

DEVELOPING A NEW WAY TO TRANSFER SHEET MUSIC VIA THE INTERNET

**By
Eric James Mosterd
B.A., University of South Dakota, 1999**

**A Thesis Submitted in Partial Fulfillment of
The Requirements for the Degree of
Master of Arts**

**Computer Science Department
In The Graduate School
The University of South Dakota
May 12, 2001**

The members of the Committee appointed to examine
the thesis of Eric James Mosterd find it
satisfactory and recommend that it be accepted.

Dr. Rich McBride, Chairperson

Dr. Gary Reeves

Dr. Dave Struckman-Johnson

Abstract

This thesis covers the issues regarding the transfer of sheet music via the Internet and the lack of a standard way to do so. Its primary focus is on both the need for such a standard and the subsequent development of a new, standard music notation markup language using eXtensible Markup Language (XML) called Extensible Music Notation Markup Language or EMNML. This new language can be used to send sheet music over the Internet without sacrificing its quality or content.

Acknowledgements

First, I'd like to thank my three advisers: Dr. Rich McBride, Dr. Gary Reeves, and Dr. Dave Struckman-Johnson. Special thanks go to Dr. Reeves for his help with music notation. I'd also like to thank the Computer Science Department for all of their help over the past six years. Additional thanks go to my beta testers. Also, I'd like to thank my neighbors for putting up with my saxophone playing during times of writer's block, or when I ran into a particularly stubborn problem while programming.

Finally, I'd like to thank my friends and family for their support throughout the entire thesis "ordeal".

Table of Contents

ABSTRACT	III
ACKNOWLEDGEMENTS	IV
TABLE OF CONTENTS	V
LIST OF FIGURES	VII
INTRODUCTION	1
<i>A Brief History of the Internet</i>	1
<i>Computers and Music</i>	3
<i>MIDI Explained</i>	3
<i>Advantages of MIDI</i>	6
<i>Drawbacks & Limitations of MIDI</i>	6
<i>Music & The Internet: The Economics of Music</i>	9
<i>Music & The Internet: A Business Example</i>	10
CURRENT RESEARCH	14
<i>Images</i>	14
<i>Portable Document Format</i>	14
<i>Notation Interchange File Format</i>	15
<i>Enigma Transportable Format (ETF)</i>	16
<i>Resource Interchange File Format</i>	17
<i>GUIDO Music Notation Language</i>	17
<i>abc</i>	20
<i>Common Music Notation</i>	20
<i>eXtensible Markup Language (XML)</i>	21
<i>Conclusion</i>	26
AN XML SOLUTION	27
DEVELOPING THE EMNML DTD	28
USABILITY TESTING	33
<i>Selecting The Users</i>	33
<i>Designing The Tests</i>	34
<i>Results</i>	35
<i>Conclusions</i>	38
CONVERTING MIDI TO EMNML	41
FUTURE RESEARCH	44
CONCLUSIONS	45
APPENDIX A: MIDI FILE FORMAT	46
<i>C Major Scale</i>	46
<i>MIDI Code For C Major Scale (in hex)</i>	46
<i>MIDI Code Explained</i>	46
APPENDIX B: EMNML DOCUMENTATION	49
<i>Introduction</i>	49
<i>EMNML Tags</i>	50
APPENDIX C: EMNML DATA TYPE DEFINITION (SPECIFICATION)	74
APPENDIX D: USABILITY TESTS	86
<i>Usability Testing Task 1 – Sheet Music</i>	86
<i>Usability Testing Task 1 – EMNML Representation</i>	86
<i>Usability Testing Task 2 – Sheet Music</i>	86

<i>Usability Testing Task 2 – EMNML Representation</i>	<i>87</i>
APPENDIX E: EMNML2MIDI SOURCE CODE	93
<i>Source Code For MIDI2TXT Header File.....</i>	<i>93</i>
<i>Source Code For MIDI2TXT File.....</i>	<i>100</i>
<i>MIDI2EMNML Wrapper Source Code.....</i>	<i>118</i>
BIBLIOGRAPHY.....	136

List of Figures

FIGURE 1: C MAJOR SCALE	4
FIGURE 2: MIDI CODES FOR C MAJOR SCALE.....	4
FIGURE 3: NOTE MIDI CODES EXPLAINED.....	5
FIGURE 4: SAMPLE SONG BEFORE CONVERSION TO MIDI	7
FIGURE 5: SAMPLE SONG AFTER CONVERSION TO MIDI	8
FIGURE 6: NMPA WORLDWIDE SALES STATISTICS FOR SHEET MUSIC.....	10
FIGURE 7: GUIDO CODE FOR FIRST MEASURE OF “QUARTETT”.....	19
FIGURE 8: SHEET MUSIC FOR “QUARTETT”	19
FIGURE 9: A CHORD IN COMMON MUSIC NOTATION.....	20
FIGURE 10: SAMPLE DOCUMENT TYPE DECLARATION FOR EXTERNAL DTD.....	22
FIGURE 11: REQUESTING AN EXTERNAL DOCUMENT TYPE DEFINITION	23
FIGURE 12: SAMPLE DOCUMENT TYPE DECLARATION FOR INTERNAL DTD	23
FIGURE 13: EXAMPLES OF MUSIC TERMINOLOGY IN DIFFERENT LANGUAGES	28
FIGURE 14: SLUR EXAMPLE.....	29
FIGURE 15: EMNML REPRESENTATION OF SLUR EXAMPLE	30
FIGURE 16: EMNML DETAILED DESCRIPTION.....	31
FIGURE 17: USABILITY TESTING TASK 1 - C MAJOR SCALE.....	34
FIGURE 18: USABILITY TESTING TASK 2 – <i>SONATA IN A</i> BY W.A. MOZART.....	35
FIGURE 19: USER ERRORS FOR TASK 1	36
FIGURE 20: USER ERRORS FOR TASK 2.....	37
FIGURE 21: NOTES IN OCTAVE 0 OF EMNML	37
FIGURE 22: USABILITY SURVEY RESULTS	38
FIGURE 23: EXAMPLES OF OLD SLUR TAG.....	39
FIGURE 24: EXAMPLE OF NEW SLUR TAG.....	40
FIGURE 25: OUTPUT FROM <i>MIDI2TXT</i> FOR C MAJOR SCALE	42
FIGURE 26: <i>MIDI2EMNML</i> OUTPUT FOR C MAJOR SCALE	43

Introduction

A Brief History of the Internet

During the 1970's, Xerox set up a little-known think tank called the Palo Alto Research Center (PARC). Xerox envisioned a paperless future, and since they were a paper document company, they wanted to make sure they had a position in this future.

PARC scientists created numerous innovations from the Graphical User Interface/What You See Is What You Get (GUI/WYSIWYG) computer, flat panel displays, and the rolling trackball to the color teal (actually caused by a failed experiment in chromatographic dye diffusion printing) and the little three-legged plastic disc that sits in the center of a pizza and to prevent the box from collapsing onto the cheese (the PARC scientists ate a lot of pizza). The PARC had two problems though: lack of insight from their management and the lack of intellectual protections for their experiments. For example, the scientist liked the color teal so much, that they started printing their business cards in the color, before they had sought patent protection. Later on, teal became one of the most popular colors, especially for automobiles (Dourish 1).

The lack of management insight at PARC is reflected in a historic incident where they allowed Steven Jobs and other Apple programmers to look at the Xerox Alto GUI/WYSIWYG computer. Using almost all of the ideas obtained from their visit, Apple later went on to fashion their entire computer product line after the Alto, basically stealing the technology.

Former PARC scientists have also “borrowed” ideas from experiments and research done at PARC. One famous example is the founders of Adobe, who based on research done at PARC, developed the Postscript printer language.

Though these technologies took away billions of dollars in potential revenue from Xerox, they pale in comparison to the “big one that got away”: Ethernet. In the early 1970s, PARC scientists developed the first network, in which they connected computers and printers together with wires. The computers could communicate through these wires, share resources and information, and print to one central printer. Had they patented the technology, Xerox would have been bigger than any other computer company. Instead, they showed their experiments to numerous people, resulting in many other companies, namely IBM, DEC, 3com (which was founded by former PARC scientists, and illustrates yet another case of former employees using their research at PARC to their companies advantage), and later, Cisco, becoming the leaders in networking.

What Xerox failed to see was that Ethernet would not only change computing history, but also human history itself. The local area network (LAN) invented by PARC would soon grow and be spread across a much greater geographical distance, and along with what was learned from the wide area network (WAN) of the Advanced Research Projects Agency Network (ARPANET), would eventually form the Internet.

The idea of a connecting computers together across great distances revolutionized the computing industry allowing universities, government agencies, and later, civilians to communicate with each other around the world. Schools could afford to buy expensive computers and justify the cost by sharing them with other institutions. Scientists could also collaborate worldwide, opening new possibilities in the field of research.

Eventually, more and more people connected to the Internet, causing explosive growth starting in the early 1990s and continuing today. People can share ideas and communicate on a global level. One of the most popular things for people to do online was to share music, and today it is the most popular pastime on the Internet, even surpassing the popularity of pornography. But before it got to where it is today, music had a long history in personal computing.

Computers and Music

Commodore Business Machines developed one of the first personal computer marketed to home consumers in 1981, which contained a built in synthesizer. Called the Sound Interface Device (SID), it gave people the power to write music and play sounds on an affordable computer. Building on this capability, four companies (Roland, Yamaha, Oberheim, and Sequential Circuits) formed a consortium and developed the Musical Instrument Digital Interface, or MIDI standard in 1983.

MIDI Explained

The original goal of MIDI was to connect instruments to layer sounds. It uses a set of binary instructions to send signals to a MIDI controller at 31,200 bits per second (or about 650 instructions per second). Though it has many uses, the most common use of MIDI is in music. Take the scale below, for example:

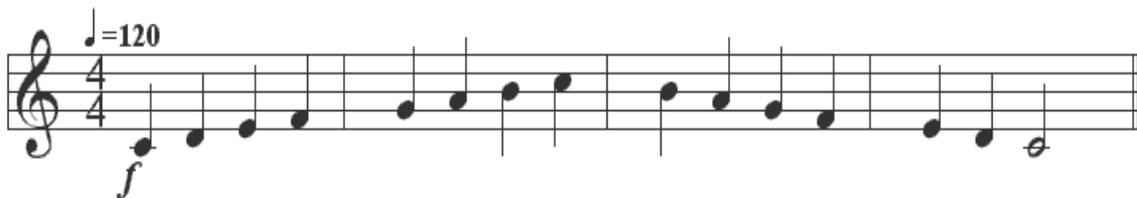


Figure 1: C Major Scale

This is how a human would see the scale, and below is what the scale would look like if it were encoded into the MIDI format:

4D	54	68	64	00	00	00	06	00	00	00	01	00	C0	4D	54	72	6B	00	00	01	25	00	FF
03	0D	43	20	4D	61	6A	6F	72	20	53	63	61	6C	65	00	FF	01	0F	42	79	20	45	72
69	63	20	4D	6F	73	74	65	72	64	00	FF	02	20	43	6F	70	79	72	69	67	68	74	20
A9	20	32	30	30	31	20	62	79	20	45	72	69	63	20	4D	6F	73	74	65	72	64	00	FF
02	13	41	6C	6C	20	52	69	67	68	74	73	20	52	65	73	65	72	76	65	64	00	FF	01
20	47	65	6E	65	72	61	74	65	64	20	62	79	20	4E	6F	74	65	57	6F	72	74	68	79
20	43	6F	6D	70	6F	73	65	72	00	B0	07	7F	00	B0	0A	40	00	FF	51	03	07	A1	20
00	FF	58	04	04	02	18	08	00	90	3C	5C	81	20	90	3C	00	20	90	3E	5C	81	20	90
3E	00	20	90	40	5C	81	20	90	40	00	20	90	41	5C	81	20	90	41	00	20	90	43	5C
81	20	90	43	00	20	90	45	5C	81	20	90	45	00	20	90	47	5C	81	20	90	47	00	20
90	48	5C	81	20	90	48	00	20	90	47	5C	81	20	90	47	00	20	90	45	5C	81	20	90
45	00	20	90	43	5C	81	20	90	43	00	20	90	41	5C	81	20	90	41	00	20	90	40	5C
81	20	90	40	00	20	90	3E	5C	81	20	90	3E	00	20	90	3C	5C	82	50	90	3C	00	00
FF	2F	00																					

Figure 2: MIDI Codes for C Major Scale

In the table above, the MIDI instructions are written in hexadecimal for simplification. Each two digit hexadecimal number represents a specific instruction. For example, the digits **90** signal the MIDI controller to turn on a particular note on the first channel. That note will stay on, or keep playing, until it is turned off by another instruction.

After the digits **90** comes the hexadecimal value for a specific note. To turn on the first note in the scale in Figure 1, the note on signal (**90**) would be sent, and then the

hexadecimal representation of middle C (3C). Figure 3 describes all of the note events found in Figure 2:

MIDI Codes	Description
90 3C 5C 81 20 90 3C 00 20	Turn on note on channel 1; the note is 3C or middle C and the velocity is 0x5C or 92, which is the notational equivalent of forte (<i>f</i>); pause 0x81 + 0x20 or 160 ticks; turn on note middle C on channel 1 with a velocity of 0 then pauses 0x20 or 32 ticks; this command actually turns the note off and is identical to using the 0x80 3C 00 note off command
90 3E 5C 81 20 90 3E 00 20	Turn on note D above middle C with a velocity of 92 (<i>f</i>); pause 160 ticks; turn off note D above middle C and pause 32 ticks
90 40 5C 81 20 90 40 00 20	Turn on note E above middle C with a velocity of 92 (<i>f</i>); pause 160 ticks; turn off note E above middle C and pause 32 ticks
90 41 5C 81 20 90 41 00 20	Turn on note F above middle C with a velocity of 92 (<i>f</i>); pause 160 ticks; turn off note F above middle C and pause 32 ticks
90 43 5C 81 20 90 43 00 20	Turn on note G above middle C with a velocity of 92 (<i>f</i>); pause 160 ticks; turn off note G above middle C and pause 32 ticks
90 45 5C 81 20 90 45 00 20	Turn on note A above middle C with a velocity of 92 (<i>f</i>); pause 160 ticks; turn off note A above middle C and pause 32 ticks
90 47 5C 81 20 90 47 00 20	Turn on note B above middle C with a velocity of 92 (<i>f</i>); pause 160 ticks; turn off note B above middle C and pause 32 ticks
90 48 5C 81 20 90 48 00 20	Turn on note C above middle C with a velocity of 92 (<i>f</i>); pause 160 ticks; turn off note C above middle C and pause 32 ticks
90 47 5C 81 20 90 47 00 20	Turn on note B above middle C with a velocity of 92 (<i>f</i>); pause 160 ticks; turn off note B above middle C and pause 32 ticks
90 45 5C 81 20 90 45 00 20	Turn on note A above middle C with a velocity of 92 (<i>f</i>); pause 160 ticks; turn off note A above middle C and pause 32 ticks
90 43 5C 81 20 90 43 00 20	Turn on note G above middle C with a velocity of 92 (<i>f</i>); pause 160 ticks; turn off note G above middle C and pause 32 ticks
90 41 5C 81 20 90 41 00 20	Turn on note F above middle C with a velocity of 92 (<i>f</i>); pause 160 ticks; turn off note F above middle C and pause 32 ticks
90 40 5C 81 20 90 40 00 20	Turn on note E above middle C with a velocity of 92 (<i>f</i>); pause 160 ticks; turn off note E above middle C and pause 32 ticks
90 3E 5C 81 20 90 3E 00 20	Turn on note D above middle C with a velocity of 92 (<i>f</i>); pause 160 ticks; turn off note D above middle C and pause 32 ticks
90 3C 5C 81 20 90 3C 00 00 FF 2F 00	Turn on note middle C with a velocity of 92 (<i>f</i>); pause 210 ticks; turn off note middle C

Figure 3: Note MIDI Codes Explained

For a more detailed description of the MIDI file format for this example, please refer to Appendix A.

Advantages of MIDI

There are two main advantages of using MIDI, the first being that the MIDI file format is very small. Relatively large and complex songs can be stored in a very small file. For example, a 42-minute song with 31 instrument parts only uses 356 kilobytes of space.

Another advantage of MIDI is that it is a standard that is widely used. Most operating systems support the format without any additional software. In fact, a normal computer with free or relatively inexpensive software can control a number of MIDI enabled instruments, mainly electric keyboards and synthesizers, as well as lighting systems.

