source>Based on Breitkopf &amp;amp; Härtel edition of 1895</source>
  </identification>
  <part-list>
    <score-part id="P1">
      <part-name>Singstimme.</part-name>
    </score-part>
    <score-part id="P2">
      <part-name>Pianoforte.</part-name>
    </score-part>
  </part-list>
```

You see that this score-header has all five of the possible top-level elements in the score-header entity: the work, movement-number, movement-title, identification, and part-list. Only the part-list is required, all other elements are optional.

Let's look at each part in turn:

```
<work>
  <work-number>D. 911</work-number>
  <work-title>Winterreise</work-title>
</work>
```

In MusicXML, individual movements are usually represented as separate files. The work element is used to identify the larger work of which this movement is a part. Schubert's works are more commonly referred to via D. numbers than opus numbers, so that is what we use in the work-number element; the work-title is the name of the larger work.

If you have all the movements in a work represented, you can use the opus element to link to the MusicXML opus file that in turn contains links to all the movements in the work.

```
<movement-number>22</movement-number>
```

Winterreise is a cycle of 24 songs. We use the movement-number to identify that "Mut" is the 22nd song in the cycle - it is not restricted to use for movements in a symphony.

```
<movement-title>Mut</movement-title>
```

Similarly, we use the movement-title element for the title of the individual song. If you have a single song that is not part of a collection, you will usually put the title of the song in the movement-title element, and not use either the work or movement-number elements.

```
<identification>
  <creator type="composer">Franz Schubert</creator>
  <creator type="poet">Wilhelm Müller</creator>
```

The identification element is defined in the identity.mod file. It contains basic metadata elements based on the Dublin Core. In this song, as many others, there are two creators: in this case, the composer and the poet. Therefore, we use two creator elements, and distinguish their roles with the type attribute. For an instrumental work with just one composer, there is no need to use the type attribute.

```
  <rights>Copyright © 2001 Recordare LLC</rights>
```

The rights element contains the copyright notice. You may have multiple rights elements if multiple copyrights are involved, say for the words and the music. As with the creator element, these can have type attributes to indicate what type of copyright is involved. In this example, both the words and music to Mut are in the public domain, but we are copyrighting our electronic edition of the work.

```
  <encoding>
    <encoding-date>2002-02-16</encoding-date>
    <encoder>Michael Good</encoder>
    <software>Finale 2002 for Windows</software>
    <encoding-description>MusicXML 1.0 example</encoding-description>
  </encoding>
```

The encoding element contains information about how the MusicXML file was created. Here we are using all four of the available sub-elements to describe the encoding. You can have multiple instances of these elements, and they can appear in any order.

```
  <source>Based on Breitkopf &amp;amp; Härtel edition of 1895</source>
</identification>
```

The source element is useful for music that is encoded from a paper published score or manuscript. Different editions of music will contain different musical information. In our case, we used the Dover reprint of the Breitkopf & Härtel edition of *Winterreise* as our starting point, correcting some errors in that published score.

The identification element also may contain a miscellaneous element. This in turn contains miscellaneous-field elements, each with a name attribute. This can be helpful if your software contains some identification information not present in the MusicXML DTD that you want to preserve when saving and reading from MusicXML.

```
<part-list>
  <score-part id="P1">
    <part-name>Singstimme.</part-name>
  </score-part>
  <score-part id="P2">
    <part-name>Pianoforte.</part-name>
  </score-part>
</part-list>
```

The part-list is the one part of the score header that is required in all MusicXML scores. It is made up of a series of score-part elements, each with a required id attribute and part-name element. By convention, our software simply numbers the parts as "P1", "P2", etc. to create the id attributes. You may use whatever technique you like as long as it produces unique names for each score-part.

In addition to the part-name, there are many optional elements that can be included in a score-part:

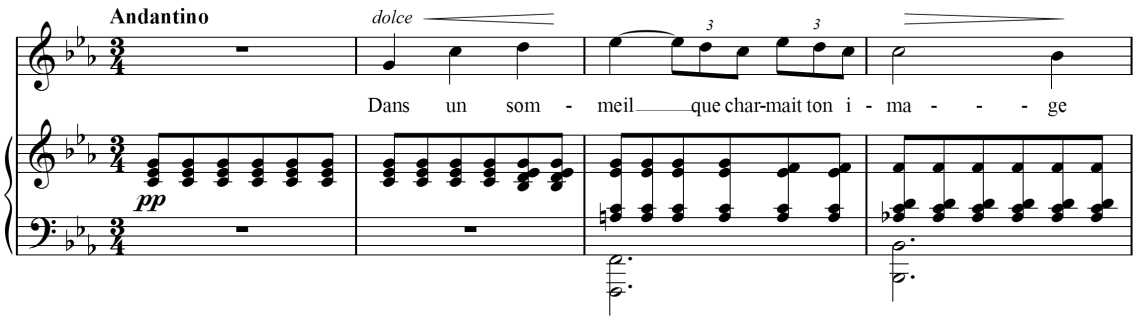
- An identification element, helpful if individual parts come from different sources.
- A part-abbreviation element. Often, you will use the part-name for the name used at the start of the score, and the part-abbreviation for the abbreviated name used in succeeding systems.
- A group element, used when different parts can be used for different purposes. In MuseData, for instance, there will often be different parts used for a printed score, a printed part, a MIDI sound file, or for data analysis.
- One or more score-instrument elements, used to describe instrument sounds and virtual instrument settings, as well as to define multiple instruments within a score-part. This element serves as a reference point for MIDI instrument changes.
- One or more midi-device elements for identifying the MIDI devices or ports that are being used in a multi-port configuration. Multiple devices let you get beyond MIDI's 16-channel barrier.
- One or more midi-instrument elements, specifying the initial MIDI setup for each score-instrument within a part.

The MIDI-Compatible Part of MusicXML

MusicXML music data contains two main types of elements. One set of elements is used primarily to represent how a piece of music should sound. These are the elements that are used when creating a MIDI file from MusicXML. The other set is used primarily to represent how a piece of music should look. These elements are used when creating a Finale file from MusicXML.

We encourage programs writing MusicXML to write as much accurate data as they can. The only elements that are required, though, are the sounding elements that relate directly to writing a MIDI file from MusicXML. This is where we will start in introducing the musical elements of a MusicXML file.

As an example, we will use the first four bars of "Après un rêve" by Gabriel Fauré:



Attributes

The attributes element contains information about time signatures, key signatures, transpositions, clefs, and other musical data that is usually specified at the beginning of a piece or at the start of a measure. We discuss the MIDI-compatible elements here; the rest are discussed in the following sections.

In this example, our Finale translator produces the following MIDI-compatible attributes:

```
<attributes>
  <divisions>24</divisions>
    <key>
      <fifths>-3</fifths>
      <mode>minor</mode>
    </key>
  <time>
    <beats>3</beats>
    <beat-type>4</beat-type>
  </time>
</attributes>
```

Divisions

Musical durations are commonly referred to as fractions: whole notes, half notes, quarter notes, and the like. While each musical note could have a fraction associated with it, MusicXML instead follows MIDI 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.

MusicXML allows divisions to change in the middle of a part, but most software will probably find it easiest to compute one divisions value per part and put that at the beginning of the first

measure. The divisions value of 24 in this example allows for both triplet eighth notes (duration of 8) and regular sixteenth notes (duration of 6).

Key

Standard key signatures are represented very much like MIDI key signatures. The fifths element specifies the number of flats or sharps in the key signature - negative for flats, positive for sharps. The fifths name indicates that this value represents the key signature's position on the circle of fifths. MusicXML uses the mode element to indicate major or minor key signatures.

Time

Standard time signatures are represented more simply in MusicXML than in MIDI. The beats element represents the time signature numerator, and the beat-type element represents the time signature denominator (vs. a log denominator in MIDI).

Transpose

If you are writing a part for a transposing instrument, the transposition must be specified in MusicXML in order for the sound output to be correct. The transpose element represents what must be added to the written pitch to get the correct sounding pitch.

The chromatic element, representing the number of chromatic steps to add to the written pitch, is the one required element. The diatonic, octave-change, and double elements are elements.

Say we have a part written for a trumpet in B-flat. A written "C" on this part will sound as a B-flat on a piano. This transposition is one diatonic step down (C to B) and two chromatic half steps down (C to B to B-flat). In MusicXML it would be represented as:

```
<transpose>
  <diatonic>-1</diatonic>
  <chromatic>-2</chromatic>
</transpose>
```

The diatonic element is not needed for correct MIDI output, but it helps get transposition notation correct and programs are encouraged to use it wherever possible.

The octave-change element is used when transpositions exceed an octave in either direction. The double element is used when the part should be doubled an octave lower, as when a single part is used for both cello and string bass.

Pitch

Pitch, duration, ties, and lyrics are all represented within the MusicXML note element. For example, the E-flat that starts bar 3 in the voice part has the following MIDI-compatible elements:

```
<note>
  <pitch>
    <step>E</step>
    <alter>-1</alter>
    <octave>5</octave>
  </pitch>
  <duration>24</duration>
  <tie type="start"/>
  <lyric>
    <syllabic>end</syllabic>
    <text>meil</text>
```

```
      <extend/>
    </lyric>
  </note>
```

In MIDI, a pitch is represented by a single number. MusicXML divides pitch up into three parts: the step element (A, B, C, D, E, F, or G), an optional alter element (-1 for flat, 1 for sharp), and an octave element (4 for the octave starting with middle C).

The pitch represents the sound, not what is notated, so an alter element must be included even if represents a flat or sharp that is part of the key signature. This is why the E-flat contains an alter element, though there is no accidental on the note.

Alter values of -2 and 2 can be used for double-flat and double-sharp. Decimal values can be used for microtones (e.g., 0.5 for a quarter-tone sharp), but not all programs may convert this into MIDI pitch-bend data.

For rests, a rest element is used instead of the pitch element. The whole rest in 3/4 that begins the voice part is represented as:

```
<note>
  <rest measure="yes"/>
  <duration>72</duration>
</note>
```

Duration

The duration element is an integer that represents a note's duration in terms of divisions per quarter note. Since our example has 24 divisions per quarter note in the voice part, a quarter note has a duration of 24. The eighth-note triplets have a duration of 8, while the eighth notes have a duration of 12.

Tied Notes

The sounding part of a tied note is indicated by the tie element. The tie element has a type of start for the starting note of a tie, and a type of stop for the ending note in a tie. A note element can have two tie elements. If a note is tied to the notes both before and after it, place the tie to the previous note, <tie type="stop">, before the <tie type="start"> to the next note.

Chords

The duration elements in MusicXML move a musical counter. To play chords, we need to indicate that a note should start at the same time as the previous note, rather than following the previous note. To do this in MusicXML, add a chord element to the note.

In our example, the piano part does not have rhythms more complex than eighth notes, so our converter sets the divisions value to 2. With 2 divisions per quarter note, the sound portion of the first chord in the piano part is represented as:

```
<note>
  <pitch>
    <step>C</step>
    <octave>4</octave>
  </pitch>
  <duration>1</duration>
</note>
<note>
  <chord/>
  <pitch>
    <step>E</step>
```

```
      <alter>-1</alter>
      <octave>4</octave>
    </pitch>
    <duration>1</duration>
  </note>
  <note>
    <chord/>
    <pitch>
      <step>G</step>
      <octave>4</octave>
    </pitch>
    <duration>1</duration>
  </note>
```

Each note in the chord following the first one includes a chord element before the pitch element.

If you find that you have notes in a chord with different durations, you are probably better representing this as multi-part music rather than a chord. If you must have notes with different durations in the chord, the longest note must be the first note in the chord.

Lyrics

While lyrics are not yet used in sound generation, they are included in Standard MIDI files, so we will discuss them here with the other MIDI-compatible features of MusicXML.

Lyrics in MusicXML use an optional syllabic element to indicate how a syllable fits into a word, rather than having conventions based on hyphens and spaces as some other formats do. The values for syllabic can be "single", "begin", "end", or "middle". We saw earlier that the E-flat starting the third measure had a syllabic value of "end", since "meil" was the end of a two-syllable word. The "ma" syllable in "image" has a syllabic value of "middle". In the second measure, the notes are:

```
  <note>
    <pitch>
      <step>G</step>
      <octave>4</octave>
    </pitch>
    <duration>24</duration>
    <lyric>
      <syllabic>single</syllabic>
      <text>Dans</text>
    </lyric>
  </note>
  <note>
    <pitch>
      <step>C</step>
      <octave>5</octave>
    </pitch>
    <duration>24</duration>
    <lyric>
      <syllabic>single</syllabic>
      <text>un</text>
    </lyric>
  </note>
  <note>
    <pitch>
      <step>D</step>
      <octave>5</octave>
```

```
    </pitch>
    <duration>24</duration>
    <lyric>
      <syllabic>begin</syllabic>
      <text>som</text>
    </lyric>
  </note>
```

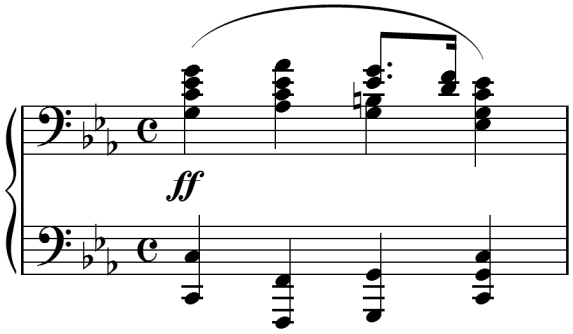
The actual text of the lyric is specified in the text element. A note may have multiple syllables, in which case the multiple syllabic/text element pairs should be separated by an elision element. Word extensions may be indicated by using the extend element, as in the “meil” syllable above.

Multiple verses are indicating using multiple lyric elements. The number and name attributes can be used to distinguish them: <lyric number="1"> for the first verse, <lyric number="2"> for the second.

MusicXML has end-line and end-paragraph elements to support Standard MIDI File Lyric meta-events specified in RP-017. These are used for karaoke and similar applications. Elements for humming and laughing may also be included, though they do not have MIDI equivalents. These lyric elements have not yet been implemented in most MusicXML software.

Multi-Part Music

While monophonic instruments like trumpet, flute, and voice move along one note at a time, instruments like the piano can have many musical lines at once. Take this simple example from the first bar of Frederic Chopin's Prelude, Op. 28, No. 20:



Within the piano part, there are two musical lines for the left hand and right hand, represented in the two staves. On the third beat of the bar, the right hand divides into two lines as well.

We mentioned earlier that the duration element in a note moves the MusicXML musical counter, and that a chord element keeps this counter from moving further. In order to represent parallel musical parts, we need to be able to move the musical counter forwards and backwards independently of notes. This is what the forward and backup elements let us do.

Let's say we have 4 divisions per quarter in this example. We could approach the divided parts in the right hand in two ways. Finale, for instance, can represent multiple parts using either the layer feature or the voice 1/voice 2 feature. When using layers, each independent part generally covers a complete measure. Say that the G and B-natural on beat 3 are in layer 2, with all the other notes in layer 1. After completing the last chord in layer 1, we would use the following to add the layer 2 notes:

```
<backup>
  <duration>16</duration>
</backup>
```

```
<forward>
  <duration>8</duration>
</forward>
<note>
  <pitch>
    <step>G</step>
    <octave>3</octave>
  </pitch>
  <duration>4</duration>
</note>
<note>
  <chord/>
  <pitch>
    <step>B</step>
    <octave>3</octave>
  </pitch>
  <duration>4</duration>
</note>
<forward>
  <duration>4</duration>
</forward>
```

The first backup element moves the counter back to the beginning of the measure. The forward element that follows serves as an invisible half rest. The two note elements provide the quarter note chord. The last forward element serves as an invisible quarter note rest to move the music counter up to the end of the measure.

On the other hand, we could use the voice 1/voice 2 feature on the third beat to indicate a temporary split in the parts. In this case, we would have something like:

```
<note>
  <pitch>
    <step>G</step>
    <octave>3</octave>
  </pitch>
  <duration>4</duration>
</note>
<note>
  <chord/>
  <pitch>
    <step>B</step>
    <octave>3</octave>
  </pitch>
  <duration>4</duration>
</note>
<backup>
  <duration>4</duration>
</backup>

<note>
  <pitch>
    <step>E</step>
    <alter>-1</alter>
    <octave>4</octave>
  </pitch>
  <duration>3</duration>
</note>
<note>
```

```
    <chord/>
    <pitch>
      <step>G</step>
      <octave>4</octave>
    </pitch>
    <duration>3</duration>
  </note>
  <note>
    <pitch>
      <step>D</step>
      <octave>4</octave>
    </pitch>
    <duration>1</duration>
  </note>
  <note>
    <chord/>
    <pitch>
      <step>F</step>
      <octave>4</octave>
    </pitch>
    <duration>1</duration>
  </note>
```

Here, all that is needed is a backup command to go backup over the quarter note so that the dotted eighth and sixteenth are positioned properly.

MusicXML can handle either type of multi-part writing. In Finale, most users find it easier to use layers rather than voice 1/voice 2, so our Finale-based examples will use that idiom more often. But the choice of what to use is up to you, based on what fits your software best.

In both cases, we would then use a <backup> element with a duration of 16 to move from the end of the first staff to the beginning of the second staff, where we can continue with the left-hand line. MusicXML offers more features for multi-part and multi-staff writing that will be described in later sections, but the elements listed here are all that is needed to multi-part sound output.

Repeats

Repeats and endings are represented by the <repeat> and <ending> elements with a <barline>, as defined in the barline.mod file.

In regular measures, there is no need to include the <barline> element. It is only needed to represent repeats, endings, and graphical styles such as double barlines.

A forward repeat mark is represented by a left barline at the beginning of the measure (following the attributes element, if there is one):

```
  <barline location="left">
    <bar-style>heavy-light</bar-style>
    <repeat direction="forward"/>
  </barline>
```

The repeat element is what is used for sound generation; the bar-style element only indicates graphic appearance.

Similarly, a backward repeat mark is represented by a right barline at the end of the measure:

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

While repeats can have forward or backward direction, endings can have three different type attributes: start, stop, and discontinue. The start value is used at the beginning of an ending, at the beginning of a measure. A typical first ending starts like this:

```
<barline location="left">
  <ending type="start" number="1"/>
</barline>
```

The stop value is used when the end of the ending is marked with a downward hook, as is typical for a first ending. It is usually used together with a backward repeat at the end of a measure:

```
<barline location="right">
  <bar-style>light-heavy</bar-style>
  <ending type="stop" number="1"/>
  <repeat direction="backward"/>
</barline>
```

The discontinue value is typically used for the last ending in a set, where there is no downward hook to mark the end of an ending:

```
<barline location="right">
  <ending type="discontinue" number="2"/>
</barline>
```

Sound Suggestions

Musical scores abound with ambiguous notations for dynamics, tempo, and other musical elements. To automatically generate a MIDI or other sound file, some value must be used for dynamics or tempo. MusicXML defaults to a MIDI velocity of 90 (roughly a forte); no default tempo is specified.

Sound suggestion elements and attributes guide the creation of a sound file. Most of the sound suggestions are found in the sound element, defined in the direction.mod file. Tempo is specified in quarter notes per minute. Dynamics are specified as a percentage of the standard MusicXML forte velocity. For example, the following sound element specifies a tempo of quarter note = 88 with a MIDI velocity of 64:

```
<sound tempo="88" dynamics="71"/>
```

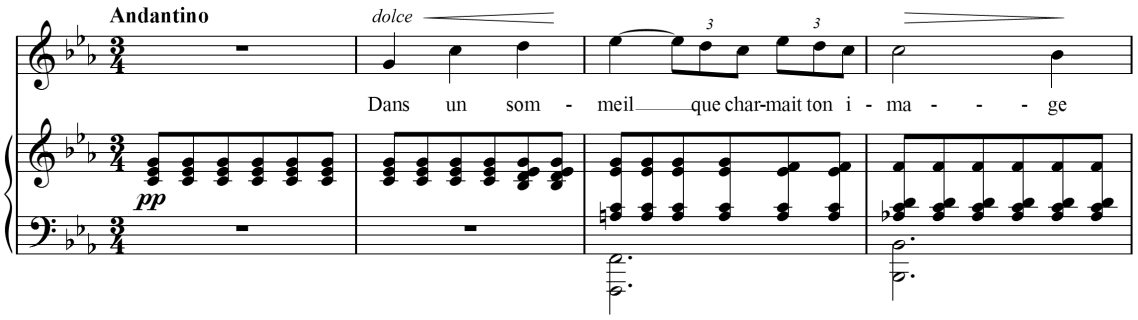
The other attributes for the sound element are described in the direction.mod file. Sound suggestions are also available for grace notes (the steal-time-previous, steal-time-following, and make-time attributes, defined in the note.mod file) and for ornaments (the trill-sound entity for trills and other ornaments, defined in the common.mod file).

Notation Basics in MusicXML

MIDI represents musical performance information, but leaves out a great deal of information about music notation. MusicXML represents this information, making it much more useful than MIDI for interchange between notation programs. In this section we describe the main elements used to represent music notation that go far beyond what is represented in MIDI files.

How Music Looks vs. How Music Sounds

Let us look again at the example we used in the previous section - the first four bars of "Après un rêve" by Gabriel Fauré:



Clearly our discussion of the MIDI-compatible portion of MusicXML left out many things represented in this music. Where are the tempo and dynamic markings: the Andantino, pp, dolce, crescendo and diminuendo wedges? Where are stem directions stored? The downstem on the initial G in the voice part is not what many programs would default to. How is the beaming represented, so that all the eighth notes are beamed together in the piano part, but separated into triplets in the voice part? How are the piano chords split between staves? How are accidentals indicated, including courtesy accidentals like the A-flat in the fourth bar?

A fundamental part of MusicXML is the distinction between elements that primarily represent the sound of the music versus those that represent its appearance. We discussed the sound elements in the previous section, and they are of great use to applications dealing with MIDI or other sound files. Now we discuss the elements for musical appearance, which are of great use to music notation applications.

Here is what the beginning of the voice part looks like for "Après un rêve," up to the end of the first measure:

```
<part id="P1">
  <measure number="1">
    <attributes>
      <divisions>24</divisions>
      <key>
        <fifths>-3</fifths>
        <mode>minor</mode>
      </key>
      <time>
        <beats>3</beats>
        <beat-type>4</beat-type>
      </time>
      <clef>
        <sign>G</sign>
        <line>2</line>
      </clef>
```

```
      </attributes>
      <direction directive="yes" placement="above">
        <direction-type>
          <words default-y="15" font-weight="bold">Andantino</words>
        </direction-type>
        <sound tempo="60"/>
      </direction>
      <note>
        <rest measure="yes"/>
        <duration>72</duration>
        <voice>1</voice>
      </note>
    </measure>
```

And here is what the beginning of the piano part looks like for "Après un rêve," up to the first chord in the piano part. We will discuss the appearance elements used in these two examples in the rest of this section.