Drawbacks & Limitations of MIDI

As shown above in Figure 2, MIDI is not music; rather it is a set of binary instructions, which makes it very difficult, if not impossible for humans to read. This binary file format also makes it vulnerable to corruption, for if one bit of the MIDI file is lost or is incorrect, the entire file may be lost. This limitation does not make MIDI the best format with which to transport sheet music over the Internet.

MIDI also suffers from further drawbacks. Since it is very difficult to directly edit, most users utilize graphical notation editing programs that usually convert MIDI into a proprietary format. When a user saves the MIDI, it will usually sound exactly how they intended, but when they want to edit the MIDI file again, they will find that even though they are using the same program they used to originally edit the MIDI file, it will

not look the same, and if they use a different program, much more information may be lost.

Figure 4 illustrates an example of this problem. Using a relatively sophisticated graphical notation-editing program, in this case Cakewalk, a song was created in standard music notation:

1

Blues

Blues C-major

IVAN
Copyright ©1997 by Igor Khoroshev. All rights reserved.

Have one for the road

The musical score is presented in two systems. The first system contains measures 1 and 2. The second system contains measures 3 and 4. The score is for a blues piece in C major, 4/4 time. It includes parts for Rhodes piano (1), Bass (2), and Cymbal (3). The Rhodes part includes chord diagrams for F (10 fr.), F9 (8 fr.), and C7 (8 fr.).

Figure 4: Sample Song Before Conversion To MIDI

Using the same notation program, the song in Figure 4 was then converted to the MIDI format and reopened. The original song, or *urtext*, now appears to be very different

(see Figure 5 below). For example, the cymbal part is no longer written in percussion notation, and the treble and bass parts for the Rhodes have been combined into one bass clef line. This is because MIDI does not store clef information, so it is up to the notation program to interpret the clef, based on the octave of the notes. Also, the chord boxes (F, F9, C7) in measure 3 are gone and there are some other minor parts of the *urtext* are changed or missing.

1

Blues
Blues C-major

Copyright ©1997 by Igor Khoroshev. All rights reserved.

Comments Missing

Extra measure of rest at the beginning

Rhodes part has been merged into one bass clef part

Guitar chord boxes in measure 3 (below) are missing

Above cymbal part no longer in percussion notation

Figure 5: Sample Song After Conversion To MIDI

The only way to avoid this problem is to not save the file in the MIDI format, leaving it in the notation editor's proprietary format. Unfortunately, if a user wants to share this file with another user, the latter must have the same notation editor or they are

both forced to use the MIDI format, which does not preserve the *urtext*, as there is no other standard way to transport sheet music between two people.

Another problem with the MIDI format is that a song may sound different from computer to computer. As it has been stated before, MIDI is a set of instructions for a computer's sound hardware, not music. How the computer's sound hardware interprets and executes those instructions can vary widely.

Music & The Internet: The Economics of Music

As illustrated by the latest statistics from the National Music Publishers Association (NMPA) in Figure 6, the sales of sheet music topped \$617,000,000 worldwide (NMPA 1). In addition, the American Society of Composers, Authors, and Publishers (ASCAP) calculates that every successful movie song generates around \$20,000 in sales from sheet music alone and a modestly popular song will sell \$25,000 worth of sheet music (Brabec 1). With all of this potential revenue, more and more distributors are being enticed to the Internet in order to increase the sales base.

MASTER SURVEY DATA 1998

\$ Millions	Distribution-Based Income		
	SALE OF PRINTED MUSIC*	RENTAL/PUBLIC LENDING	TOTAL
USA	233.73	N/A	233.73
Germany	131.92	9.07	140.99
Japan ⁽¹⁾	18.68	35.57	54.25
United Kingdom	65.11	N/A	65.11
France	54.83	N/A	54.83
Italy	20.52	N/A	20.52
Spain	N/A	N/A	N/A
Netherlands	21.35	N/A	21.35
Belgium	N/A	N/A	N/A
Switzerland	24.24	0.54	24.78
Canada	13.84	N/A	13.84
	• • •		
Total	\$617.33	\$45.26	\$662.59

Figure 6: NMPA Worldwide Sales Statistics For Sheet Music

Unfortunately, as mid-2000 has shown, having a homepage and selling products through it, does not guarantee that a business will be successful. For each business that has been a success on the Internet, many more have not been. In order to have an effective web presence, the business needs to have a clearly defined strategy, and the technology to carry out the strategy.

Music & The Internet: A Business Example

So how should sheet music be sold on the Internet? There are many factors that must be considered, the first being the format in which the customer can first view the sheet music. There are a two different ways to make sheet music available for preview.

The most common way today is to scan the original printed sheet music and let the customers view parts of the images, or the whole image of the sheet music itself. The advantage of this method is that it is a fairly easy and inexpensive way to allow the

customers to preview the sheet music; however it suffers from some major drawbacks when compared to the alternative method.

First, the sheet music must be manually scanned, which is a time consuming process. The scanned images are almost never perfect and usually need to be touched up or edited in order to make them presentable. This process takes a great deal of time, and if an error is discovered in the printed music, the entire process must be repeated. Also, in using this method, there is no way for a customer to listen to the music. If a business wants to make sound clips available they have to use those supplied by the publisher (which are often not available), or they must hire a band to make a recording of the sheet music, or they need to type in the music into the computer. In either latter case someone has to be paid for performing, or inputting the music, and this cost is handed down to the customer.

Yet another limitation of this method is that there is no automatic way for a customer to transpose the music so that they can see how the music would look in a different key or hear how the sheet music would sound on a particular instrument. Either the publisher or business must do this manually, or the customers must first purchase the sheet music and transpose it themselves. In either case this is another time consuming and costly process.

This method also requires the business to keep an inventory of the music and physically store that sheet music somewhere. They must keep track of the quantity on hand and order more when those quantities run low. They also must maintain some sort of warehouse in which to store the sheet music. Both of these cost large amounts of money, time and effort.

Shipping costs in this method are also more expensive and time consuming. The business must first go to the warehouse and select all the items to fulfill the order. They must then package and ship the order to the customer. All of these costs are passed on to the customer, who must then wait a few days to receive the order.

So is there a better way to sell sheet music on the Internet? Yes, sell it in electronic form. This method, though initially costly, eventually saves time and money. In order to successfully sell sheet music in electronic form, the business must first decide in which electronic format they want to sell the sheet music. As discussed above, the MIDI format is not the ideal format in which to do so. For more details on other formats, see the Current Research section, beginning on page 15.

Once the business has decided on the format in which they wish to sell sheet music, they must convert the music into this format. The procedure is similar to that of scanning the sheet music into images. They must scan the sheet music into a special optical character recognition (OCR) program, which will interpret the image and store it electronically in the desired format. As with scanning images, this procedure is not 100% effective and correct, so the resulting electronic form of the sheet music must be edited.

So far, the two methods have had similar procedures in getting the music ready for customers to preview, but after that their paths diverge. Now that the sheet music is in electronic form, the business need only store one copy of the sheet music on their computer. There is no need for expensive inventory tracking and the electronic form takes up no physical warehouse space. A business can have many more songs for sale as they take up relatively little space, require very little effort to inventory, and they do not have to worry about selling out of a particular song.

There are many additional benefits of having the sheet music in electronic format. For example, a computer can transpose sheet music many times faster and with more accuracy than a human. What would take a person hours or days to do, the computer can do in seconds, and it can do it in any key over and over again. A customer can also view specific parts of the sheet music and have the computer play that specific part. The computer's rendition of the sheet music will never sound as good as a live band, but it is accurate enough to give a consumer a good idea about how it sounds. Also, if a mistake is discovered in the song, it can be easily fixed, and need only be fixed in one location.

When it comes to transporting the sheet music in electronic to a customer, the task is much simpler and less costly than the previous method. A customer merely needs to pay online, and then they can either download the sheet music from the business, or the business can send it to them via email. This method is much quicker than traditional shipping method as it only takes a few seconds for the user to receive their order, rather than a few days.

The electronic method has many advantages over the print method, but it is less secure and a successful electronic copyright protection scheme for anything distributed via the Internet has yet to be introduced. The electronic method makes it much easier for people to copy music and send it to others. However, the print method has no such protection and has worked on the honor system for decades.

Unfortunately, security is not the largest concern when dealing with choosing a format to use to transfer sheet music via the Internet, rather it is choosing the appropriate format, and no single, standard format has been developed to do so effectively.

Current Research

Images

During the early stages of the Internet, the most common way to distribute sheet music was to use an image format, the most popular being the Graphics Interchange Format (GIF). A distributor or musician would scan their original score and save it in this format. Though still used today, it is slowly losing favor.

There are many reasons for this drop in popularity, the first being that a GIF file typically provides poor to moderate print quality, which makes it less desirable for publishers. Another reason why this is a poor choice for distributing music is that it is extremely difficult to edit once the image has been created. For example, if an error is found in the piece of music, the image either has to be changed using a graphics editor, or the original score has to be corrected, and then scanned once again.

Unrelated to the quality and editing problems inherent to images, there are also musical limitations. A musical score saved as an image cannot be searched, nor can it be transposed. If a musician is looking for a particular rhythm or melody, they will not be able to automatically find it within a score saved as an image, rather they will have to search the entire score visually. Also, if they need the score transposed (i.e. written in another key), they will have to convert the image by hand, which is a tedious and time-consuming endeavor.

Portable Document Format

Adobe Systems Incorporated developed the Portable Document Format (PDF) with the purpose of providing a compact way to transport documents via the Internet.

Viewers have been written for various platforms, which makes it one of the most popular transport formats on the Internet today. This format is currently the most widely used format amongst Internet sheet music distributors.

PDFs offer a few advantages over images, in that they are generally smaller than images, and they are easier to duplicate. With the exception of those two advantages, they still suffer from the same disadvantages as images. Non-text elements (e.g. notes, rests) within the document are still difficult to edit, and the editor itself is an expensive piece of software. These documents cannot be transposed and only the text with them can be searched. In order to view a PDF, a user is responsible for downloading and installing a free viewer, whereas web browsers directly support most image formats. One other inherent disadvantage is that the PDF format is proprietary (i.e. closed-source), meaning that no one other than Adobe can modify the format to fit the specific needs of users.

Notation Interchange File Format

During the last decade, both music software companies and music distributors recognized both the lack of a standardized, open-sourced notation file format and the great need to fill this void. In 1994-1995, Passport Designs (publisher of Encore), San Andreas Press (publisher of Score), Coda Music Technologies (publisher of Finale), Musitek (publisher of MidiScan) and TAP Music Systems/Music Ware (publisher of NoteScan) developed the Notation Interchange File Format, also referred to as NIFF (NIFF 1).

The goal of NIFF was to be the musical notation equivalent to the MIDI format, in that it would be universally supported. In order to accomplish this, the developers set upon three tasks: make NIFF extensible, flexible, and compact. The developers understood that notation, like any natural language, is in a constant state of change, so NIFF needed the ability to be modified and updated to suit the needs of musicians. At the time of its development, the Internet was in its infancy and connections were generally slow, so the developers wanted to make the format compact enough to support even the slowest connection speeds (NIFF 1).

NIFF's demise was foreshadowed early in its development when one of the largest financial and technical contributors, Coda Music Technology, withdrew to work on its own cross-platform format called the Enigma Transport Format. Mark of the Unicorn and Twelve Tone Systems (publisher of Cakewalk) joined the development in Coda's place (NIFF 1). The standard was completed in September of 1995 and last updated in 1998, but was never widely accepted and some of NIFF's developers even rejected the format, for example Twelve Tone Systems' notation editor, Cakewalk, does not support the format.

Enigma Transportable Format (ETF)

As stated above, Coda Music Technology withdrew from NIFF development to concentrate on its own format, the Enigma Transportable Format (ETF). ETF is an ASCII format developed as a cross-platform format for use with Coda's Finale notation editor. Enigma is a perfect choice of name for this particular format as it is extremely difficult to work with, and almost impossible to read. No documentation has been

provided to the public, making ETF useless to anyone who does not use the Finale notation editor.

Resource Interchange File Format

In 1991, a joint development effort by IBM and Microsoft produced the Resource Interchange File Format (RIFF). According to their original published document:

The Resource Interchange File Format (RIFF), a tagged file structure, is a general specification upon which many file formats can be defined. The main advantage of RIFF is its extensibility; file formats based on RIFF can be future-proofed, as format changes can be ignored by existing program applications (RIFF 1).

RIFF is suitable for many multimedia tasks, including playing and recording multimedia, as well as exchanging multimedia data between applications and across platforms.

Microsoft took over development of RIFF and all of its multimedia formats (WAV, ASF, etc.) are based on RIFF. Windows Media Player also makes heavy usage of RIFF-based file formats. Though widely used, very little information about RIFF is freely available, as Microsoft owns all rights to the file format. Twelve Tone Systems' Cakewalk Pro Audio Notation Editor supports RIFF as opposed to NIFF, which it helped develop.

GUIDO Music Notation Language

GUIDO, named after the famous music theorist Guido d'Arezzo, is an ASCII-based notation language resembling TeX and designed as a general purpose language to represent sheet music (Hoos 1). A lot of development has gone into GUIDO and there is even a real time web based Java application which allows a person to type in the GUIDO

code to represent a piece of music. The application will then display the graphical representation of the sheet music.

GUIDO has many benefits over other languages in that it is pure text, so it is small, easy to transport, and relatively robust when it comes to corruption. There are also three levels of GUIDO: basic, advanced, and extended. A user can first learn the basic level and then build from there to work with more advanced levels, each of which introduces more and more complex music notation capabilities.

Below is an example of both the GUIDO code (Figure 7) and the graphical representation of the sheet music produced by the GUIDO code (Figure 8) (Note: the code to the left only pertains to the highlighted portions of the sheet music on the right):

```

[[ % 1st voice
\pageFormat<"A4">
\title<"Quartett",dx=0cm,dy=1cm,adj="tc">
\composer<"Ludwig van Beethoven\n
op.130",dx=0,dy=1.4cm>
\staff<1> \clef<"g2"> \key<"G">
\meter<"3/8">
\tempo<"Allegro asai",dx=0,dy=9.6>
\space<7.36> \i<"p",dy=-5.76>
\beam(\slur<y=1.92>(
\crescBegin<dx=1.28,dy=-5.76> d2/8
\space<6.4>
\merge( b1/16 \crescEnd<dy=-5.76>
\dimBegin<dx=1.28,dy=-5.76> b )
\space<6.4> d2/16 \dimEnd<dx=0,dy=-
5.76>)) _/16 ... ],
[ % 2nd voice
\staff<2> \clef<"g2"> \key<"G">
\meter<"3/8">
\space<7.36> \i<"p",dy=-8,dx=0>
\beam(\slur<dy=-1.92>(
\crescBegin<dx=1.92,dy=-8.96> b0/8
\space<6.4>
\merge( d1/16 \crescEnd<dx=0,dy=-8.96>
\dimBegin<dx=1.28,dy=-8.96> d )
\space<6.4> b0/8 \dimEnd<dx=0,dy=-8.96>))
... ],
[ % 3rd voice
\staff<3> \clef<"c3"> \key<"G">
\meter<"3/8">
\space<7.36> \i<"p",dy=-5.76,dx=0>
\beam(\slur<dy=1.92>(
\crescBegin<dx=1.28,dy=-5.76> g0/8
\space<6.4>
\merge( b1/16 \crescEnd<dx=0,dy=-5.76>
\dimBegin<dx=1.28,dy=-5.76> b )
\space<6.4> g0/8 \dimEnd<dx=0,dy=-5.76>))
... ],
[ % 4th voice
\staff<4> \clef<"f4"> \key<"G">
\meter<"3/8">
\space<7.36> \i<"p",dy=-5.76,dx=0>
\crescBegin<dx=1.28,dy=-5.76>
\merge( g0*3/16 \crescEnd<dx=0,dy=-5.76>
\dimBegin<dx=8.32,dy=-5.76> g0/16 )
\space<12.8> _/8 \dimEnd<dx=0,dy=-5.76>
... ] }

```

Figure 7: GUIDO Code for First Measure of “Quartett”

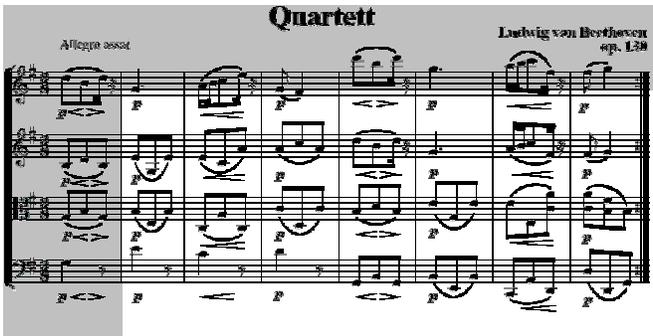


Figure 8: Sheet Music for “Quartett”

There are a few drawbacks to GUIDO. Since it was designed to be easily parsed by a computer, GUIDO is very difficult for a person to read. Also, only two programs support GUIDO, one of which was developed by the authors of GUIDO to test its capabilities.

abc

abc is one of many macro packages designed to give the TeX language the ability to display music notation. TeX is a computer language that was developed for typesetting mathematical and other technical material (TeX 1). Before the explosion of the Internet, and in fact before the rise of the personal computer, TeX was one of the only ways available for mathematicians to use to print their formulas.

TeX has been adopted in many other fields like the natural sciences, computer science, and even linguistics, as well as music. Unfortunately, with the development of HTML and more sophisticated markup languages, TeX has become less and less popular, if not altogether obsolete.

Common Music Notation

Common Music Notation is a LISP-based notation language (Castan 7). It requires knowledge of LISP, without which it is very difficult to use. For example, a chord in CMN would look like the figure below:

(chord (notes b3, g4) q)

Figure 9: A Chord in Common Music Notation

The **chord ()** is a function that has two parameters, one for notes (**notes()**) and one for the duration of the chord (**q**). Since CMN was written in LISP, it was never a very popular notation language and is rarely used, if at all, today.

eXtensible Markup Language (XML)

A relatively new development, eXtensible Markup Language (XML) promises to be a powerful tool in many aspects of computing. XML is a meta-markup language, in that a language is defined to solve a problem. XML defines structures, not formatting, thus it only describes a documents structure and meaning, not how to format it (Harold 5).

At a basic level, XML is an incredibly simple data format. It can be written in 100 percent pure ASCII text. As stated before, ASCII text is reasonably resistant to corruption. The removal of bits or even large sequences of bytes does not noticeably corrupt the remaining text. This starkly contrasts with many other formats, such as compressed data where the loss of even a single byte can render the entire remainder of the file unreadable (Harold 6).

Another useful characteristic is that XML stores metadata (data about the data) with the data itself. Therefore in 20 years the file will still be understandable, as opposed to the hundreds of proprietary file formats that were developed over the last few decades. If the original program that wrote the file in its proprietary format is not available, the file itself is useless. This is why most of the data written by computers over the last 40 years has been lost (Harold 6). Using XML, anyone can create a customized, browser and platform independent application to solve a particular problem

To solve a problem using XML, a designer must first thoroughly define the problem and how best to solve it programmatically. This is a very necessary step as XML is much more rigid than HTML and more like a programming language. Next, the

designer must develop a Document Type Definition (DTD) in which they define guidelines for attributes and elements that were defined and developed in the first step. Finally, with the DTD in hand, a designer can write an XML application or document, which follows the guidelines set within the DTD.

In order to view the XML application or document, the user's browser or program will first request the application or document itself. In the header of that document is a document type declaration, which specifies either the DTD itself, or a location where the DTD can be found. The figure below illustrates the latter case:

```
<!DOCTYPE EMNML PUBLIC "-//EMNML//DTD/EN" "http://www.usd.edu/eric/thesis/emnml.dtd">
```

Figure 10: Sample Document Type Declaration For External DTD

The document declaration above specifies that the DTD for this particular XML application/document can be found at the URL:

<http://www.usd.edu/eric/thesis/emnml.dtd>. When the user's browser/program requests this particular document/application it will read the header and then request the DTD specified before it continues reading the remainder of the document/application:

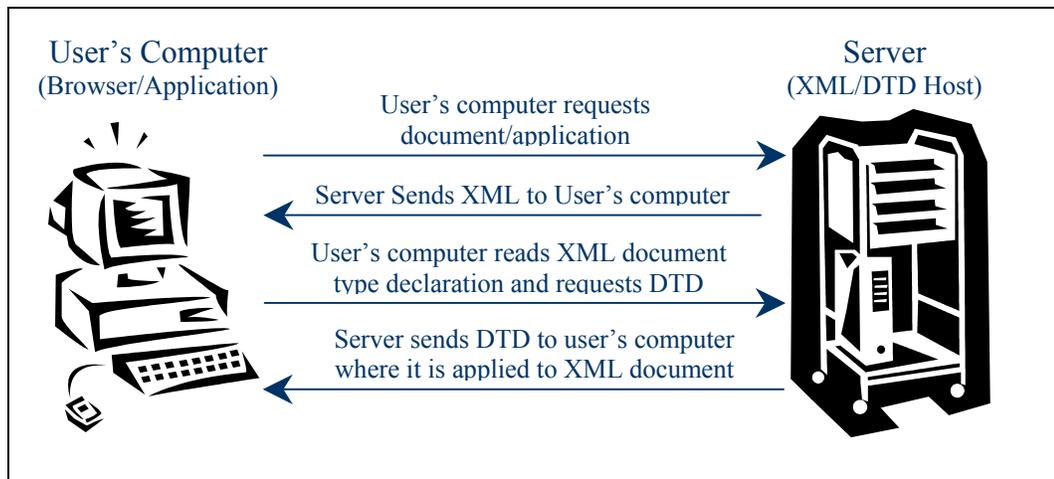


Figure 11: Requesting An External Document Type Definition

Frequently the XML document/application is packaged with a separate DTD, therefore it is not necessary to download the DTD. The document type declaration for this circumstance would look like the figure below:

```

    <!DOCTYPE EMNML SYSTEM "emnml.dtd">
  
```

Figure 12: Sample Document Type Declaration For Internal DTD

When applied for use in representing sheet music, XML has many advantages over the methods and formats discussed above. First, XML is written entirely in ASCII characters, which make it compact, reasonably robust against corruption, readable by humans, and platform independent. It is also very flexible and can be adapted to solve many problems and work in many different ways. One major benefit of using XML is that it is supported by the World Wide Web Consortium (W3C) and is open source.

Writing an open-sourced language for sheet music using an open sourced technology like XML has many advantages. First, and foremost, when users save and transport sheet music in an such a language, they do not have to worry about paying

royalties to the company, nor to they have to worry about trademark and copyright infringement issues that are inherent in using proprietary formats. Since the language is open-source, they can tailor the language to suit their needs. If they need a particular tool, they can add it themselves, or ask the developer to add the feature.

There are limitations to XML though, especially when it comes to developing a standard way to transport sheet music via the Internet. First, XML is relatively new and is not yet widely supported, especially when it comes to web browsers. Also, XML can be used to describe sheet music, but not display it. Either the sheet music language must be included in the XML specification, or a special viewer must be written. All of these limitations are relatively minor when compared to the many benefits of using XML to transport sheet music via the Internet.

Quite a bit of research has already been done with using XML to represent sheet music. Most has stopped well short of being a comprehensive solution to the problem of creating a standard way to transport sheet music over the Internet. Instead authors have merely experimented to show that it in fact could be done. Listed below are some examples of research done using XML to represent sheet music:

[MusicML](#)

MusicML was one of the first attempts to prove that sheet music could be represented by XML (MusicML 1). It incorporates only basic music notation elements and is fairly difficult to read. It does, however, have a nice Java viewer to help visualize what the XML code represents.

MusiXML

MusiXML is one of the simplest attempts at using XML to represent sheet music. It supports only the most basic music notation elements like notes and rests, and is difficult to read (Castan 9).

Music Notation Markup Language

The objective of Music Notation Markup Language (MNML) was the same as the goal of this thesis: to develop a standard way to transport sheet music over the Internet (Wei 1). As with the previous two XML languages, MNML's developers only implemented basic music notation elements to prove that it was possible to accomplish their goal. There has been no development past version 2.0 of MNML and its development site is no longer accessible.

XScore

XScore is one of the most comprehensive sheet music implementation of XML available. It has the same basic support for music notation as the implementations above; however, it supports more intermediate features like lyrics and chord boxes and it is fairly easy to read (Grigatis 1).

Unfortunately, there are quite a few notational limitations to this implementation, and the most current version has not been updated in well over two years.

MML

Of all of the XML implementations used to represent sheet music, MML is the most comprehensive. It supports most standard music notation elements as well as some features unrelated to music notation, like play lists (Steyn 1). However, it is rather difficult to read and very complex to learn. It is designed to be used primarily with a notation editor, and was not designed to be edited by hand. It also missing some common notation elements like chord boxes, and some note ornamentations.

Conclusion

As the above research has shown, there have been many attempts over the past two decades to develop a standard way to transmit sheet music via the Internet. Twenty years later, there still is no way to do so effectively and the need for such a standard still exists today.

An XML Solution

The first step in designing an XML solution for transporting sheet music via the Internet is to develop the Document Type Definition (i.e. language specification). Music Notation Markup Language (MNML) as well MusicML, xScore, and Music Markup Language (MML) serve as guides, in the development of a new music notation language called Extensible Music Notation Markup Language (EMNML). The purpose of EMNML is to offer all that is found in previously mentioned markup languages, yet be easy enough to use for musicians without a special editor or viewer.

In EMNML, all of the tags and tag attributes are defined using common music naming conventions and nomenclature. For example, if a composer or arranger would like to show notes as being slurred, they simply need to encapsulate the notes inside a **<SLUR>** tag.

The design goal of EMNML is to keep the musician in mind during the entire development process. In order to assure that this goal is achieved, musicians did usability testing of the EMNML DTD following the development of the initial version of the DTD.

Developing the EMNML DTD

As previously stated in the Current Research section about XML, the first step in designing an application in XML is to write the DTD or specification. This is perhaps the most important step in the entire development of this project as everything depends upon the DTD being both accurate and robust enough to handle all the needs of the application.

The development of an ASCII-based language to represent sheet music proved to be very difficult. Music notation has much in common with a modern language: it is very complex, there are many subtle nuances inherent in the language and there are many different ways to accomplish the same task. In fact, some elements of music notation can be found in many different languages, including English, German, French, and more predominantly in Italian (Blood 1). At times, these issues can be very difficult to deal with in a uniform way (see Figure 13).

Terminology	Definition
<i>Agitato</i> or <i>agiatatamente</i> (Italian) <i>Agité</i> (French) <i>Agitirt</i> or <i>Agitiert</i> (German)	agitated, agitatedly
<i>Fantastico</i> (Italian) <i>Fantasque</i> (French) <i>Fantastisch</i> (German)	fantastic, whimsical, capricious
<i>Maestoso</i> (Italian) <i>Majestueux</i> or <i>Majestueuse</i> (French) <i>Majestätisch</i> (German)	majestic(ly)
<i>Tenore</i> (Italian) <i>Ténor</i> (French) <i>Tenor</i> (German)	tenor

Figure 13: Examples of Music Terminology in Different Languages

Other than musical notation factors, the syntax factors of XML itself had to dealt with as well. When nesting tags, the inner tags have to be closed before the outer ones to

be syntactically correct. This caused many problems with tags like **<SLUR>**, **<TIE>**, etc. that could span across **<MEASURE>** tags, and so the concept of a **CONTINUE** attribute was developed. Using this attribute, the tag could be closed, thus conforming to the syntax rules of XML, yet when a parser sees the **CONTINUE** attribute, it knows that the tag continues into the next measure (see Appendix B for more details).

There were two main goals during the development of the EMNML DTD. It had to support most common types of music notation and certain variations of music notation, and more importantly, it had to be easy to read. To accomplish the first goal, the author relied on his many years of experience in music, music theory references (see Bibliography) and advice from music professors. These professors also reviewed the DTD many times before it was released for usability testing (see next section for further details).

The other major development goal of easy readability was the most difficult issue. All the tags and subsequent attributes had to be named in such a way that a person could immediately associate the EMNML tag with its music notation counterpart. For example, a tag called something similar to slur should enclose notes within a slur. To illustrate this point, take the following short piece of music:



Figure 14: Slur Example

In EMNML, Figure 14 would look like Figure 15:

```
1 <!DOCTYPE EMNML PUBLIC "-//EMNML//DTD/EN"
  "http://www.usd.edu/eric/thesis/emnml.dtd">
2 <EMNML>
3 <STAFF NAME="example"/>
4 <CLEF STAFF="example" TYPE="treble"/>
5 <KEY PITCH="C" SCALE="Major"/>
6 <TIME STYLE="common"/>
7 <MEASURE STAFF="example" NAME="1">
8   <SLUR>
9     <NOTE PITCH="D" OCTAVE="1" LENGTH="1/4"/>
10    <NOTE PITCH="C" OCTAVE="1" LENGTH="1/4"/>
11  </SLUR>
12
13  <SLUR>
14    <NOTE PITCH="B" OCTAVE="0" LENGTH="1/4"/>
15    <NOTE PITCH="A" OCTAVE="0" LENGTH="1/4"/>
16  </SLUR>
17 </MEASURE>
18 </EMNML>
```

Figure 15: EMNML Representation of Slur Example

In the EMNML version, the notes represented by the **<NOTE>** tag are enclosed within a **<SLUR>** tag, which in turn represents a slur. Even though the sheet music in Figure 14 outwardly seems to be simple, there is a lot of implied information that must be explicitly stated in the EMNML representation. Figure 16 shows a more detailed description of what the EMNML represents:

EMNML Representation	Sheet Music
<p><EMNML> This tag denotes that the following tags are in the EMNML format. It is analogous to the <HTML> tag in HTML documents.</p>	(none)
<p><STAFF NAME="example"/> This tag designates a staff. It allows for multiple staves in different clefs, each designated by a unique name. This is necessary to associate the measures and their content with their respective staff in the sheet music.</p>	(none)
<p><CLEF STAFF="example" TYPE="treble"/> For each staff, there needs to be a clef. In this example, the song is written in treble clef, as designated in the TYPE attribute.</p>	
<p><KEY PITCH="C" SCALE="Major"/> All songs have a key signature that designates the key in which the song is written. In this example, the song is written in the key C Major in which there are no sharps or flats. The two attributes designate the key itself and whether the scale is Major (as in this example) or minor.</p>	
<p><TIME STYLE="common"/> The TIME tag designates the time in which the music is to be played. Typical times are 4/4, 3/4, and 6/8. In this example, the music is written in common time, which is the same as 4/4. The TIME tag could have been written as follows and accomplished the same thing musically, though not notationally:</p> <p><TIME BEATS="4" NOTE="4"/></p> <p>In either case, this designates that a quarter note receives one beat or count, and that there are four quarter notes per measure.</p>	
<p><MEASURE STAFF="example" NAME="1"> To designate the division of sheet music into measures, the MEASURE tag must be used. Its attributes designate the staff to which the measure belongs as well the name of the measure. Typically, the name is the measure number on the sheet music itself, but it can be anything the user wishes it to be.</p>	
<p><SLUR> <NOTE PITCH="D" OCTAVE="1" LENGTH="1/4"> <NOTE PITCH="C" OCTAVE="1" LENGTH="1/4"> </SLUR></p> <p>The above statements represent the first two slurred notes. The notes themselves are encapsulated by the SLUR tag, which designates that they are to be slurred. The NOTE tags themselves contain information specific to the note, like its pitch, its location on the staff or octave and the duration or length of the note.</p> <p>In EMNML, middle C () is represented by an OCTAVE of 0. In this case, the C is an octave, or 12 semitones (1/2 steps) above the middle C, therefore it has an OCTAVE of 1, as would any note above it until the next higher C.</p>	
<p><SLUR> <NOTE PITCH="B" OCTAVE="0" LENGTH="1/4"> <NOTE PITCH="A" OCTAVE="0" LENGTH="1/4"> </SLUR></p> <p>The statements above are structured exactly as those in the previous description. All that has changed are the note pitches and octaves. Since both of these notes are above middle C, but below the C above middle C, they have an OCTAVE of 0.</p>	
<p></MEASURE> </EMNML></p> <p>These two tags merely close the encapsulation of the rest of the data. They designate the end of the measure and the end of the EMNML document, respectively.</p>	(none)

Figure 16: EMNML Detailed Description

In keeping with the goal of easy readability, many of the common music notation concepts were directly applied to the DTD. A good example of this is the **<MEASURE>** tag. Since all sheet music is divided into measures, it was felt that EMNML should also divide the representation of the sheet music into measures as well. As shown in Figure 15, measure tags encapsulate all four notes that comprise the single measure in Figure 14.