```
  <part id="P2">
    <measure number="1">
      <attributes>
        <divisions>2</divisions>
        <key>
          <fifths>-3</fifths>
          <mode>minor</mode>
        </key>
        <time>
          <beats>3</beats>
          <beat-type>4</beat-type>
        </time>
        <staves>2</staves>
        <clef number="1">
          <sign>G</sign>
          <line>2</line>
        </clef>
        <clef number="2">
          <sign>F</sign>
          <line>4</line>
        </clef>
      </attributes>
      <direction placement="below">
        <direction-type>
          <dynamics default-x="129" default-y="-75">
            <pp/>
          </dynamics>
        </direction-type>
        <staff>1</staff>
        <sound dynamics="40"/>
      </direction>
      <note default-x="140">
        <pitch>
          <step>C</step>
          <octave>4</octave>
        </pitch>
        <duration>1</duration>
        <voice>1</voice>
        <type>eighth</type>
```

```
      <stem default-y="3">up</stem>
      <staff>1</staff>
      <beam number="1">begin</beam>
    </note>
    <note>
      <chord/>
      <pitch>
        <step>E</step>
        <alter>-1</alter>
        <octave>4</octave>
      </pitch>
      <duration>1</duration>
      <voice>1</voice>
      <type>eighth</type>
      <stem>up</stem>
      <staff>1</staff>
    </note>
    <note>
      <chord/>
      <pitch>
        <step>G</step>
        <octave>4</octave>
      </pitch>
      <duration>1</duration>
      <voice>1</voice>
      <type>eighth</type>
      <stem>up</stem>
      <staff>1</staff>
    </note>
```

Attributes

Staves

The staves element indicates the number of staves in a musical part, which in this case is 2 staves for the piano part. The staves element is optional. If it is not present, as is the case in the voice part, there is 1 staff for the part.

Clef

The clef element is used to indicate the clef for the staff. By specifying the clef's sign and its line, MusicXML handles both the common treble and bass clefs along with tenor, alto, percussion, tab, and older clefs. The treble clef definition indicates that the second line from the bottom of the staff is a G; the bass clef definition indicates that the fourth line from the bottom of the staff is an F. The number attribute indicates the staff number if the part has more than one staff.

The clef element may also contain a clef-octave-change element after the line element. This is used for clefs that are written either an octave higher or lower than sounding pitch. For example, the tenor line in choral music is usually written in treble clef, an octave higher than the notes actually sound.

The clef for this part would be represented as:

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

```
        <clef-octave-change>-1</clef-octave-change>
      </clef>
```

While attributes usually appear at the start of a measure, they can appear anywhere within the measure. Mid-measure clef changes are the main use for this feature.

Time

To represent common and cut time signatures, use the symbol attribute of the time element.

For common time, use:

```
<time symbol="common">
  <beats>4</beats>
  <beat-type>4</beat-type>
</time>
```

For cut time, use:

```
<time symbol="cut">
  <beats>2</beats>
  <beat-type>2</beat-type>
</time>
```

Without the symbol attribute, these time signatures would appear as 4/4 and 2/2, respectively.

Musical Directions

Musical directions are used for the expression marks in a musical score that are not clearly tied to a particular note. The beginning of the voice part in measure 2, for instance, looks like this:

```
<direction placement="above">
  <direction-type>
    <words default-x="15" default-y="15"
           font-size="9" font-style="italic">dolce</words>
  </direction-type>
</direction>
<note default-x="27">
  <pitch>
    <step>G</step>
    <octave>4</octave>
  </pitch>
  <duration>24</duration>
  <voice>1</voice>
  <type>quarter</type>
  <stem default-y="6">down</stem>
  <lyric default-y="-80" number="1">
    <syllabic>single</syllabic>
    <text>Dans</text>
  </lyric>
</note>
<direction placement="above">
  <direction-type>
    <wedge default-y="20" type="crescendo"/>
  </direction-type>
  <offset>-8</offset>
</direction>
```

This indicates that the "dolce" mark starts a little more than one space before the first note in a 9 point italic font, while the crescendo wedge starts two-thirds of the way between the first and second notes in the measure. The placement attribute is used to indicate whether the directions go

above or below the staff. The default-x and default-y attributes provide more precise positioning, and use units of tenths of interline space. For these elements, the default-x attribute is measured from the start of the current measure (for other elements, it is measured from the left-hand side of the note or the musical position within the bar). The default-y element is measured from the top barline of the staff. The offset element measures horizontal distance in terms of divisions, just like the duration element.

If two directions go together, they can be linked by having multiple direction-type elements within a single direction. A MusicXML direction-type can contain many different elements, including words, dynamics, wedge, segno, coda, rehearsal, dashes, pedal, metronome, and octave-shift (for 8va and related marks).

Elements that continue over time have a type attribute to indicate the start and end points, as well as positioning when continued over system breaks. For a wedge, the type may be crescendo, diminuendo, stop, or continue. For octave-shift, the choices are up, down, stop, or continue. The shift indicates whether the note appears up or down from the sounding pitch, so the start of an 8va has a type of down. For dashes, the choices are start or stop. For pedal, the choices are start, stop, sostenuto, continue, or change. The continue value is used with pedal lines to indicate a pedal lift and retake.

Note Appearance

Symbolic Note Types

Given the duration of a note and the divisions attribute, a program can usually infer the symbolic note type (e.g. quarter note, dotted-eighth note). However, it is much easier for notation programs if this is represented explicitly, rather than making the program infer the correct symbolic value. In some cases, the intended note duration does not match what is written, be it some of Bach's dotted notations, notes inégales, or jazz swing rhythms.

The type element is used to indicate the symbolic note type, such as quarter, eighth, or 16th. MusicXML symbolic note types range from 1024th notes to maxima notes: 1024th, 512th, 256th, 128th, 64th, 32nd, 16th, eighth, quarter, half, whole, breve, long, and maxima. The type element may be followed by one or more empty dot elements to indicate dotted notes.

Tuplets

The time-modification element is used to make it easier for applications to handle tuplets properly. For a normal triplet, this would look like:

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

This indicates that three notes are placed in the time usually allotted for two notes.

There is an optional normal-type element that is used when the type of the note does not match the type of the normal-notes in the triplet. Say you have an eighth note triplet, but instead of three eighth notes, you have a quarter note and eighth note instead. Without a normal-type element, software that reads the quarter note in the tuplet will likely assume that this is starting a quarter-note triplet, not an eighth note triplet. In this case, the symbolic type and tuplet would be encoded as:

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

```
    <normal-type>eighth</normal-type>
  </time-modification>
```

The time-modification element cannot represent all aspects of tuplets, such as detailed formatting and where nested tuplets begin and end. It is recommended that the tuplet element also be used to notate where tuplets begin and end, along with any additional formatting details that may be needed. The tuplet element is a child of the notations element described below.

Stems

Stem direction is represented with the stem element, whose value can be up, down, none, or double. For up and down stems, the default-y attribute represents where the stem ends, measured in tenths of interline space from the top line of the staff.

Beams

Beams are represented by beam elements. Their value can be begin, continue, end, forward hook, and backward hook. Each element has a beam-level attribute which ranges from 1 to 8 for eighth-note to 256th-note beams.

Accidentals

The accidental element represents actual notated accidentals. The most common values are sharp, flat, natural, double-sharp, and flat-flat. Many microtonal accidental values are also available. An accidental element has optional courtesy and editorial attributes to indicate courtesy and editorial accidentals. The bracket, parentheses, and size attributes offer more precise visual representations for these types of accidentals.

Notations

Many additional elements can be associated with a note. In MusicXML, these are collected under the notations object. Tied notes, slurs, tuplets, fermatas, and arpeggios are represented by top-level children of the notations element. Dynamics, ornaments, articulations, and technical indications specific to particular instruments are also top-level children of the notations element. A staccato mark would then be placed within the articulations element.

The tied element represents the visual part of a tie, and the tuplet element represents the visual part of a tuplet. The tie element affects the sound, and the time-modification affects placement, but the tied and tuplet elements indicate that there is something to see on the score indicating the tie or tuplet. (With ties, the two nearly always go together, but with tuplets this is not the case.) The second E-flat in measure 3 of the voice part, which is the end of a tie and start of a tuplet, is represented as:

```
<note>
  <pitch>
    <step>E</step>
    <alter>-1</alter>
    <octave>5</octave>
  </pitch>
  <duration>4</duration>
  <tie type="stop"/>
  <voice>1</voice>
  <type>eighth</type>
  <time-modification>
    <actual-notes>3</actual-notes>
    <normal-notes>2</normal-notes>
```

```
    </time-modification>
    <stem>down</stem>
    <notations>
      <tied type="stop"/>
      <tuplet bracket="no" number="1" placement="above"
              type="start"/>
    </notations>
  </note>
```

Slur, tied, and tuplet elements all have a number attribute to distinguish overlapping graphical elements.

Full details for all the different notations can be found in their definitions in the note.mod file.

Multi-Part Music

MusicXML contains two elements to help distinguish what is happening in multi-part music: the voice and staff elements.

A staff element should be used wherever possible in multi-staff music like piano parts. Note, forward, and direction elements can all include a staff element. The first cross-staff chord in measure 3 of the piano part is represented as:

```
<note default-x="26">
  <pitch>
    <step>A</step>
    <octave>3</octave>
  </pitch>
  <duration>1</duration>
  <voice>1</voice>
  <type>eighth</type>
  <accidental>natural</accidental>
  <stem default-y="91">up</stem>
  <staff>2</staff>
  <beam number="1">begin</beam>
</note>
<note default-x="26">
  <chord/>
  <pitch>
    <step>C</step>
    <octave>4</octave>
  </pitch>
  <duration>1</duration>
  <voice>1</voice>
  <type>eighth</type>
  <stem>up</stem>
  <staff>2</staff>
</note>
<note default-x="26">
  <chord/>
  <pitch>
    <step>E</step>
    <alter>-1</alter>
    <octave>4</octave>
  </pitch>
  <duration>1</duration>
  <voice>1</voice>
  <type>eighth</type>
```

```
    <stem>up</stem>
    <staff>1</staff>
  </note>
  <note default-x="26">
    <chord/>
    <pitch>
      <step>G</step>
      <octave>4</octave>
    </pitch>
    <duration>1</duration>
    <voice>1</voice>
    <type>eighth</type>
    <stem>up</stem>
    <staff>1</staff>
  </note>
```

The voice element helps keep track of multiple independent voice parts. Specifying the voice makes it much easier to import MusicXML files into other programs that handle multiple voices, such as Finale with its layer feature. In the piano chord above, having all the notes be part of a chord in voice 1 but with different staff elements ensures that this will be represented as a cross-staff chord. After completing the six chords in the right-hand part, the left-hand chord is represented as a chord in voice 2, using:

```
  <backup>
    <duration>6</duration>
  </backup>
  <note default-x="26">
    <pitch>
      <step>F</step>
      <octave>1</octave>
    </pitch>
    <duration>6</duration>
    <voice>2</voice>
    <type>half</type>
    <dot/>
    <stem default-y="-105.5">down</stem>
    <staff>2</staff>
  </note>
  <note default-x="26">
    <chord/>
    <pitch>
      <step>F</step>
      <octave>2</octave>
    </pitch>
    <duration>6</duration>
    <voice>2</voice>
    <type>half</type>
    <dot/>
    <stem>down</stem>
    <staff>2</staff>
  </note>
```

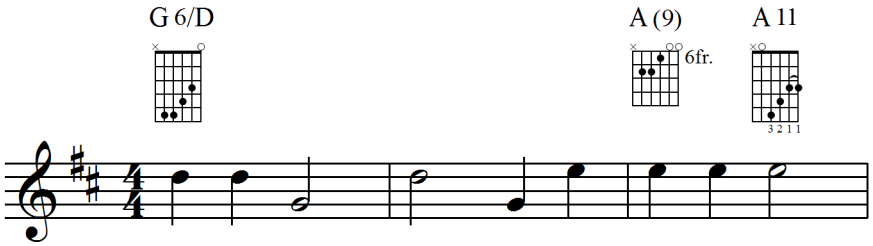
Chord Symbols and Diagrams in MusicXML

Much contemporary sheet music makes use of chord symbols and chord diagrams. These notations of musical harmony are found in such different types of sheet music as lead sheets, piano / vocal / guitar arrangements, worship music, and jazz big-band charts.

MusicXML's harmony element provides a rich description of both harmonic content and the appearance of chord symbols and diagrams. It can be used both for functional harmony analysis as well as chord symbols. Chord symbols and diagrams are the most common use, and that is what we focus on here.

Chord Symbols

Here is a three-bar example of a simple lead sheet. It contains the melody together with chord symbols and diagrams for how to play the chords on a guitar:



The first chord is a G major sixth chord with the fifth (D) in the bass. The second chord is notated as an A major chord with an added ninth degree. Another analysis might be to call it a dominant ninth chord with a missing seventh degree. MusicXML supports both types of analysis. For this example, we follow the written chord diagram notation. The third chord, an A11, will be discussed in the chord diagram section, as it includes both fingerings and a barre symbol.

Here is how the first G6 chord symbol is represented in the MusicXML file, omitting the chord diagram for the time being:

```
<harmony default-y="100">
  <root>
    <root-step>G</root-step>
  </root>
  <kind halign="center" text="6">major-sixth</kind>
  <bass>
    <bass-step>D</bass-step>
  </bass>
</harmony>
```

Each chord symbol has at least two elements: a <root> element to indicate the root of the chord, and a kind element to indicate the type of the chord. Here, we have a root of G and a kind of major-sixth. MusicXML 3.1 supports 33 different kind elements, listed in the direction.mod file. The kind element has a text attribute that indicates that the chord is displayed as G6, not as Gmaj6, GM6, or other spelling that could represent the same chord. This symbol also indicates the bass of the chord, represented using the bass element.

Both the root and the bass element divide the pitch into step and alter elements, similar to how the pitch element works. The root element uses the root-step and root-alter elements, while the bass element uses the bass-step and bass-alter elements. There is no element that corresponds to the octave element for pitch, since this information is not considered part of the harmonic analysis or the chord symbol.

MusicXML can represent all sorts of alterations to the built-in 33 kinds of chords. Degrees in the chord can be added, subtracted (e.g. “no 3”), or altered (e.g. “#5”). Here is how the second A(9) chord symbol is presented in MusicXML, using an added degree:

```
<harmony default-y="100">
  <root>
    <root-step>A</root-step>
  </root>
  <kind halign="center" parentheses-degrees="yes">major</kind>
  <degree>
    <degree-value>9</degree-value>
    <degree-alter>0</degree-alter>
    <degree-type text="">add</degree-type>
  </degree>
</harmony>
```

The degree element shows that we are adding an unaltered 9th degree to the chord, notating it with just the degree value (not as “add 9”) and with the added degrees in parentheses. The value of the degree-type element can be add, alter, or subtract. If the degree-type value is alter or subtract, the degree-alter value is relative to the degree already in the chord based on its kind element. If the degree-type value is add, the degree-alter value is relative to a dominant chord.

MusicXML’s harmony element contains additional features to support formatting and harmonic analysis. Full details are available in the direction.mod file.

Chord Diagrams

Chord diagrams, also known as chord frames, are used to indicate how a chord is played on a fretted instrument such as the guitar. The vertical lines in the chord diagrams represent strings, while the horizontal spaces represent frets. An x above a string indicates that the string is muted, while an o above a string represents an open string.

MusicXML uses the frame element to represent chord diagrams. Let us look at the harmony element for the first G6 chord again, this time including the frame element for the chord diagram:

```
<harmony default-y="100">
  <root>
    <root-step>G</root-step>
  </root>
  <kind halign="center" text="6">major-sixth</kind>
  <bass>
    <bass-step>D</bass-step>
  </bass>
  <frame default-y="83" halign="center"
         relative-x="5" unplayed="x" valign="top">
    <frame-strings>6</frame-strings>
    <frame-frets>5</frame-frets>
    <frame-note>
      <string>5</string>
      <fret>5</fret>
    </frame-note>
    <frame-note>
      <string>4</string>
      <fret>5</fret>
    </frame-note>
    <frame-note>
      <string>3</string>
      <fret>4</fret>
```