For more detailed information about EMNML, please refer to the EMNML documentation in Appendix B, and the EMNML DTD Specification in Appendix C.

Usability Testing

To assure that the design goals of EMNML were met, usability testing was performed on the initial specification. The usability tests were conducted prior to the formal documentation being available, so the users had to use the basic documentation included within the specification itself (see Appendix C).

Selecting The Users

In order to fully test the capabilities of EMNML, a wide range of users were asked to participate in the usability testing. Of key importance was that all of the usability testers had some musical background. A total of seven users, who matched this criterion, were selected.

Two of the usability testers had doctoral degrees in their respective fields. User 1 had a Ph.D. in Music and had some experience in HTML, which is important as EMNML has a similar structure to that of HTML. User 2 had a Ph.D. in Political Science and had moderate experience with music notation but very little experience with HTML.

The next three usability testers were all graduate students. Users 3 and 4 were both computer science graduate students, and User 5 was a Public Administration graduate student. User 3 had moderate music notation experience and was an expert in HTML. User 4 was less experienced in music notation, but also was an expert in HTML. User 5 had moderate experience in both music notation and HTML.

The final two usability testers were undergraduate students. User 6 was a computer science major with moderate knowledge in music notation and was an expert in

HTML. User 7 majored in music, had moderate to advanced music notation experience, but had very little knowledge of HTML.

Designing The Tests

Two tasks were developed for the usability testing. The first was a very simple example using a C Major scale as shown in the figure below:



Figure 17: Usability Testing Task 1 - C Major Scale

The scale was manually converted into EMNML (see Appendix C, Usability Task 1) and given to the users. Using only the EMNML version of the scale in Figure 17, they were asked to recreate the scale as closely as possible using standard music notation. The purpose of the first task was to act as an introduction to EMNML, namely, to give the users a good feel for the language before tackling more difficult tasks. The task only deals with one scale and only has simple quarter and half notes.

The second task was designed to test more complex features of EMNML. As was the case in the first task, a piece of sheet music (see Figure 18) was manually converted into EMNML (see Appendix D, Usability Task 2) and given to the users to recreate.

SONATA IN A

Wolfgang Amadeus Mozart



The image displays a musical score for the Sonata in A by Wolfgang Amadeus Mozart. It consists of two systems of music, each with a treble and bass staff. The first system is marked 'Andante (♩=105)' and 'mp'. The second system is marked 'rit.'. The score includes various musical notations such as slurs, ties, and chords.

Figure 18: Usability Testing Task 2 – *Sonata in A* by W.A. Mozart

This task contains more complex implementations of EMNML. For example, there are two staves and clefs to deal with as well as complex ties, slurs and chords.

After the users completed both tasks, they were asked to complete a short, five-question survey. The survey asked the users to rate the readability, navigability, understandability, user's personal success, and overall usability based on a 10-point scale.

Results

Each task was graded and checked for three types of errors. The first, and most prevalent error was the **Slur Error**. If a user failed to slur two or more notes, or slurred too many notes, the measure was graded as a **Slur Error**. The next error was a **Note Error**, which is defined as a note of the incorrect pitch, octave, or length. The final type of error, **Other Error**, covered the rest of errors that occurred in the testing. For example, if a user forgot to write down the dynamic or tempo, the measure was graded as an **Other**

Error. Also, if there was more than one of the same type of error in a measure, it was graded only as one error, so a measure could only have a maximum of three errors, one of each type.

As the results in Figure 19 show, all users were very successful in completing the first task and few mistakes were made. Only 5 errors were made, and most were very minor errors related to the dynamic and tempo.

	User 1	User 2	User 3	User 4	User 5	User 6	User 7	Averages
Slur Errors	0	0	0	0	0	0	0	0
Note Errors	0	0	0	0	0	0	0	0
Other Errors	0	2	2	0	0	0	1	0.714286

Figure 19: User Errors for Task 1

In the more complex Task 2, the most common difficulty was with the <SLUR> and <TIE> tags. Most users, who made slur mistakes, had difficulty determining where one slur ended and where the other began. Figure 20 shows the number and types of errors the users made during Task 2.

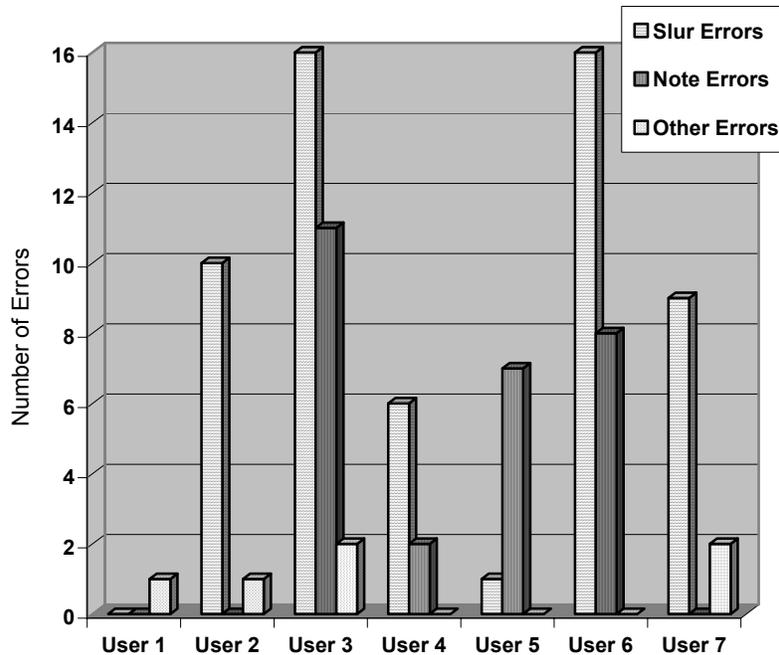


Figure 20: User Errors For Task 2

Another common error in Task 2 was determining the octave of a note. Some users were initially confused by what notes belong in octave 0. In EMNML, octave 0 represents middle C (c1) and every note above middle C, until the next C (c2) is also in octave 0 (see Figure 21).

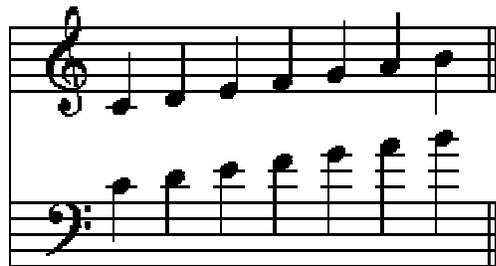


Figure 21: Notes in Octave 0 of EMNML

During post usability testing interviews, when asked to read the description given to them regarding the octave, almost all of the users realized their mistakes and attributed them to not carefully reading the instructions.

Conclusions

Statistics compiled from the usability surveys and from post usability test interviews show that in general, most users were very satisfied about their experience with EMNML (see Figure 22).

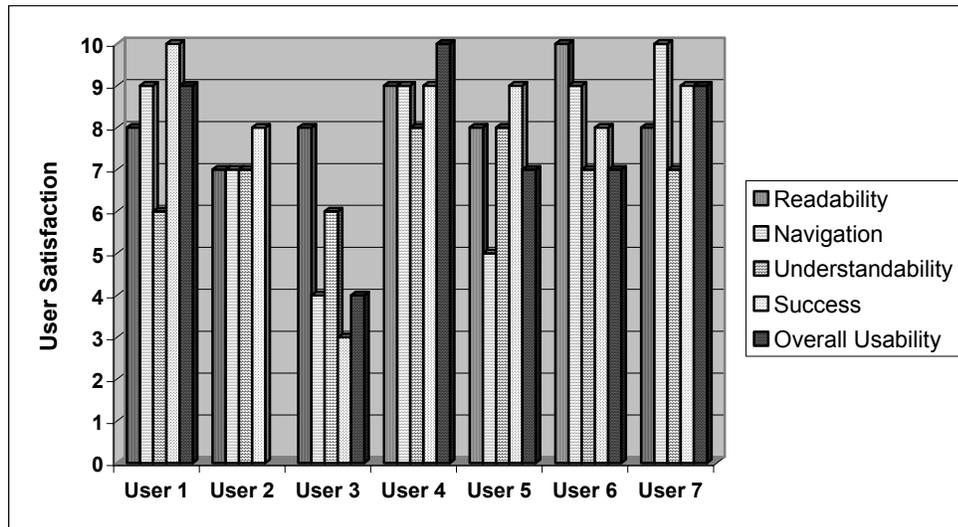


Figure 22: Usability Survey Results

On a scale from 1 to 10, where 1 shows great dissatisfaction and 10 shows great satisfaction, the average rankings of the five main categories were: Readability (8.5), Navigation (7.7), Understandability (7), Success (8), Overall Usability (7.7). Had the formal documentation been ready for use in the testing, these rankings might have been higher.

Based on the results of the testing and the surveys, it was determined that the **<SLUR>** and **<TIE>** tags needed to be changed to make them both easier to use and read. During the usability testing, the **<SLUR>** and **<TIE>** tags had an attribute called **CONTINUE**. The user could then specify whether the slur or tie continued to the next measure using the “next” value, or if the slur or tie continued from the previous measure using the “previous” value, as shown in the figure below (note: both examples produce the same output):

<pre> <SLUR> <NOTE PITCH="C" OCTAVE="1" LENGTH="1/4"/> </SLUR> </MEASURE> <MEASURE STAFF="Violin I" NAME="2"> <SLUR CONTINUE="previous"> <NOTE PITCH="B" OCTAVE="0" LENGTH="1/4"/> </SLUR> </pre>	
<pre> <SLUR CONTINUE="next"> <NOTE PITCH="C" OCTAVE="1" LENGTH="1/4"/> </SLUR> </MEASURE> <MEASURE STAFF="Violin I" NAME="2"> <SLUR> <NOTE PITCH="B" OCTAVE="0" LENGTH="1/4"/> </SLUR> </pre>	

Figure 23: Examples of Old Slur Tag

All of the slurs in the tasks were written using the first example from Figure 23. It required the users to first start the slur, and then read ahead to see if the previous slur was a part of the first. This caused much of the confusion with the slurs, so it was decided that the **CONTINUE** attribute be simplified to just the word itself, **CONTINUE**, as illustrated in the figure below:

<pre> <SLUR CONTINUE> <NOTE PITCH="C" OCTAVE="1" LENGTH="1/4"/> </SLUR> </MEASURE> <MEASURE STAFF="Violin I" NAME="2"> <SLUR> <NOTE PITCH="B" OCTAVE="0" LENGTH="1/4"/> </SLUR> </pre>	
--	---

Figure 24: Example of New Slur Tag

This example works much like the example of the “next” value in Figure 23. However, the user need no longer specify where the slur or tie will continue, as it will always continue until the next **<SLUR>** or **<TIE>** tag. This logic was also passed on to other tags that use the CONTINUE attribute, such as **<DYNAMIC>** and **<TEMPO>**.

In general, the usability tests were very successful at pointing out some weaknesses of EMNML. For the most part, the users we very successful in completing the tasks, and based on their comments, EMNML achieves its goal of being both easy to read and easy to use.

Converting MIDI to EMNML

Converting MIDI to EMNML is a complex matter. As has been previously discussed in the Introduction section (starting on page 3), MIDI is only a set of instructions for a computer's sound card. To be able to convert it into EMNML, the instructions must first be interpreted and the note values calculated. Only after those two steps are completed can the actual conversion take place.

Many programs have been written to convert MIDI instructions into ASCII text. Since the motto of computer science is "Why reinvent the wheel?" one of these programs was used as the initial step to convert MIDI to EMNML. The program is fittingly called *midi2txt* and was written by Günter Nagler (Wallace 1). The reason it was chosen was that it is the most capable MIDI to text converter available (see Appendix E for source code).

Since the complete source code was not available for *midi2txt*, a wrapper program called *midi2emnml* was written in PERL to parse the output from the *midi2txt* program. Using *midi2txt* to convert a C Major scale MIDI file into text would yield the output as found in Figure 25.

1	// c_scale2.mid	(continued from previous)
2	mthd	
3	version 0 // single multichanneltrack	24 +a4 \$7F;
4	// 1 track	25 1/4;-a4 \$00;
5	unit 120 // is 1/4	26 +b4 \$7F;
6	end mthd	27 1/4;-b4 \$00;
7		28 +c5 \$7F;
8	mtrk(\$1) // track 1	29 1/4;-c5 \$00;
9	trackname "untitled"	30 +b4 \$7F;
10	text "Eric Mosterd"	31 1/4;-b4 \$00;
11	tact 4 / 4 24 8	32 +a4 \$7F;
12	key "Cmaj"	33 1/4;-a4 \$00;
13	beats 120.00000 /* 500000 microsec/beat */	34 +g4 \$7F;
14	+c4 \$7F;	35 1/4;-g4 \$00;
15	1/4;-c4 \$00;	36 +f4 \$7F;
16	+d4 \$7F;	37 1/4;-f4 \$00;
17	1/4;-d4 \$00;	38 +e4 \$7F;
18	+e4 \$7F;	39 1/4;-e4 \$00;
19	1/4;-e4 \$00;	40 +d4 \$7F;
20	+f4 \$7F;	41 1/4;-d4 \$00;
21	1/4;-f4 \$00;	42 +c4 \$7F;
22	+g4 \$7F;	43 2/4;-c4 \$00;
23	1/4;-g4 \$00;	44 end mtrk

Figure 25: Output From *midi2txt* For C Major Scale

The first part of the output in Figure 25 is the header information from the MIDI file. It contains text and copyright information as well as information about how the song is to be played, like the tempo, time and key signature, and volume. Following the header information are the actual notes themselves. They are designated by a length, whether or not the note is to be turned on or off (represented by a + or -), the note itself, and the velocity of the note. For example, `1/4;-e4 $00;` states that there is a pause of a quarter note in length, or 192 ticks, then the note E above middle C is to be turned off with a velocity of 0.

The wrapper takes this output, parses through each line, and outputs the appropriate EMNML tags. The major task that the wrapper has to complete is the reconstruction of measures. To do this, it keeps track of note values and when the values total the number of notes per measure as designated in the header information (in the case of Figure 25, 4/4) it starts a new measure. Tempo and dynamic changes are also tracked in a similar manner.

The wrapper also translates note lengths, which becomes more complex when dotted notes are taken into consideration. When a note is dotted, its value is increased by half. For each subsequent dot, half of the previous dot's value is added to the note. Another complexity of MIDI that the program handles is the assignment of notes to the correct track(s).

If the text from Figure 25 were used as input for the wrapper, the output shown in Figure 26 would be produced.

```

<!DOCTYPE EMNML PUBLIC "-//EMNML//DTD/EN" "http://www.usd.edu/eric/thesis/emnml.dtd">
<EMNML>
<STAFF NAME="staff 1"/>
<CLEF STAFF="staff 1" TYPE="treble"/>
<COMMENT>Eric Mosterd\x0a</COMMENT>
<KEY PITCH="C" SCALE="Major"/>
<TEMPO NOTE="1/4" BEAT="120"/>
<TIME BEATS="4" NOTE="1/4"/>
<MEASURE STAFF="staff 1" NAME="1">
  <DYNAMIC VOLUME="fff">
    <NOTE PITCH="C" OCTAVE="0" LENGTH="1/4"/>
    <NOTE PITCH="D" OCTAVE="0" LENGTH="1/4"/>
    <NOTE PITCH="E" OCTAVE="0" LENGTH="1/4"/>
    <NOTE PITCH="F" OCTAVE="0" LENGTH="1/4"/>
  </MEASURE>
<MEASURE STAFF="staff 1" NAME="2">
  <NOTE PITCH="G" OCTAVE="0" LENGTH="1/4"/>
  <NOTE PITCH="A" OCTAVE="0" LENGTH="1/4"/>
  <NOTE PITCH="B" OCTAVE="0" LENGTH="1/4"/>
  <NOTE PITCH="C" OCTAVE="1" LENGTH="1/4"/>
</MEASURE>
<MEASURE STAFF="staff 1" NAME="3">
  <NOTE PITCH="B" OCTAVE="0" LENGTH="1/4"/>
  <NOTE PITCH="A" OCTAVE="0" LENGTH="1/4"/>
  <NOTE PITCH="G" OCTAVE="0" LENGTH="1/4"/>
  <NOTE PITCH="F" OCTAVE="0" LENGTH="1/4"/>
</MEASURE>
<MEASURE STAFF="staff 1" NAME="4">
  <NOTE PITCH="E" OCTAVE="0" LENGTH="1/4"/>
  <NOTE PITCH="D" OCTAVE="0" LENGTH="1/4"/>
  <NOTE PITCH="C" OCTAVE="0" LENGTH="1/2"/>
</MEASURE>
</EMNML>

```

Figure 26: MIDI2EMNML Output For C Major Scale

Future Research

Even though EMNML is fairly comprehensive, there are still few areas that could be expanded upon. The first and most important development needs to be either direct browser support for EMNML, or a browser plug-in. This will facilitate the use of EMNML as a replacement for MIDI, since a third-party viewer or notation editor would not be required for viewing EMNML. The viewer should have playback, search, and transportation capabilities to allow users to fully preview a score in EMNML.

When it comes to the notation that EMNML supports, there is also room for enhancement and improvement. Support for guitar tablature and notation needs to be added, as well as support for lesser-used notation elements.

EMNML also needs to be expanded to support internationalization (abbreviated as I18N). Some types of dynamics, tempos, and other types of notation have different names in different languages and cultures, and this should be supported by EMNML. Text elements like the **<TITLE>**, **<COMMENT>**, etc. tags should have an attribute that specifies the language of the text between the tag.

Security is another feature, which is not yet introduced in EMNML. In order to be a viable option for businesses to use, there needs to be some kind of copy protection. Unfortunately such protection is lacking, even for the most common formats on the Internet, like ZIP files, MP3s, and the many graphics formats available.

Unrelated to the features of EMNML itself, the MIDI2EMNML program needs to be improved. It needs the ability to determine a particular staff's clef based on the octave of the notes in a particular track. Furthermore, the wrapper should be combined with the converter program itself into one program.

Conclusions

There were two main design goals for the development of EMNML. First, it had to be a comprehensive way to transport sheet music via the Internet without sacrificing the content or quality of the sheet music itself so that EMNML could be accepted as the way to transport sheet music via the Internet. The second, and most important goal was that the EMNML language had to be easy to read.

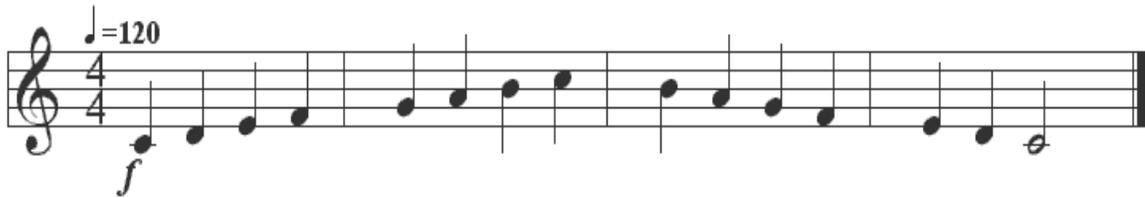
The Current Research section showed that there have been many attempts at creating a standard way to transport sheet music via the Internet, but most have not succeeded. And when it comes to representing sheet music using XML, many attempts have been made, yet none have been successful. These attempts failed to combine the two goals of EMNML together into one design. Some were comprehensive enough to represent sheet music accurately, but they were difficult to read and designed to be parsed by computers. Thus, users are forced to rely upon third-party technology to read them. Other attempts were easy to read, but not comprehensive enough to make them a viable option. So there was still a need for a comprehensive, easy to read language.

EMNML was designed to be that language. Through hours of research and testing, EMNML has proven that it is comprehensive enough to represent most types of sheet music notation. Usability testing has proven that most users find EMNML easy to work with, very readable, and easily learned with little or no documentation.

Though EMNML still requires enhancements to its design, the foundations of a comprehensive and easily readable language for transporting sheet music via the Internet have been laid. All that remains is the acceptance of the language and its adoption as a standard.

Appendix A: MIDI File Format

C Major Scale



MIDI Code For C Major Scale (in hex)

4D 54 68 64 00 00 00 06 00 00 00 01 00 C0 4D 54 72 6B 00 00 01 25 00 FF 03 0D 43 20 4D
61 6A 6F 72 20 53 63 61 6C 65 00 FF 01 0F 42 79 20 45 72 69 63 20 4D 6F 73 74 65 72 64
00 FF 02 20 43 6F 70 79 72 69 67 68 74 20 A9 20 32 30 30 31 20 62 79 20 45 72 69 63 20
4D 6F 73 74 65 72 64 00 FF 02 13 41 6C 6C 20 52 69 67 68 74 73 20 52 65 73 65 72 76 65
64 00 FF 01 20 47 65 6E 65 72 61 74 65 64 20 62 79 20 4E 6F 74 65 57 6F 72 74 68 79 20
43 6F 6D 70 6F 73 65 72 00 B0 07 7F 00 B0 0A 40 00 FF 51 03 07 A1 20 00 FF 58 04 04 02
18 08 00 90 3C 5C 81 20 90 3C 00 20 90 3E 5C 81 20 90 3E 00 20 90 40 5C 81 20 90 40 00
20 90 41 5C 81 20 90 41 00 20 90 43 5C 81 20 90 43 00 20 90 45 5C 81 20 90 45 00 20 90
47 5C 81 20 90 47 00 20 90 48 5C 81 20 90 48 00 20 90 47 5C 81 20 90 47 00 20 90 45 5C
81 20 90 45 00 20 90 43 5C 81 20 90 43 00 20 90 41 5C 81 20 90 41 00 20 90 40 5C 81 20
90 40 00 20 90 3E 5C 81 20 90 3E 00 20 90 3C 5C 82 50 90 3C 00 00 FF 2F 00

MIDI Code Explained

MIDI Hex Code	Description
4D 54 68 64	Designates that this is a midi file (ASCII = "MThd" = MIDI header)
00 00 00 06	32 bit representation of the decimal number 6 which is the length of the MIDI header in bytes
00 00 00 01 00 C0	<p>These 6 bytes were determined by the length from above.</p> <p>The first 0x0000 (or word) are the format of the MIDI file, which in this is a type 0 MIDI file; in other words it contains a single multi-channel track.</p> <p>The next word, 0x0001, designates the number of tracks in the file. In this case, there is only one track.</p> <p>Finally, the last word, 0x00C0, is the division of a quarter note. In this case 192 ticks equals one quarter note</p>
4D 54 72 6B	Designates a Track Data Format (ASCII = "MTrk"); this is where the actual song is stored
00 00 01 25	Designates the length of the MTrk, which in this case is 293 bytes (note: this accounts for all remaining bytes in the file)
00 FF 03 0D	Designates the name of the song and its length; in this case the next 13 bytes is the name itself

43 20 4D 61 6A 6F 72 20 53 63 61 6C 65	The name of the song in hex; the ASCII representation is: "C Major Scale"
00 FF 01 0F	Designates a text event of 0x0F or 15 bytes in length
42 79 20 45 72 69 63 20 4D 6F 73 74 65 72 64	The hex representation of the text itself; the ASCII representation is: "By Eric Mosterd"
00 FF 02 20	Designates the copyright notice and its length of 32 bytes
43 6F 70 79 72 69 67 68 74 20 A9 20 32 30 30 31 20 62 79 20 45 72 69 63 20 4D 6F 73 74 65 72 64	The hex representation of the copyright text; in ASCII: "Copyright © 2001 by Eric Mosterd"
00 FF 02 13	Designates another copyright notice with a length of 19 bytes
41 6C 6C 20 52 69 67 68 74 73 20 52 65 73 65 72 76 65 64	The hex representation of the copyright text; in ASCII: "All Rights Reserved"
00 FF 01 20	Designates a text event of 32 bytes in length
47 65 6E 65 72 61 74 65 64 20 62 79 20 4E 6F 74 65 57 6F 72 74 68 79 20 43 6F 6D 70 6F 73 65 72	The hex representation of the text event; in ASCII: "Generated by NoteWorthy Composer"
00 B0 07 7F	Designates a Control or Mode change on channel 1; in this case, it sets the volume to 127
00 B0 0A 40	Designates a Control or Mode change on channel 1; in this case, it sets the pan positioning or balance to 64
00 FF 51 03 07 A1 20	Designates the tempo (0x00 FF 51 03) and sets it to 500000 (0x07 A1 20) microseconds per beat
00 FF 58 04 04 02 18 08	Designates the time signature (0x00 FF 58 04) and sets it to 4 (0x04) 2 (0x02) 24 (0x18) 8 (0x08) or 4/4 (first number and 2 to the power of the second number), 24 clock ticks per dotted-quarter note and 8 thirty-second notes per MIDI quarter note
00 90 3C 5C	Turn on note on channel 1; the note is 3C or middle C and the velocity is 0x5C or 92, which has the notational equivalent of forte (<i>f</i>)
81 20	This pauses 0x81 + 0x20 or 160 ticks
90 3C 00 20	Turn on note middle C on channel 1 with a velocity of 0 then pauses 0x20 or 32 ticks; this command actually turns the note off and is identical to using the 0x80 3C 00 note off command
90 3E 5C	Turn on note D above middle C with a velocity of 92 (<i>f</i>)

81 20	This pauses 160 ticks
90 3E 00 20	Turn off note D above middle C and pause 32 ticks
90 40 5C	Turn on note E above middle C with a velocity of 92 (<i>f</i>)
81 20	This pauses 160 ticks
90 40 00 20	Turn off note E above middle C and pause 32 ticks
90 41 5C	Turn on note F above middle C with a velocity of 92 (<i>f</i>)
81 20	This pauses 160 ticks
90 41 00 20	Turn off note F above middle C and pause 32 ticks
90 43 5C	Turn on note G above middle C with a velocity of 92 (<i>f</i>)
81 20	This pauses 160 ticks
90 43 00 20	Turn off note G above middle C and pause 32 ticks
90 45 5C	Turn on note A above middle C with a velocity of 92 (<i>f</i>)
81 20	This pauses 160 ticks
90 45 00 20	Turn off note A above middle C and pause 32 ticks
90 47 5C	Turn on note B above middle C with a velocity of 92 (<i>f</i>)
81 20	This pauses 160 ticks
90 47 00 20	Turn off note B above middle C and pause 32 ticks
90 48 5C	Turn on note C above middle C with a velocity of 92 (<i>f</i>)
81 20	This pauses 160 ticks
90 48 00 20	Turn off note C above middle C and pause 32 ticks
90 47 5C	Turn on note B above middle C with a velocity of 92 (<i>f</i>)
81 20	This pauses 160 ticks
90 47 00 20	Turn off note B above middle C and pause 32 ticks
90 45 5C	Turn on note A above middle C with a velocity of 92 (<i>f</i>)
81 20	This pauses 160 ticks
90 45 00 20	Turn off note A above middle C and pause 32 ticks
90 43 5C	Turn on note G above middle C with a velocity of 92 (<i>f</i>)
81 20	This pauses 160 ticks
90 43 00 20	Turn off note G above middle C and pause 32 ticks
90 41 5C	Turn on note F above middle C with a velocity of 92 (<i>f</i>)
81 20	This pauses 160 ticks
90 41 00 20	Turn off note F above middle C and pause 32 ticks
90 40 5C	Turn on note E above middle C with a velocity of 92 (<i>f</i>)
81 20	This pauses 160 ticks
90 40 00 20	Turn off note E above middle C and pause 32 ticks
90 3E 5C	Turn on note D above middle C with a velocity of 92 (<i>f</i>)
81 20	This pauses 160 ticks
90 3E 00 20	Turn off note D above middle C and pause 32 ticks
90 3C 5C	Turn on note middle C with a velocity of 92 (<i>f</i>)
82 50	This pauses for 210 ticks
90 3C 00 00	Turn off note middle C
FF 2F 00	Designates the end of the track

Appendix B: EMNML Documentation

Introduction

Extensible Music Notation Markup Language (EMNML) was developed to provide a standard, entirely ASCII-based method for transporting and sharing sheet music via the Internet. Resembling the structure of HTML, it has a specialized set of tags designed to completely represent sheet music in text form.

The tags within EMNML were based as closely as possible to the actual name of the music notation they represent. If two notes need to be slurred, they would be placed within a **<SLUR>** tag. For example if you wanted to use EMNML to represent the following song:



The code would look very similar to HTML:

```
<EMNML>
<STAFF NAME="piano"/>
<CLEF STAFF="piano"/>
<KEY PITCH="C" TONE="Major"/>

<MEASURE STAFF="piano" NAME="1">
  <SLUR>
    <DYNAMIC VOLUME="f"/>
    <NOTE PITCH="D" OCTAVE="1" LENGTH="1/4"/>
    <NOTE PITCH="C" OCTAVE="1" LENGTH="1/4"/>
  </SLUR>

  <SLUR>
    <NOTE PITCH="B" OCTAVE="0" LENGTH="1/4"/>
    <NOTE PITCH="A" OCTAVE="0" LENGTH="1/4"/>
  </SLUR>
</MEASURE>
</EMNML>
```

As shown above, the note C and D are placed between the **<SLUR>** tags. These tags are enclosed by the **<MEASURE>** tag, which as its name implies, divides the score into measures. The **<MEASURE>** tag always references a staff. This is done to assure that if there are more than one staff in a score, the appropriate measures, notes, rests, etc. are attributed to the correct staff. In this case, all of the notes belong inside the first measure, which belongs in the staff called "piano".

For more examples of EMNML tags, please refer to the EMNML Tags section, which immediately follows this section on page 50.

EMNML Tags

ARRANGER	
Attributes:	
<i>(none)</i>	
Usage:	
<code><ARRANGER>Rudolf Pribil</ARRANGER></code>	
Description:	
As the name implies, the ARRANGER tag is used to store the information about the arranger of the score. This tag may be used more than once, if more than one arranger need to be stored.	
See Also:	
COMMENTS, COMPOSER, COPYRIGHT, PUBLISHER	

BAR	
Attributes:	
STAFF <i>(Required)</i>	This contains the name of the staff (as defined in the STAFF tag) where the bar line is to reside.
TYPE <i>(Required)</i>	Specifies one of the following types of bar lines: <div style="text-align: center;"> double section open section close : local repeat open : local repeat close [: master repeat open :] master repeat close </div>
Usage:	
<code><BAR STAFF="Violin I" TYPE=" "/></code>	
<code><BAR STAFF="Violin I" TYPE="master repeat open"/></code>	
Description:	
This tag is used to override the default single bar and allows for repeats.	
See Also:	
ENDING, REPEAT, STAFF	

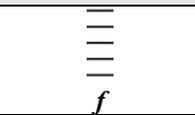
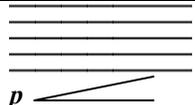
CHORD	
Attributes:	
ARPEGGIO <i>(Optional)</i>	When used, this attribute specifies that the notes within the chord are to be played as an arpeggio. Note: this attribute takes no arguments and has no values.
INTERVAL <i>(Optional)</i>	Specifies the interval (3-9) of the chord.
KEY <i>(Optional)</i>	Specifies the key (A-G) of the chord.
NAME <i>(Optional)</i>	Allows a name to be associated with the chord. For example: "D min 9".
TRIAD <i>(Optional)</i>	Specifies one of the following triads: <div style="text-align: center;"> I III V ii iii vi vii vii^o vii^d III⁺ </div>
Usage:	
<pre><CHORD> <NOTE PITCH="E" OCTAVE="1" LENGTH="1/4"/> <NOTE PITCH="C" OCTAVE="1" LENGTH="1/4"/> </CHORD></pre>	
<pre><CHORD> <NOTE PITCH="E" OCTAVE="1" LENGTH="1/4"/> <NOTE PITCH="C" OCTAVE="1" LENGTH="1/4"/> <NOTE PITCH="A" OCTAVE="0" LENGTH="1/4"/> </CHORD></pre>	
<pre><CHORD NAME="Cma7"> <NOTE PITCH="B" OCTAVE="0" LENGTH="1/4"/> <NOTE PITCH="G" OCTAVE="0" LENGTH="1/4"/> <NOTE PITCH="E" OCTAVE="0" LENGTH="1/4"/> <NOTE PITCH="C" OCTAVE="0" LENGTH="1/4"/> </CHORD></pre>	
<pre><CHORD KEY="C" TRIAD="I"> <NOTE PITCH="G" OCTAVE="0" LENGTH="1/4"/> <NOTE PITCH="E" OCTAVE="0" LENGTH="1/4"/> <NOTE PITCH="C" OCTAVE="0" LENGTH="1/4"/> </CHORD></pre>	
Description:	
<p>This tag allows the user to perform many important chord tasks within EMNML. In its simplest form, it allows the user to show that two or more notes should be played at the same time. In more advanced applications, they can name chord changes for piano parts or for instrument solos in Jazz.</p>	
See Also:	
NOTE, GUITAR, FIGURED	