```
      </frame-note>
      <frame-note>
        <string>2</string>
        <fret>3</fret>
      </frame-note>
      <frame-note>
        <string>1</string>
        <fret>0</fret>
      </frame-note>
    </frame>
  </harmony>
```

The frame element starts with frame-strings and frame-fret elements to indicate the size of the frame. Each string that is played is then represented with a frame-note element. The lowest string, string 6, is muted in this diagram, so it has no corresponding frame-note element. The highest string, string 1, is open, so its fret value is set to 0. The frame element's unplayed attribute indicates what to display above a string like string 6 that has no frame-note element.

The positioning attributes indicate the vertical position of both the chord symbol text in the harmony element and the chord diagram in the frame element. The valign attribute of the frame element indicates that the default-y position represents the top of the chord diagram. The halign attributes indicate that both the chord symbol text and chord diagram are center-aligned.

In the second chord diagram, the first fret we see in the diagram corresponds to the sixth fret on the guitar. This is represented in MusicXML by the first-fret element. Unlike the other diagrams, this one shows only four frets.

Here is the frame element for the second chord diagram:

```
<frame default-y="83" halign="center"
       unplayed="x" valign="top">
  <frame-strings>6</frame-strings>
  <frame-frets>4</frame-frets>
  <first-fret location="right" text="6fr.">6</first-fret>
  <frame-note>
    <string>5</string>
    <fret>7</fret>
  </frame-note>
  <frame-note>
    <string>4</string>
    <fret>7</fret>
  </frame-note>
  <frame-note>
    <string>3</string>
    <fret>6</fret>
  </frame-note>
  <frame-note>
    <string>2</string>
    <fret>0</fret>
  </frame-note>
  <frame-note>
    <string>1</string>
    <fret>0</fret>
  </frame-note>
</frame>
```

The text attribute of the first-fret element shows how the text is displayed, and the location attribute indicates where it is displayed: to the right of the chord frame.

The third chord symbol adds fingerings and a barre indication. Here is the MusicXML harmony element for this symbol:

```
<harmony default-y="100">
  <root>
    <root-step>A</root-step>
  </root>
  <kind halign="center" text="11">dominant-11th</kind>
  <frame default-y="83" halign="center"
         unplayed="x" valign="top">
    <frame-strings>6</frame-strings>
    <frame-frets>5</frame-frets>
    <frame-note>
      <string>5</string>
      <fret>0</fret>
    </frame-note>
    <frame-note>
      <string>4</string>
      <fret>6</fret>
      <fingering>3</fingering>
    </frame-note>
    <frame-note>
      <string>3</string>
      <fret>4</fret>
      <fingering>2</fingering>
    </frame-note>
    <frame-note>
      <string>2</string>
      <fret>3</fret>
      <fingering>1</fingering>
      <barre type="start"/>
    </frame-note>
    <frame-note>
      <string>1</string>
      <fret>3</fret>
      <fingering>1</fingering>
      <barre type="stop"/>
    </frame-note>
  </frame>
</harmony>
```

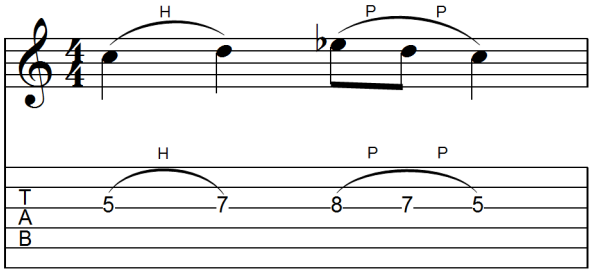
The fingering element is used to indicate the fingering for each string in the diagram that is neither muted nor open. The barre element is used to indicate where a barre symbol starts and stops within the frame. The type is “start” for the lowest-pitched string and “stop” for the highest-pitched string. This corresponds to the left-to-right order from low to high strings that is used in chord diagrams.

Tablature in MusicXML

Tablature notation provides a direct guideline to the strings and frets used to play music on a guitar or other fretted instrument, often at the expense of precise rhythmic information. In MusicXML, this rhythmic information needs to be specified, although the display is likely to be hidden on a tab part. The main things that need to be added are the fret and string information, the details of the how the strings are tuned, and techniques specific to guitars and other related instruments.

Fret and String

Here is a simple one-measure guitar part example that we will use to illustrate the basic MusicXML tablature features, using the standard 6-string guitar tuning:



The fret and string information needed to generate tablature for guitar and other stringed instruments is handled the same way as technical indications for other instruments, such as piano fingerings and violin bowings. The technical element contains these types of notations, and two of its component elements represent the note's fret and string. Frets are numbered starting at 0 for the open string. Strings are numbered starting at 1 for the highest string on the instrument.

The fret and string for first note in this example are represented using:

```
<technical>
  <string>3</string>
  <fret>5</fret>
</technical>
```

String Tuning

An attributes element may include a staff-details element to specify the details of a tab staff. The staff-lines element specifies the number of lines on a tablature staff, usually one for each string. Staff tunings are described with the staff-tuning and capo elements. TAB is one of the values available for clef elements. The print-object attribute of the key element is used to indicate that the key signature should not be displayed on this staff.

The tab part in our example begins with the following attributes:

```
<attributes>
  <divisions>2</divisions>
  <key print-object="no">
     <fifths>0</fifths>
     <mode>major</mode>
  </key>
  <clef>
    <sign>TAB</sign>
    <line>5</line>
```

```
      </clef>
      <staff-details>
        <staff-lines>6</staff-lines>
        <staff-tuning line="1">
          <tuning-step>E</tuning-step>
          <tuning-octave>2</tuning-octave>
        </staff-tuning>
        <staff-tuning line="2">
          <tuning-step>A</tuning-step>
          <tuning-octave>2</tuning-octave>
        </staff-tuning>
        <staff-tuning line="3">
          <tuning-step>D</tuning-step>
          <tuning-octave>3</tuning-octave>
        </staff-tuning>
        <staff-tuning line="4">
          <tuning-step>G</tuning-step>
          <tuning-octave>3</tuning-octave>
        </staff-tuning>
        <staff-tuning line="5">
          <tuning-step>B</tuning-step>
          <tuning-octave>3</tuning-octave>
        </staff-tuning>
        <staff-tuning line="6">
          <tuning-step>E</tuning-step>
          <tuning-octave>4</tuning-octave>
        </staff-tuning>
      </staff-details>
    </attributes>
```

Hammer-ons and Pull-offs

Contemporary guitar notation contains many elements idiomatic to the guitar (and specifically the electric guitar). While elements like harmonics and bends have sporadic support in current MusicXML software, the hammer-on and pull-off are both supported in Finale and other applications.

These elements are represented as technical indications that often go together with a notated slur. In the first half of the bar, a single hammer-on goes together with a single slur:

```
    <note default-x="82">
      <pitch>
        <step>C</step>
        <octave>4</octave>
      </pitch>
      <duration>2</duration>
      <voice>1</voice>
      <type>quarter</type>
      <stem>none</stem>
      <notations>
        <technical>
          <hammer-on number="1" type="start">H</hammer-on>
          <string>3</string>
          <fret>5</fret>
        </technical>
        <slur number="1" placement="above" type="start"/>
      </notations>
    </note>
```

```
<note default-x="177">
  <pitch>
    <step>D</step>
    <octave>4</octave>
  </pitch>
  <duration>2</duration>
  <voice>1</voice>
  <type>quarter</type>
  <stem>none</stem>
  <notations>
    <technical>
      <hammer-on number="1" type="stop"/>
      <string>3</string>
      <fret>7</fret>
    </technical>
    <slur number="1" type="stop"/>
  </notations>
</note>
```

But in the second half of the bar, a single slur goes together with two pull-offs:

```
<note default-x="272">
  <pitch>
    <step>E</step>
    <alter>-1</alter>
    <octave>4</octave>
  </pitch>
  <duration>1</duration>
  <voice>1</voice>
  <type>eighth</type>
  <stem>none</stem>
  <notations>
    <technical>
      <pull-off number="1" type="start">P</pull-off>
      <string>3</string>
      <fret>8</fret>
    </technical>
    <slur number="1" placement="above" type="start"/>
  </notations>
</note>
<note default-x="330">
  <pitch>
    <step>D</step>
    <octave>4</octave>
  </pitch>
  <duration>1</duration>
  <voice>1</voice>
  <type>eighth</type>
  <stem>none</stem>
  <notations>
    <technical>
      <pull-off number="1" type="stop"/>
      <pull-off number="1" type="start">P</pull-off>
      <string>3</string>
      <fret>7</fret>
    </technical>
  </notations>
</note>
```

```
<note default-x="389">
  <pitch>
    <step>C</step>
    <octave>4</octave>
  </pitch>
  <duration>2</duration>
  <voice>1</voice>
  <type>quarter</type>
  <stem>none</stem>
  <notations>
    <technical>
      <pull-off number="1" type="stop"/>
      <string>3</string>
      <fret>5</fret>
    </technical>
    <slur number="1" type="stop"/>
  </notations>
</note>
```

Percussion in MusicXML

Percussion instruments can be either pitched or unpitched. Percussion instruments with definite pitch, such as timpani or mallet instruments, are handled using normal musical notation.

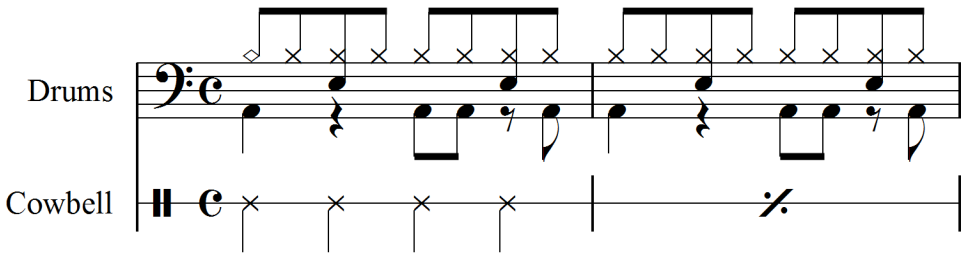
When the instrument has no definite pitch, like a bass drum or snare drum, notation gets stretched. The use of vertical space to represent pitch instead is used to represent different instruments. Yet sometimes we have more percussion instruments handled by one player than can fit into the lines on a staff. In other cases, we may have only one unpitched instrument to notate, where the vertical space on a 5-line staff would be wasted.

MusicXML allows percussion music to be represented both as heard and as notated by use of several features:

- Representing unpitched instruments visually by specifying placement on the staff.
- As with tablature, staves may be more or less than 5 lines high.
- Specifying multiple instruments per part
- Specifying the MIDI playback for each instrument (for instance, by a single MIDI pitch in a percussion kit).
- Alternate notehead shapes let multiple instruments to share a single line on the staff.

Unpitched Notes

To illustrate MusicXML's percussion features, here's a two-bar example for two players: one on a drum kit, and one on cowbell:



In the drum part, the top space (B in bass clef) is used for the cymbal (diamond notehead) and hi-hat (x notehead). The E space is used for the snare drum, and the bottom A space is used for the bass drum. The cowbell player has only one instrument, so it is represented on a one-line staff.

Since these notes have no definite pitch, it would be misleading to represent them using the pitch element. An analysis program looking for a series of repeated B's should not return this piece of music. Neither should a program looking for a series of repeated F-sharps, based on the General MIDI pitch for a closed hi-hat.

Instead, we represent percussion and other unpitched instruments with the unpitched element. As with rests, there are display-step and display-octave elements to indicate where the note should appear on the staff, based on the current clef. So the cymbal and hi-hat notes on this bass clef staff are represented as:

```
<unpitched>
  <display-step>B</display-step>
  <display-octave>3</display-octave>
</unpitched>
```

Percussion clef is treated like treble clef when determining the display-step and display-octave. So if this part had been notated in percussion clef instead of bass clef, the resulting MusicXML would be:

```
<unpitched>
  <display-step>G</display-step>
  <display-octave>5</display-octave>
</unpitched>
```

Staff Lines

The cowbell part is an example where one player has one instrument, so the part is notated on one line both for clarity and saving space. The single line is specified within the attributes element for the part:

```
<attributes>
  <divisions>2</divisions>
  <key>
    <fifths>0</fifths>
    <mode>major</mode>
  </key>
  <time symbol="common">
    <beats>4</beats>
    <beat-type>4</beat-type>
  </time>
  <clef>
    <sign>percussion</sign>
  </clef>
  <staff-details>
    <staff-lines>1</staff-lines>
  </staff-details>
</attributes>
```

How do we determine the display-step and display-octave for a one-line staff? Since the percussion clef is treated like a treble clef, the G in octave 4 is on the second line of the staff. For this one-line staff, that G is one imaginary line above the staff (for a 2-line staff, it's the top line, and the middle line on a 3-line staff). Therefore the unpitched elements for the cowbell part are:

```
<unpitched>
  <display-step>E</display-step>
  <display-octave>4</display-octave>
</unpitched>
```

Multiple Instruments Per Part

Percussion parts are one case of many where multiple instruments are sharing the same part. This is represented in MusicXML by including more than one score-instrument element within a score-part. Each note can then contain an instrument element that refers back to the id of the score-instrument. Note that the score-instrument id must be unique throughout the entire piece. Two score-part elements cannot each have a score-instrument element with the same id.

To represent MIDI playback for each instrument, we add a midi-instrument element for each score-instrument. In this case, we are using a General MIDI instrument, so we include a midi-channel of 10 for each instrument. We then use the midi-unpitched element to indicate the MIDI pitch that corresponds to a particular sound in a General MIDI drum kit.

The part-list for our two-bar example looks like this:

```
<part-list>
```

```
    <score-part id="P1">
      <part-name>Drums</part-name>
      <score-instrument id="P1-X4">
         <instrument-name>Snare Drum</instrument-name>
       </score-instrument>
       <score-instrument id="P1-X2">
         <instrument-name>Kick Drum</instrument-name>
       </score-instrument>
       <score-instrument id="P1-X13">
         <instrument-name>Crash Cymbal</instrument-name>
       </score-instrument>
       <score-instrument id="P1-X6">
         <instrument-name>Hi-Hat%g Closed</instrument-name>
       </score-instrument>
       <midi-instrument id="P1-X4">
         <midi-channel>10</midi-channel>
         <midi-program>1</midi-program>
         <midi-unpitched>39</midi-unpitched>
       </midi-instrument>
       <midi-instrument id="P1-X2">
         <midi-channel>10</midi-channel>
         <midi-program>1</midi-program>
         <midi-unpitched>37</midi-unpitched>
       </midi-instrument>
       <midi-instrument id="P1-X13">
         <midi-channel>10</midi-channel>
         <midi-program>1</midi-program>
         <midi-unpitched>50</midi-unpitched>
       </midi-instrument>
       <midi-instrument id="P1-X6">
         <midi-channel>10</midi-channel>
         <midi-program>1</midi-program>
         <midi-unpitched>43</midi-unpitched>
       </midi-instrument>
     </score-part>
    <score-part id="P2">
      <part-name>Cowbell</part-name>
      <score-instrument id="P2-X1">
        <instrument-name>Cowbell</instrument-name>
        <instrument-sound>metal.bells.cowbell</instrument-sound>
      </score-instrument>
      <midi-instrument id="P2-X1">
        <midi-channel>10</midi-channel>
        <midi-program>1</midi-program>
        <midi-unpitched>57</midi-unpitched>
      </midi-instrument>
    </score-part>
  </part-list>
```

Each note then includes an unpitched element to show where the note is on the staff, and an instrument element to indicate which instrument is used and how to play it back using MIDI. The first kick drum note looks like this:

```
      <note default-x="78">
        <unpitched>
          <display-step>A</display-step>
          <display-octave>2</display-octave>
```

```
      </unpitched>
      <duration>2</duration>
      <instrument id="P1-X2"/>
      <voice>2</voice>
      <type>quarter</type>
      <stem default-y="-65.5">down</stem>
    </note>
```

Finale creates these instrument names like "P1-X2" to make it easy to generate unique names for each score instrument. It would be better to use more readable names that are still unique across the entire piece.

Notehead Shapes

The one remaining task is to specify the alternate noteheads that distinguish the hi-hat from the cymbal. While the MusicXML playback can distinguish these by the use of different instruments, a drummer will certainly appreciate having different notehead shapes for different instrument on the same line.

The MusicXML notehead element specifies these different shapes. Values can be slash, triangle, diamond, square, cross, x, circle-x, inverted triangle, arrow down, arrow up, circled, slashed, back slashed, normal, cluster, circle dot, left triangle, rectangle, none, do, re, mi, fa, fa up, so, la, ti, and other. Enclosed shapes like normal, diamond, triangle, and square can use the filled attribute to indicate a filled or hollow shape.

The first two notes of the cymbal/hi-hat line look like this:

```
    <note default-x="78">
      <unpitched>
        <display-step>B</display-step>
        <display-octave>3</display-octave>
      </unpitched>
      <duration>1</duration>
      <instrument id="P1-X13"/>
      <voice>1</voice>
      <type>eighth</type>
      <stem default-y="40">up</stem>
      <notehead filled="no">diamond</notehead>
      <beam number="1">begin</beam>
    </note>
    <note default-x="109">
      <unpitched>
        <display-step>B</display-step>
        <display-octave>3</display-octave>
      </unpitched>
      <duration>1</duration>
      <instrument id="P1-X6"/>
      <voice>1</voice>
      <type>eighth</type>
      <stem default-y="40">up</stem>
      <notehead>x</notehead>
      <beam number="1">continue</beam>
    </note>
```

The filled attribute on the notehead element is also useful for multi-part piano music. There are many cases where, for instance, a downstem half note shares a hollow notehead with an upstem eighth note. The eighth note can specify that it uses an unfilled normal notehead, making things display correctly when moving back and forth between notation programs.

Measure Styles

Our cowbell part also shows the use of a one-bar repeat symbol. This and similar repeats and multimeasure rests are represented using the measure-style element. The music in the second bar should be specified just as in the first bar (four notes with display-step E, display-octave 4, and instrument P2-X1). At the start of the bar, we indicate the beginning of the one-measure repeat style:

```
<measure number="2">
  <attributes>
    <measure-style>
      <measure-repeat type="start">1</measure-repeat>
    </measure-style>
  </attributes>
```

Further details about the measure-style element can be found in the attributes.mod file.

Compressed .MXL Files

Regular, plain text MusicXML files can be very large - much larger than the original Finale and Sibelius files, or corresponding MIDI files. This was not a big problem for using MusicXML as an interchange format, but it inhibited MusicXML's use as a sheet music distribution format.

MusicXML 2.0 introduced a new compressed format which reduces MusicXML file sizes to roughly the same size as the corresponding MIDI files. The format uses zip compression and a special .mxl file suffix and Internet media type to identify files as MusicXML files vs. generic XML files. This section describes how the compressed format works.

Compressed File Format

MusicXML uses a zip-based XML format similar to that used by Open Office and many other XML formats. The MusicXML 3.1 zip file format is compatible with the zip format used by the java.util.zip package and Java JAR files. It is based on the Info-ZIP format described at:

http://www.info-zip.org/doc/appnote-19970311-iz.zip

The JAR file format is specified at:

http://docs.oracle.com/javase/8/docs/technotes/guides/jar/jar.html

Note that, compatible with JAR files, file names should be encoded in UTF-8 format.

Files with the zip container are compressed the DEFLATE algorithm. The DEFLATE Compressed Data Format (RFC 1951) is specified at:

http://www.ietf.org/rfc/rfc1951.txt

By making these choices, software can usually use the zip libraries available for Java and most other programming languages to create the compressed zip files.

File Suffixes and Media Types

The recommended file suffixes for MusicXML files are .mxl for compressed files and .musicxml for uncompressed files. Using unique file suffixes lets programs distinguish MusicXML files from generic XML files in a way that using the .xml suffix for uncompressed files does not.

Many XML editing tools such as XMLSpy and Oxygen now support the editing of zip-compressed XML files. All that is necessary is to let the editor application know that .mxl files are a zip-based format. The editor will then let you edit both the structure of the .mxl file as well as the .musicxml files that are contained within the zip archive.

For Internet use, MusicXML 3.1 has registered media types available for both compressed .mxl and uncompressed .musicxml files. The recommended media type for a compressed MusicXML file is:

application/vnd.recordare.musicxml

The recommended media type for an uncompressed MusicXML file is:

application/vnd.recordare.musicxml+xml

Zip Archive Structure

Following the example of many zip-based XML formats, MusicXML has a strictly defined way to determine where the root MusicXML file is within an archive. Each MusicXML zip archive has a file located at META-INF/container.xml. This container describes the starting point for the MusicXML version of the file, as well as alternate renditions such as PDF and audio versions of the musical score.

The container.xml file is defined by the container.dtd file. Given the simplicity of the format, there is no corresponding .xsd file at this time.

As an example, let us look at the Dichterliebe01.mxl file available on the MakeMusic web site. Here is its META-INF/container.xml file:

```
<?xml version="1.0" encoding="UTF-8">
<container>
    <rootfiles>
        <rootfile full-path="Dichterliebe01.xml"
                  media-type="application/vnd.recordare.musicxml+xml"/>
    </rootfiles>
</container>
```

The container element is the document element for this file. It contains a single rootfiles element. The rootfiles element in turn contains one or more rootfile elements. The MusicXML root must be described in the first rootfile element. The full-path attribute provides the path relative to the root folder of the zip file. A MusicXML file used as a rootfile may have score-partwise, score-timewise, or opus as its document element.

Additional rootfile elements can describe alternate versions such as PDF and audio files. Multiple rootfile elements can be distinguished by each one having a different media-type attribute value. For instance, if the Dichterliebe01.mxl file contained a PDF rendition as well as the MusicXML file, the container.xml file would look like this:

```
<?xml version="1.0" encoding="UTF-8">
<container>
    <rootfiles>
        <rootfile full-path="Dichterliebe01.xml"
                  media-type="application/vnd.recordare.musicxml+xml"/>
        <rootfile full-path="Dichterliebe01.pdf"
                  media-type="application/pdf"/>
    </rootfiles>
</container>
```

If no media-type value is present, a MusicXML file is assumed. However, if multiple rootfiles are present, it will probably clarify things to include the media-type attribute for all rootfile elements.

The zip archive can also include images that are referenced with the MusicXML image and credit-image elements, or other media that are referenced using the MusicXML link element. MusicXML 3.1 does not specify where these files need to be located in the zip file, but it is probably least confusing if images or other files of a similar type are grouped together in a subfolder within the zip archive.

The first file in the zip container should be a file named mimetype. The contents of this file should be the MIME media type string

application/vnd.recordare.musicxml

The mimetype file should be encoded in US-ASCII and must not be compressed or encrypted, and there must not be an extra field in its zip header. The contents of the mimetype file must not contain any laeding padding or white space, and must not begin with a byte order mark.

The mimetype file helps applications identify a compressed MusicXML file by reading the first few bytes of the file. EPUB and other zip-based formats also use a mimetype file for the same reason. Older versions of MusicXML before version 3.1 did not specify this mimetype file, so applications may still see containers without it.