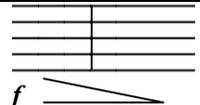
CLEF																							
Attributes:																							
SHIFT (Optional)	Optional: Tells whether or not the clef is to be octave-shifted either “up” or “down”.																						
STAFF (Required)	This contains the name of the staff (as defined in the STAFF tag) where the clef is to reside.																						
TYPE (Required)	One of the following types of clefs: <table border="0" style="width: 100%; text-align: center;"> <tr><td>treble</td><td></td></tr> <tr><td>bass</td><td></td></tr> <tr><td>percussion</td><td></td></tr> <tr><td>soprano</td><td></td></tr> <tr><td>mezzo-soprano</td><td></td></tr> <tr><td>alto</td><td></td></tr> <tr><td>tenor</td><td></td></tr> <tr><td>baritone</td><td></td></tr> <tr><td>subbass</td><td></td></tr> <tr><td>violin</td><td></td></tr> <tr><td>indefinite</td><td></td></tr> </table>	treble		bass		percussion		soprano		mezzo-soprano		alto		tenor		baritone		subbass		violin		indefinite	
treble																							
bass																							
percussion																							
soprano																							
mezzo-soprano																							
alto																							
tenor																							
baritone																							
subbass																							
violin																							
indefinite																							
Usage:																							
<code><CLEF STAFF="Violin I" TYPE="treble"/></code>																							
<code><CLEF STAFF="Violin I" TYPE="treble" SHIFT="up"/></code>																							
Description:																							
This tag defines the clef in which the staff is to be played. It can also be used to change the clef inside of the score. The clefs themselves can either be octave-shifted up or down (as illustrated in the second usage example above) for more legible notation.																							
See Also:																							
STAFF																							

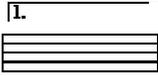
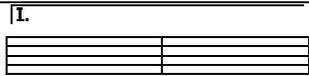
COMMENTS	
Attributes:	
	<i>(none)</i>
Usage:	
	<code><COMMENTS>As performed by the Philharmonia Slavonica.</COMMENTS></code>
Description:	
	This tag allows for comments about the score to be stored.
See Also:	
	ARRANGER, COMPOSER, COPYRIGHT, PUBLISHER

COMPOSER	
Attributes:	
	<i>(none)</i>
Usage:	
	<code><COMPOSER>Johann Sebastian Bach</COMPOSER></code>
Description:	
	This tag stores composer information about the score. If needed, it can be used multiple times.
See Also:	
	ARRANGER, COMMENTS, COPYRIGHT, PUBLISHER

COPYRIGHT					
Attributes:					
	<table border="1"> <tr> <td>OWNER <i>(Optional)</i></td> <td>The owner of the copyright.</td> </tr> <tr> <td>YEAR <i>(Optional)</i></td> <td>The four-digit year of the copyright.</td> </tr> </table>	OWNER <i>(Optional)</i>	The owner of the copyright.	YEAR <i>(Optional)</i>	The four-digit year of the copyright.
OWNER <i>(Optional)</i>	The owner of the copyright.				
YEAR <i>(Optional)</i>	The four-digit year of the copyright.				
Usage:					
	<code><COPYRIGHT YEAR="1990" OWNER="Regency Music"> All Rights Reserved. </COPYRIGHT></code>				
Description:					
	This tag stores copyright information about the score. If needed, it can be used multiple times.				
See Also:					
	ARRANGER, COMMENTS, COMPOSER, PUBLISHER				

DYNAMIC	
Attributes:	
CONTINUE <i>(Optional)</i>	When used, this attribute specifies that the dynamic is to continue into the next measure. Note: this attribute takes no arguments and has no values (see Usage below).
VARIANCE <i>(Optional)</i>	Specifies one of the following dynamic variances: <div style="text-align: center;"> <p>< <i>crescendo</i></p> <p>> <i>decrescendo</i></p> <p><i>dim.</i> <i>diminuendo</i></p> <p><i>fp</i> <i>fortepiano</i></p> </div>
VOLUME <i>(Optional)</i>	Specifies one of the following volumes: <div style="text-align: center;"> <p><i>ppp</i> <i>pianississimo</i></p> <p><i>pp</i> <i>pianissimo</i></p> <p><i>p</i> <i>piano</i></p> <p><i>mp</i> <i>mezzo piano</i></p> <p><i>mf</i> <i>mezzo forte</i></p> <p><i>f</i> <i>forte</i></p> <p><i>ff</i> <i>fortissimo</i></p> <p><i>fff</i> <i>fortississimo</i></p> </div>
Usage:	
<code><DYNAMIC VOLUME="f"/></code>	
<code><DYNAMIC VOLUME="p" VARIANCE="crescendo"/></code>	

<code><DYNAMIC VOLUME="f" VARIENCE="decrescendo" CONTINUE/></code>	
Description:	
<p>This tag controls the dynamics throughout the score. Other than the volume of particular notes, this tag can define dynamic variances like crescendo and decrescendo, which in turn can be spanned across measures using the CONTINUE attribute. If this attribute is used, it will continue the dynamic variance into the next measure.</p>	

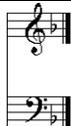
ENDING	
Attributes:	
<p>CONTINUE <i>(Optional)</i></p>	<p>When used, this attribute specifies that the ending is to continue into the next measure. Note: this attribute takes no arguments and has no values (see Usage below).</p>
<p>NAME <i>(Required)</i></p>	<p>Specifies the name of the ending. Usually this is the ending number.</p>
Usage:	
<p><code><ENDING NAME="1" /></code></p>	
<p><code><ENDING NAME="1" CONTINUE/></code></p>	
Description:	
<p>The ENDING tag is used to handle special endings. The user specifies a name, which is usually the number of the ending. The user may also use the CONTINUE attribute to specify that the special ending is longer than a measure. The ending will then continue into the next measure. Note: the ENDING tag is usually followed by a master repeat closed bar.</p>	
See Also:	
BAR, REPEAT	

FIGURED	
Attributes:	
ACCIDENTAL <i>(Optional)</i>	Specifies one of the following accidentals: sharp # flat b natural double sharp x double flat bb / -
INTERVAL <i>(Required)</i>	Specifies the intervals (3-9).
Usage:	
<code><FIGURED INTERVAL="6"/></code>	<u>6</u> — — — —
<code><FIGURED INTERVAL="3" ACCIDENTAL=""/></code>	<u>♯</u> — — — —
Description:	
This tag allows for the representation of figured bass in the score.	
See Also:	
CHORD	

FORMATTING	
Attributes:	
NAME <i>(Optional)</i>	Any text name.
VALUE <i>(Optional)</i>	A value, which represents the name.
Usage:	
<code><FORMATTING NAME="page layout" VALUE="landscape">8.5 x 11</FORMATTING></code>	
Description:	
This is a generic tag, which allows developers to store information not found within the specification. The example above shows how the tag could be used to store page layout information about the score.	

GUITAR	
Attributes:	
CAPO <i>(Optional)</i>	Specifies which of 12 frets, upon which the capo is to be placed
NAME <i>(Optional)</i>	Specifies the name of the guitar chord.
S1 <i>(Optional)</i>	Specifies which of 12 frets should be fingered on the first (topmost) string. If the string is to be played without fingering a fret, the number 0 is specified.
S2 <i>(Optional)</i>	Specifies which of 12 frets should be fingered on the second string. If the string is to be played without fingering a fret, the number 0 is specified.
S3 <i>(Optional)</i>	Specifies which of 12 frets should be fingered on the third string. If the string is to be played without fingering a fret, the number 0 is specified.
S4 <i>(Optional)</i>	Specifies which of 12 frets should be fingered on the fourth string. If the string is to be played without fingering a fret, the number 0 is specified.
S5 <i>(Optional)</i>	Specifies which of 12 frets should be fingered on the fifth string. If the string is to be played without fingering a fret, the number 0 is specified.
S6 <i>(Optional)</i>	Specifies which of 12 frets should be fingered on the sixth string. If the string is to be played without fingering a fret, the number 0 is specified.
Usage:	
<pre><GUITAR NAME="D" S2="0" S3="0" S4="2" S5="3" S6="2"/></pre>	<p>D</p>
Description:	
This tag allows the placement of guitar chord boxes within the score.	

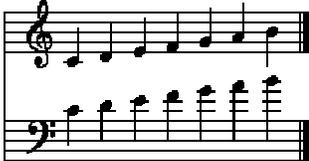
GRACE	
Attributes:	
NOTE <i>(Required)</i>	Specifies the grace notes (A-G).
STYLE <i>(Optional)</i>	Specifies one of the following styles of grace note to play: mordent trill acciaccatura
Usage:	
<code><GRACE NOTE="B"> <NOTE PITCH="C" OCTAVE="1" LENGTH="1/4"/> </GRACE></code>	
<code><GRACE NOTE="D" STYLE="acciaccatura"> <NOTE PITCH="C" OCTAVE="1" LENGTH="1/4"/> </GRACE></code>	
Description:	
This tag places a grace note in front of a note specified between the tag.	
See Also:	
NOTE	

KEY	
Attributes:	
PITCH (Required)	Specifies which note (A-G) is the key
STAFF (Optional)	Specifies the key for a particular staff. This is useful for doubled parts, especially in Jazz.
SCALE (Required)	Specifies whether the key is "Major" or "minor".
TONE (Optional)	Specifies whether the key is sharp or flat using the following notation: <div style="text-align: center;"> sharp # flat b </div>
Usage:	
<code><KEY PITCH="F" SCALE="Major"/></code>	
Description:	
This tag is used to specify the key signature within the score. It can also be used within the song to change the key as needed.	

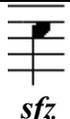
LYRICS	
Attributes:	
VERSE (Optional)	Specifies the verse number (1-8) of the lyric tag.
Usage:	
<code><LYRIC>These are some lyr - ics</LYRIC></code>	
<code><LYRIC VERSE="1">This is the first verse.</LYRIC></code> <code><LYRIC VERSE="2">This is the next verse.</LYRIC></code>	
Description:	
This tag allows lyrics to be placed within the score. As shown in the first example, use a hyphen to split a word into different beats. You may also specify up to 8 verses. The lyrics can either be placed directly after each individual note in the measure, or after all of the notes in the measure.	

MEASURE	
Attributes:	
NAME <i>(Optional)</i>	Specifies a name for the measure. Typically, this is the measure number.
STAFF <i>(Required)</i>	This contains the name of the staff (as defined in the STAFF tag) where the measure is to reside.
Usage:	
<code><MEASURE STAFF="Violin I" NAME="1">(notes/dynamics etc.)</MEASURE></code>	<div style="border: 1px solid black; display: inline-block; padding: 2px;">1</div> 
Description:	
This tag is used to split the score into to logical measures. The measures refer to a particular staff and can be named (usually with measure numbers) as shown in the example above.	

NOTE	
Attributes:	
ACCIDENTAL <i>(Optional)</i>	<p>Specifies if the note is one of the following accidentals:</p> <p style="text-align: center;">sharp # double sharp x flat b double flat bb natural</p>
DOTTED <i>(Optional)</i>	<p>Increments the length of the note based on the number of dots specified:</p> <p style="text-align: center;">1 . 2 .. 3 ...</p>
LENGTH <i>(Required)</i>	<p>Specifies one of the following durations for the note:</p> <p style="text-align: center;">double whole 2 whole 1 half 1/2 quarter 1/4 eighth 1/8 sixteenth 1/16 thirty-second 1/32 sixty-fourth 1/64</p>

<p>OCTAVE (Required)</p>	<p>Specifies one of the following octaves for the note:</p> <p style="text-align: center;">-5 -4 -3 -2 -1 0 (middle) = 1 2 3 4 5</p> 
<p>PITCH (Required)</p>	<p>Specifies the pitch (A-G) of the note.</p>
<p>STEM (Optional)</p>	<p>Overrides the default stem direction:</p> <p style="text-align: center;">up down</p>
<p>Usage:</p>	
<p><NOTE PITCH="A" OCTAVE="0" LENGTH="1/4" DOTTED=".." ACCIDENTAL="x"/></p>	
<p>Description:</p>	
<p>The NOTE tag specifies various attributes about a note, including its pitch, octave and length. Various other attributes may be specified using the note tag, and the NOTE tag may also be embedded within other tags to further modify how the note is played.</p>	
<p>See Also:</p>	
<p>CHORD, DYNAMIC, FIGURED, GRACE, ORNAMENTATION, SLIDE, SLUR, TIE, TUPLET</p>	

ORNAMENTATION	
Attributes:	
STYLE <i>(Required)</i>	<p>Specifies one of the following styles with which a note, or notes, is to be played:</p> <ul style="list-style-type: none"> • <i>staccato</i> <i>mezzo staccato</i> ^ <i>marcato</i> > <i>accent</i> ‘ <i>staccatissimo</i> — <i>tenuto</i> <i>sfz</i> <i>sf</i> <i>sforzando</i> <i>sfp</i> <i>sfortzando piano</i> / <i>smear</i> ~ <i>turn</i> <i>trill</i> <i>mordent</i> <i>inverted mordent</i> <i>mordent trill</i> <i>ascending trill</i> <i>descending trill</i> <i>ascending mordent trill</i> <i>descending mordent trill</i> <i>down bow</i> <i>up bow</i> <i>quindicesima</i> <i>15ma</i> <i>all’ottava</i> <i>8va</i> <i>ottava bassa</i> <i>8va bassa</i>
Usage:	

<pre><ORNAMENTATION STYLE="sfz"> <NOTE PITCH="C" OCTAVE="1" LENGTH="1/2"/> </ORNAMENTATION></pre>	
<pre><ORNAMENTATION STYLE="sfz"> <ORNAMENTATION STYLE="trill"> <NOTE PITCH="C" OCTAVE="1" LENGTH="1/4"/> </ORNAMENTATION> </ORNAMENTATION></pre>	
Description:	
<p>This tag allows the user to add special ornamentation elements to notes, in order to change how the note is played. It may be nested, as shown in the second example, and may contain a dynamic or a chord.</p>	
See Also:	
<p style="text-align: center;">CHORD, DYNAMIC, NOTE</p>	

PUBLISHER
Attributes:
<p style="text-align: center;"><i>(none)</i></p>
Usage:
<p style="text-align: center;"><code><PUBLISHER>Regency Music</PUBLISHER></code></p>
Description:
<p style="text-align: center;">This tag allows publisher information about the score to be stored.</p>

REPEAT	
Attributes:	
TYPE <i>(Required)</i>	<p>Specifies the type of repeat, or the location to repeat:</p> <p style="padding-left: 40px;">% repeat measure %% repeat 2 measures <i>D.S.</i> <i>del Signe</i> <i>Coda</i> <i>D.C. al Fine</i> <i>D.C. al Coda</i> <i>D.S. al Fine</i> <i>D.S. al Coda</i></p>
Usage:	
<code><REPEAT TYPE="Coda"/></code>	
Description:	
This tag specifies either a type of repeat, or a location to repeat to. It also handles single and double measure repeats.	
See Also:	
BAR, ENDING	

REST	
Attributes:	
DOTTED <i>(Optional)</i>	<p>Increments the length of a rest based on the number of dots specified:</p> <p style="text-align: center;">1 . 2 .. 3 ...</p>
LENGTH <i>(Optional)</i>	<p>Specifies the duration of the rest:</p> <p style="text-align: center;">whole 1 half 1/2 quarter 1/4 eighth 1/8 sixteenth 1/16 thirty-second 1/32 sixty-fourth 1/64</p>
SPAN <i>(Optional)</i>	Specifies the number of measures to span the rest.
Usage:	
<code><REST LENGTH="1/4" /></code>	
<code><REST SPAN="5" /></code>	
Description:	
This tag handles the various types of rests. It supports all standard rest lengths, as well as rests that span multiple measures, as illustrated in the second example.	

SLIDE	
Attributes:	
DIRECTION <i>(Required)</i>	Specifies the direction of the slide: ascending descending
Usage:	
<code><SLIDE DIRECTION="ascending"/></code>	
Description:	
The SLIDE tag allows for slides and falls in the same tag. The user specifies whether to slide up to a note, or down.	

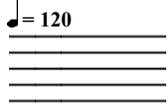
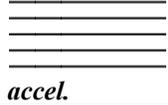
SLUR	
Attributes:	
CONTINUE <i>(Optional)</i>	When used, this attribute specifies that the slur is to continue into the next measure. Note: this attribute takes no arguments and has no values (see Usage below).
Usage:	
<code><SLUR></code> <code><NOTE PITCH="C" OCTAVE="1" LENGTH="1/4"/></code> <code><NOTE PITCH="A" OCTAVE="0" LENGTH="1/4"/></code> <code></SLUR></code>	
<code><SLUR CONTINUE></code> <code><NOTE PITCH="C" OCTAVE="1" LENGTH="1/4"/></code> <code></SLUR></code> <code></MEASURE></code> <code><MEASURE STAFF="Violin I" NAME="2"></code> <code><SLUR></code> <code><NOTE PITCH="C" OCTAVE="1" LENGTH="1/4"/></code> <code></SLUR></code>	
Description:	
This tag allows two or more notes to be slurred together. The notes can either be in the same measure, or adjacent measures. To slur notes in adjacent measures, the CONTINUE attribute is employed. As shown in the second example, the slur will start in its respective measure and will slur all notes from that point into the next measure.	
See Also:	
TIE	

STAFF	
Attributes:	
NAME (Required)	Specifies the name of a particular staff.
Usage:	
<STAFF NAME="Violin I"/>	
Description:	
This tag specifies the individual staves within a score. Tags like MEASURE , and CLEF reference the NAME attribute of the staff in order to keep associated information with the appropriate staff.	
See Also:	
CLEF, MEASURE	

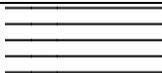
SUBTITLE	
Attributes:	
(none)	
Usage:	
<SUBTITLE>in F Major (BMV 1043)</SUBTITLE>	
Description:	
Similar to the TITLE tag, the SUBTITLE tag allows for a minor title heading, that follows underneath the score's title, as illustrated in the example above.	
See Also:	
TITLE	

TEMPO	
Attributes:	
BEAT (Optional)	A numeric value (40-255) that represents the tempo.
CONTINUE (Optional)	When used, this attribute specifies that the tempo is to continue into the next measure. Note: this attribute takes no arguments and has no values (see Usage below).
STYLE (Optional)	Specifies one of the following tempo styles: <i>largo</i> <i>larghetto</i> <i>adagio</i> <i>lento</i> <i>andante</i> <i>andantino</i> <i>moderato</i> <i>allegretto</i>

	<p><i>allegro</i> <i>presto</i> <i>prestissimo</i></p>
<p>NOTE (Optional)</p>	<p>Used in conjunction with the BEAT attribute, this specifies which note it to receive the number of beats specified:</p> <p>double whole 2 whole 1 half 1/2 quarter 1/4 eighth 1/8 sixteenth 1/16 thirty-second 1/32 sixty-fourth 1/64</p>
<p>VARIANCE (Optional)</p>	<p>Specifies one of the following tempo variances:</p> <p><i>a tempo</i> <i>accelerando</i> <i>allargando</i> <i>breath mark</i> // <i>caesura</i> <i>fermata</i> <i>rallentando</i> <i>ritardando</i> <i>ritenuto</i> <i>rubato</i> <i>stringendo</i></p>
<p>Usage:</p>	
<p><TEMPO STYLE="allegro"/></p>	<p><i>allegro</i> _____ _____ _____ _____</p>

<TEMPO NOTE="1/4" BEAT="120"/>	
<TEMPO VARIENCE="accelerando"/>	
<TEMPO VARIENCE="ritardando" CONTINUE/>	
Description:	
<p>This tag handles all notation that deals with the tempo of the score. The user can define their own tempo by using the BEAT and NOTE attributes, or they can use a predefined tempo as specified in the STYLE attribute. This tag, using the VARIENCE attribute, also handles tempo variances. As illustrated in the final example, if a user wishes to span a tempo variance across multiple measures, they can use the CONTINUE attribute, which will sustain the variance into the next measure.</p>	

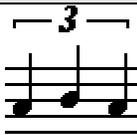
TEXT	
Attributes:	
LEVEL <i>(Optional)</i>	Specifies the level (1-3) of the text, either above or below the staff, as defined in the LOCATION attribute.
LOCATION <i>(Required)</i>	Specifies where the text should be placed: <p style="text-align: center;">above below</p>
SIZE <i>(Optional)</i>	Specifies the size (1-5) of the text.
STYLE <i>(Optional)</i>	Specifies the appearance of the text: <p style="text-align: center;">bold <i>italics</i> <u>underline</u> <i>bold-italics</i> <u>bold-underline</u> <i><u>bold-italics-underline</u></i></p>
Usage:	
<TEXT LOCATION="above" SIZE="2" STYLE="bold-italics">a due</TEXT>	<i>a due</i>

	
Description:	
This is a generic tag that allows the user to specify text in the score. This is useful for giving performance instructions. Note: this tag should not be used for lyrics as there is a separate tag defined for that task.	
See Also:	
LYRICS	

TIE	
Attributes:	
CONTINUE <i>(Optional)</i>	When used, this attribute specifies that the tie is to continue into the next measure. Note: this attribute takes no arguments and has no values (see Usage below).
Usage:	
<pre><TIE> <NOTE PITCH="C" OCTAVE="1" LENGTH="1/4"/> <NOTE PITCH="C" OCTAVE="1" LENGTH="1/4"/> </TIE></pre>	
<pre><TIE CONTINUE> <NOTE PITCH="C" OCTAVE="1" LENGTH="1/4"/> </TIE> </MEASURE> <MEASURE STAFF="Violin I" NAME="2"> <TIE> <NOTE PITCH="C" OCTAVE="1" LENGTH="1/4"/> </TIE></pre>	
Description:	
Similar to the SLUR tag, this tag connects two or more notes together. The notes can be in the same measure, or can be tied in adjacent measures by using the CONTINUE attribute. This will tie all of the notes into the next measure.	

TIME	
Attributes:	
BEATS <i>(Optional)</i>	Specifies the number of beats (1-20) per measure (top number in the time signature).
NOTE <i>(Optional)</i>	Specifies which note receives the beat (bottom number in the time signature): <div style="text-align: center;"> whole 1 half 1/2 quarter 1/4 eighth 1/8 sixteenth 1/16 thirty-second 1/32 sixty-fourth 1/64 </div>
STYLE <i>(Optional)</i>	Specifies a time style: <div style="text-align: center;"> C common cut <i>alla breve</i> </div>
Usage:	
<code><TIME BEATS="4" NOTE="1/4"/></code>	$\frac{4}{4}$
<code><TIME STYLE="alla breve"/></code>	$\frac{2}{2}$
Description:	
This tag is used to represent the time signature. The user can specify their own meter by using the BEAT and NOTE attributes, or they can use one of the time styles.	

TITLE
Attributes:
<i>(none)</i>
Usage:
<TITLE>Brandenburg Concerto No. 2</TITLE>
Description:
This tag is used to store the title of the score.
See Also:
SUBTITLE

TUPLET	
Attributes:	
TYPE <i>(Required)</i>	Number that designates the type of tuplet: 2 = duplet 3 = triplet ...
Usage:	
<TUPLET TYPE="3"> <NOTE PITCH="A" OCTAVE="0" LENGTH="1/4"/> <NOTE PITCH="B" OCTAVE="0" LENGTH="1/4"/> <NOTE PITCH="A" OCTAVE="0" LENGTH="1/4"/> </TUPLET>	
Description:	
This tag is used to represent tuplets, the most common being the triplet. The user specifies the number of notes or rests within the TYPE attribute, and is then required to put a combination of that number of notes or rests within the TUPLET tags.	

Appendix C: EMNML Data Type Definition

```
<?XML VERSION="1.0" STANDALONE="yes"?>
```

```
<!-- This is the main EMNML tag (analogous to the HTML tag) -->  
<!DOCTYPE EMNML [
```

```
<!--
```

```
    Extensible Music Notation Markup Language (EMNML)
```

```
    Author:          Eric Mosterd  
                   Computer Science Department  
                   University of South Dakota
```

```
    Purpose:         This document type definition (DTD) is part of the EMNML project for my graduate thesis, which, in turn,  
                   is for partial fulfillment of my Master of Arts degree.
```

```
    Description:     EMNML is designed to be one of the most complete markup languages for music score notation. Its  
                   development centers completely on the end user and was designed to be easily read without the assistance  
                   of a viewer.
```

```
    Revision:
```

```
        Version:    1.0 alpha 5  
        Last Update: 4/19/2001  
        Changes:
```

```
                The following changes were made after the defense of this thesis and were recommended by  
                the committee:
```

- Added an optional STAFF attribute to the KEY tag (it was previously removed) that allows for the key to be changed in a particular staff. This is useful for doubled parts, especially in Jazz where a tenor saxophone may be required to play the flute, etc.
- Added an optional STEM attribute that can be used to override the default stem direction in the NOTE tag.

```
        Version:    1.0 alpha 4  
        Last Update: 4/1/2001  
        Changes:
```

- Changed the arpeggio attribute in the CHORD tag to have no arguments
- Rest tag now allows for multiple measure rests
- Added more clefs to the CLEF tag and an option for octave shifts
- Removed MEASURE from the list of included tags within the DYNAMIC tag

- Modified the NOTE attribute of the TIME tag to use the same format as the LENGTH attribute in the NOTE tag and the NOTE attribute in the TEMPO tag
- Moved fp from ORNAMENTATION to DYNAMIC
- Added sf (a.k.a. sfz) to ORNAMENTATION as well as the CHORD and the DYNAMIC as allowed tags
- Added an ENDING tag to handle special (e.g. numbered) endings
- Removed the length "double whole" from the REST tag
- Changed the NAME attribute to STYLE in the TEMPO tag
- Fixed some more syntax errors
- Removed STAFF attribute from KEY

Version: 1.0 alpha 3
 Last Update: 3/21/2001
 Changes:

All changes are due to results of usability testing

- Modified all tags (TIE, SLUR, DYNAMIC, TEMPO, etc.) that use the CONTINUE attribute; now just uses the CONTINUE attribute with no arguments to continue to the next tag

Version: 1.0 alpha 2
 Last Update: 2/26/2001
 Changes:

- added more levels to the TEXT element (similar to the LYRIC element)
- added an element for figured bass
- updated example comments to show proper syntax
- updated SLUR, TIE, DYNAMIC, TEMPO, CHORD to properly span measures (see examples for more detail)
- fixed a syntax error in the KEY tag
- updated the TIME element:
 - removed STAFF attribute as time should change for all staves
 - removed the 3 option in NOTE attribute
 - added text notes to NOTE attribute (e.g. whole, half)
- major updates to the CHORD element:
 - added "yes" or "no" options to ARPEGGIO attribute
 - added TRIAD attribute (see example for details)
 - added KEY attribute (see example for details)
 - added NAME attribute (see example for details)
- fixed spelling error in ORNAMENTATION element
- changed thirtysecond and sixtyfourth to thirty-second and sixty-fourth respectively

Version: 1.0 alpha 1

Last Update: 1/26/2001
Changes: Initial Draft

Copyright: Eric Mosterd, ©2001. All rights reserved. You may freely use/distribute this DTD so long as this header remains intact and at the beginning of this file.

-->

<!-- Listed below are some common/useful entities: -->

<!ENTITY lt "&#60;">
<!ENTITY gt ">">
<!ENTITY amp "&#38;">
<!ENTITY quot """>

<!-- Begin EMNML elements and attributes -->
<!--

Header information entities

These entities are for the non-notational information about the score (e.g. title, composer).
-->

<!--
Description: the TITLE element is for the title of the score.
Usage: <TITLE>Brandenburg Concerto No. 2</TITLE>
-->
<!ELEMENT TITLE (#PCDATA)>

<!--
Description: the SUBTITLE element is for the subtitle of the score (e.g. opus number)
Usage: <SUBTITLE>in F Major (BMV 1043)</SUBTITLE>
-->
<!ELEMENT SUBTITLE (#PCDATA)>

<!--
Description: the COMPOSER, ARRANGER, and PUBLISHER elements allow you to store information about the score pertaining to those who have developed it

```

Usage:      <COMPOSER>Johann Sebastian Bach</COMPOSER>
            <ARRANGER>Rudolf Pribil</ARRANGER>
            <PUBLISHER>Regency Music</PUBLISHER>
-->
<!ELEMENT COMPOSER (#PCDATA)>
<!ELEMENT ARRANGER (#PCDATA)>
<!ELEMENT PUBLISHER (#PCDATA)>

<!--
Description:  the COPYRIGHT element allows you to list any copyright information and comments

Usage:      <COPYRIGHT YEAR="1990" OWNER="Regency Music">All Rights Reserved.</COPYRIGHT>
-->
<!ELEMENT COPYRIGHT (#PCDATA)>
  <!-- this is for the copyright year -->
  <!ATTLIST COPYRIGHT YEAR CDATA "2001">
  <!-- this is for the name of the person or owner who owns the copyright -->
  <!ATTLIST COPYRIGHT OWNER CDATA>

<!--
Description:  the FORMATTING tag is for future use of EMNML and allows developers to store such information as the layout,
size and other formatting attributes pertaining to the score

Usage:      <FORMATTING NAME="page layout" VALUE="landscape">8.5 X 11</FORMATTING>
-->
<!ELEMENT FORMATTING (#PCDATA)>
  <!ATTLIST FORMATTING NAME CDATA>
  <!ATTLIST FORMATTING VALUE CDATA>

<!--
Description:  the COMMENTS tag is a generic tag for storing information about the score
such as editorial comments, etc.

Usage:      <COMMENTS>As performed by the Philharmonia Slavonica.</COMMENTS>
-->
<!ELEMENT COMMENTS (#PCDATA)>

<!--
EMNML Notational Elements

```

The following elements deal directly with score notation
-->

```
<!--
  Description:  the STAFF element identifies the individual staves within a score

  Usage:       <STAFF NAME="Violin I"/>
-->
<!ELEMENT STAFF EMPTY>
  <!ATTLIST STAFF NAME ID #REQUIRED>
```

```
<!--
  Description:  the CLEF element is separate from the staff so that it can be changed
                throughout the score

  Usage:       <CLEF STAFF="Violin I" TYPE="treble"/>
                or
                <CLEF STAFF="Violin I" TYPE="treble" SHIFT="up">

                (note:  the example above would allow for an octave shift on the treble clef;  this is
                usually done when the piece is written too high or low to be legible on a normal
                clef)
-->
```

```
<!ELEMENT CLEF EMPTY>
  <!ATTLIST CLEF STAFF IDREF #REQUIRED>
  <!ATTLIST TYPE ("treble" | "bass" | "percussion" | "soprano" | "mezzo-soprano" | "alto" | "tenor" | "baritone" | "subbass" |
"violin" | "indefinite") #REQUIRED>
  <!ATTLIST SHIFT ("up" | "down")>
```

```
<!--
  Description:  the TIME element identifies the time signature;  it can be used anywhere within the score

  Usage:       <TIME STYLE="alla breve"/>
                or
                <TIME BEATS="2" NOTE="1/2"/>
-->
```

```
<!ELEMENT TIME EMPTY>
  <!ATTLIST TIME BEATS ("1" | "2" | "3" | "4" | "5" | "6" | "7" | "8" | "9" | "10" | "11" | "12" | "13" | "14" | "15" | "16" |
"17" | "18" | "19" | "20")>
```

```

<!ATTLIST TIME NOTE ("whole" | "half" | "quarter" | "eighth" | "sixteenth" | "thirty-second" | "sixty-fourth" | "1" | "1/2" |
"1/4" | "1/8" | "1/16" | "1/32" | "1/64")>
<!ATTLIST TIME STYLE ("C" | "common" | "cut" | "alla breve")>

<!--
Description:  the KEY element defines the key signature and can be used anywhere within the
score

Usage:       <KEY PITCH="F" SCALE="Major"/>
-->
<!ELEMENT KEY EMPTY>
<!ATTLIST KEY PITCH ("A" | "B" | "C" | "D" | "E" | "F" | "G") #REQUIRED>
<!ATTLIST KEY TONE ("sharp" | "#" | "flat" | "b")>
<!ATTLIST KEY SCALE ("Major" | "minor") #REQUIRED>
<!ATTLIST KEY STAFF IDREF>

<!--
Description:  the TEMPO element defines the tempo via a description or numeric beats per
minute; it also handles tempo variances such as ritardando, a tempo, etc.

Usage:       <TEMPO STYLE="allegro"/>
              or
              <TEMPO NOTE="1/4" BEAT="120"/>
              or
              <TEMPO VARIANCE="accelerando"/>
              or
              <TEMPO VARIANCE="ritardando" CONTINUE/>

              (note: the CONTINUE attribute is used to span across measures; in this example, if the ritardando spans two
              measures, you would use the CONTINUE attribute to continue the ritardando by first using the example
              above and then by using one of the following:

              <TEMPO VARIANCE="ritardando" CONTINUE/> (to continue the ritardando to another measure)
              or
              <TEMPO VARIANCE="ritardando"/> (to end the ritardando in this measure0
              )
-->
<!ELEMENT TEMPO EMPTY>
<!ATTLIST TEMPO NOTE ("double whole" | "whole" | "half" | "quarter" | "eighth" | "sixteenth" | "thirty-second" | "sixty-fourth"
| "2" | "1" | "1/2" | "1/4" | "1/8" | "1/16" | "1/32" | "1/64")>
<!-- this should be a range from 40-255 -->

```

```

<!ATTLIST TEMPO BEAT CDATA>
<!ATTLIST TEMPO STYLE ("largo" | "larghetto" | "adagio" | "lento" | "andante" | "andantino" | "moderato" | "allegretto" |
"allegro" | "presto" | "prestissimo")>
<!ATTLIST TEMPO VARIANCE ("a tempo" | "accelerando" | "allargando" | "breath mark" | "/" | "caesura" | "fermata" |
"rallentando" | "ritardando" | "ritenuto" | "rubato" | "stingendo")>
<!ATTLIST TEMPO CONTINUE ("true") #IMPLIED>

<!--
Description:   the MEASURE element specifies a measure and contains all note elements, lyrics, and note modifiers

              note:           that the name is not a unique identifier (like STAFF) is, rather it is merely there to help
                              composers keep track of where they are, kind of like a comment

Usage:        <MEASURE STAFF="Violin I" NAME="1">(here is where you would find the note, rest, and etc. elements</MEASURE>
-->
<!ELEMENT MEASURE (NOTE*, REST*, DYNAMIC*, TUPLET*, SLUR*, TIE*, LYRICS*, TEXT*, CLEF*, KEY*, TEMPO*, SLIDE*, CHORD*, ORNAMENTATION*)>
<!ATTLIST MEASURE STAFF IDREF #REQUIRED>
<!ATTLIST MEASURE NAME CDATA>

<!--
Description:   the NOTE element defines a specific note as well as any modifier to that specific note

Usage:        <NOTE PITCH="A" LENGTH="1/4" DOTTED=".." ACCIDENTAL="x" OCTAVE="0"/>
-->
<!ELEMENT NOTE EMPTY>
<!ATTLIST NOTE PITCH ("A" | "B" | "C" | "D" | "E" | "F" | "G") #REQUIRED>
<!ATTLIST NOTE LENGTH ("double whole" | "whole" | "half" | "quarter" | "eighth" | "sixteenth" | "thirty-second" | "sixty-
fourth" | "2" | "1" | "1/2" | "1/4" | "1/8" | "1/16" | "1/32" | "1/64") #REQUIRED>
<!ATTLIST NOTE DOTTED ("1" | "2" | "3" | "." | ".." | "...")>
<!ATTLIST NOTE ACCIDENTAL ("sharp" | "#" | "double sharp" | "x" | "flat" | "b" | "double flat" | "bb" | "natural")>

<!--
              the OCTAVE attribute defines the octave of the note:

                    0 represents middle C
                    1 represents C above middle C
                    -1 represents C below middle C, etc.
-->
<!ATTLIST NOTE OCTAVE ("-5" | "-4" | "-3" | "-2" | "-1" | "0" | "1" | "2" | "3" | "4" | "5") #REQUIRED>
<!ATTLIST NOTE STEM ("up" | "down")>

<!--

```

Description: the DYNAMIC element supports both dynamic markings like forte, but also dynamic variances like crescendo

Usage: <DYNAMIC VOLUME="f"/>
or
<DYNAMIC VOLUME="p" VARIANCE="crescendo"/>
or
<DYNAMIC VOLUME="p" VARIANCE="crescendo" CONTINUE/>

(note: the above example would be used to span a crescendo across two or more measures. The first starts at a dynamic of piano and crescendos until it reaches the next crescendo, which is at a dynamic of forte)

-->

```
<!ELEMENT DYNAMIC EMPTY>
  <!ATTLIST DYNAMIC VOLUME ("ppp" | "pianississimo" | "pp" | "pianissimo" | "p" | "piano" | "mp" | "mezzo piano" | "mf" | "mezzo
forte" | "f" | "forte" | "ff" | "fortissimo" | "fff" | "fortississimo")>
  <!ATTLIST DYNAMIC VARIANCE ("<" | "crescendo" | ">" | "decrescendo" | "dim." | "diminuendo" | "fp" | "fortepiano")>
  <!ATTLIST DYNAMIC CONTINUE ("true") #IMPLIED>
```

<!--

Description: the ORNAMENTATION element was originally an attribute of the NOTE element, however there is a need to have multiple styles per note

Usage: <ORNAMENTATION STYLE="sfz">
<ORNAMENTATION STYLE="trill">(place notes here)</ORNAMENTATION>
</ORNAMENTATION>

-->

```
<!ELEMENT ORNAMENTATION (NOTE+, ORNAMENTATION*, DYNAMIC*, CHORD*)>
  <!ATTLIST ORNAMENTATION STYLE ( "." | "staccato" | "mezzo staccato" | "^" | "marcato" | ">" | "accent" | "'" | "staccatissimo" |
"_" | "tenuto" | "sfz" | "sf" | "sforzando" | "sfp" | "sforzando piano" | "/" | "smear" | "~" | "turn" | "trill" | "mordent" |
"inverted mordent" | "mordent trill" | "ascending trill" | "descending trill" | "ascending mordent trill" | "descending mordent trill"
| "down bow" | "up bow" | "quindicesima" | "15ma" | "all'ottava" | "8va" | "ottava bassa" | "8va bassa") #REQUIRED>
```

<!--

Description: the REST element has two of the same attributes as that of the NOTE element

Usage: <REST LENGTH="whole"/>
or
<REST LENGTH="1/4" DOTTED=".."/>
or
<REST SPAN="5"/>
(allows a rest to span multiple measures)

-->

```

<!ELEMENT REST EMPTY>
  <!ATTLIST REST LENGTH ("whole" | "half" | "quarter" | "eighth" | "sixteenth" | "thirty-second" | "sixty-fourth" | "1" | "1/2" |
"1/4" | "1/8" | "1/16" | "1/32" | "1/64") #REQUIRED>
  <!ATTLIST REST DOTTED ("1" | "2" | "3" | "." | ".." | "...")>
  <!ATTLIST REST SPAN CDATA>

```

```

<!--
Description:   the CHORD element provides the facility to create a chord out of one or more notes

Usage:        <CHORD>(place one or more notes here)</CHORD>
              or
              <CHORD ARPEGGIO>(place one or more notes here)</CHORD>
              or
              <CHORD NAME="d min 9">(place one or more notes here)</CHORD>
              or
              <CHORD KEY="D" TRIAD="III">(place one or more notes here)</CHORD>
-->

```

```

<!ELEMENT CHORD (NOTE+, ORNAMENTATION*)>
  <!ATTLIST CHORD ARPEGGIO ("true") #IMPLIED>
  <!ATTLIST CHORD KEY CDATA>
  <!ATTLIST CHORD TRIAD ("I" | "III" | "v" | "ii" | "iii" | "vi" | "vii" | "viii" | "viid" | "III+")>
  <!ATTLIST CHORD INTERVAL ("3" | "4" | "5" | "6" | "7" | "8" | "9")>
  <!ATTLIST CHORD NAME CDATA>

```

```

<!--
Description:   the LYRIC element allows lyrics to placed below notes in a measure and allows for up to 8 verses

note:        you may use hyphens to split up syllables across notes

Usage:        <LYRIC>These are some lyr - ics</LYRIC>
              <LYRIC VERSE="1">This is the first verse</LYRIC>
              <LYRIC VERSE="2">This is the next verse</LYRIC>
-->

```

```

<!ELEMENT LYRICS #PCDATA>
  <!ATTLIST LYRICS VERSE ("1" | "2" | "3" | "4" | "5" | "6" | "7" | "8")>

```

```

<!--
Description:   the TEXT element allows notes or other music notation to placed above or below notes in a measure

Usage:        <TEXT LOCATION="above" SIZE="2" STYLE="bold-italics">a due</TEXT>
-->

```

```

<!ELEMENT TEXT #PCDATA>
  <!ATTLIST TEXT LOCATION ("above" | "below") #REQUIRED>
  <!ATTLIST TEXT LEVEL ("1" | "2" | "3") "1">
  <!ATTLIST TEXT SIZE ("1" | "2" | "3" | "4" | "5")>
  <!ATTLIST TEXT STYLE ("bold" | "italics" | "underline" | "bold-italics" | "bold-underline" | "bold-italics-underline" |
"italics-underlined")>

<!--
  Description:   the TIE element ties two or more of the same notes together

  Usage:        <TIE>(one or more notes here)</TIE>
                or
                <TIE CONTINUE>(one or more notes here)</TIE>

                (note: just like in the DYNAMIC and ORNAMENTATION elements, the CONTINUE attribute in the TIE element allows
                notes to be tied across measures)
-->
<!ELEMENT TIE (NOTE+, CHORD*, DYNAMIC*, ORNAMENTATION*)>
  <!ATTLIST TIE CONTINUE ("true") #IMPLIED>

<!--
  Description:   the SLUR element slurs two or more notes or rests together

  Usage:        <SLUR>(one or more notes or rests here)</SLUR>
                <SLUR CONTINUE>(one or more notes here)</SLUR>

                (note: just like the TIE element, the CONTINUE attribute in the SLUR element allows two or more notes to be
                slurred across measures)
-->
<!ELEMENT SLUR (NOTE+, CHORD*, DYNAMIC*, ORNAMENTATION*, REST*)>
  <!ATTLIST SLUR CONTINUE ("true") #IMPLIED>

<!--
  Description:   the GRACE element allows for grace notes

  Usage:        <GRACE NOTE="A">(a single note here)</GRACE>
-->
<!ELEMENT GRACE (NOTE)>
  <!ATTLIST GRACE NOTE ("A" | "B" | "C" | "D" | "E" | "F" | "G") #REQUIRED>
  <!ATTLIST GRACE STYLE ("mordent" | "trill" | "acciaccatura")>

```

```

<!--
Description:   the SLIDE element allows for glissandos and falls within the same tag

Usage:        <SLIDE DIRECTION="ascending">(a single note here)</SLIDE>
-->
<!ELEMENT SLIDE (NOTE)>
  <!ATTLIST SLIDE DIRECTION ("ascending" | "descending") "ascending" #REQUIRED>

<!--
Description:   the TUPLLET element allows for duplets, triplets, etc.

              note:   that the type designates kind of tuplet:

                    2 =   duplet
                    3 =   triplet, etc.

Usage:        <TUPLLET TYPE="3">(a combination of three notes or rests here)</TUPLLET>
-->
<!ELEMENT TUPLLET (NOTE+, REST*)>
  <!ATTLIST TUPLLET TYPE CDATA "3">

<!--
Description:   the GUITAR element allows for chord boxes

Usage:        <GUITAR NAME="D Maj" S2="0" S3="0" S4="2" S5="3" S6="2"/>
-->
<!ELEMENT GUITAR EMPTY>
  <!ATTLIST GUITAR NAME CDATA>
  <!ATTLIST GUITAR S1 ("0" | "1" | "2" | "3" | "4" | "5" | "6" | "7" | "8" | "9" | "10" | "11" | "12")>
  <!ATTLIST GUITAR S2 ("0" | "1" | "2" | "3" | "4" | "5" | "6" | "7" | "8" | "9" | "10" | "11" | "12")>
  <!ATTLIST GUITAR S3 ("0" | "1" | "2" | "3" | "4" | "5" | "6" | "7" | "8" | "9" | "10" | "11" | "12")>
  <!ATTLIST GUITAR S4 ("0" | "1" | "2" | "3" | "4" | "5" | "6" | "7" | "8" | "9" | "10" | "11" | "12")>
  <!ATTLIST GUITAR S5 ("0" | "1" | "2" | "3" | "4" | "5" | "6" | "7" | "8" | "9" | "10" | "11" | "12")>
  <!ATTLIST GUITAR S6 ("0" | "1" | "2" | "3" | "4" | "5" | "6" | "7" | "8" | "9" | "10" | "11" | "12")>
  <!ATTLIST GUITAR CAPO ("1" | "2" | "3" | "4" | "5" | "6" | "7" | "8" | "9" | "10" | "11" | "12")>

<!--
Description:   the BAR element allows for special bars or repeats and is used to override the default single bar

```

```

Usage:      <BAR TYPE="||"/>
-->
<!ELEMENT BAR EMPTY>
  <!ATTLIST BAR STAFF IDREF #REQUIRED>
  <!ATTLIST BAR TYPE ("||" | "double" | "[|]" | "section open" | "|]" | "section close" | "[:]" | "local repeat open" | "[:|" |
"local repeat close" | "[:]" | "master repeat open" | "[:]" | "master repeat close")>

<!--
Description:  the REPEAT element allows for different repeat markers as well as numbered endings

              note:  this tag should only be used within the measure tag

Usage:      <REPEAT TYPE="Coda"/>
-->
<!ELEMENT REPEAT EMPTY>
  <!ATTLIST REPEAT TYPE ("% " | "repeat measure" | "%%" | "repeat 2 measures" | "del Signe" | "Coda" | "D.C. al Fine" | "D.C. al
Coda" | "D.S. al Fine" | "D.S. al Coda") #REQUIRED>

<!-- Description:  the ENDING element allows for special (e.g. numbered) endings

Usage:      <ENDING NAME="1">
              or
              <ENDING NAME="1" CONTINUE>
-->
<!ELEMENT ENDING EMPTY>
  <!ATTLIST ENDING NAME CDATA>
  <!ATTLIST ENDING CONTINUE ("true") #IMPLIED>

<!--
Description:  the FIGURED element allows for the entry of figured bass text

              if you have more than one line of figured bass, place the top most interval first, and then follow it with the
              next interval

Usage:      <FIGURED INTERVAL="6">
              <FIGURED INTERVAL="3" ACCIDENTAL="/">
-->
<!ELEMENT FIGURED EMPTY>
  <!ATTLIST FIGURED INTERVAL ("3" | "4" | "5" | "6" | "7" | "8" | "9") #REQUIRED>
  <!ATTLIST FIGURED ACCIDENTAL ("sharp" | "#" | "flat" | "b" | "natural" | "double sharp" | "x" | "double flat" | "bb" | "/" | "-
")

```

Appendix D: Usability Tests

Usability Testing Task 1 – Sheet Music



Usability Testing Task 1 – EMNML Representation

```
<EMNML>

<STAFF NAME="piano"/>

<CLEF STAFF="piano" TYPE="treble"/>

<TIME BEATS="4" NOTE="4"/>

<KEY STAFF="piano" PITCH="C" SCALE="Major"/>

<TEMPO NOTE="quarter" BEATS="120"/>

<MEASURE STAFF="piano" NAME="1">
  <DYNAMIC VOLUME="f">
    <NOTE PITCH="C" OCTAVE="0" LENGTH="1/4"/>
    <NOTE PITCH="D" OCTAVE="0" LENGTH="1/4"/>
    <NOTE PITCH="E" OCTAVE="0" LENGTH="1/4"/>
    <NOTE PITCH="F" OCTAVE="0" LENGTH="1/4"/>
  </MEASURE>

<MEASURE STAFF="piano" NAME="2">
  <NOTE PITCH="G" OCTAVE="0" LENGTH="1/4"/>
  <NOTE PITCH="A" OCTAVE="0" LENGTH="1/4"/>
  <NOTE PITCH="B" OCTAVE="0" LENGTH="1/4"/>
  <NOTE PITCH="C" OCTAVE="1" LENGTH="1/4"/>
</MEASURE>

<MEASURE STAFF="piano" NAME="3">
  <NOTE PITCH="B" OCTAVE="0" LENGTH="1/4"/>
  <NOTE PITCH="A" OCTAVE="0" LENGTH="1/4"/>
  <NOTE PITCH="G" OCTAVE="0" LENGTH="1/4"/>
  <NOTE PITCH="F" OCTAVE="0" LENGTH="1/4"/>
</MEASURE>

<MEASURE STAFF="piano" NAME="4">
  <NOTE PITCH="E" OCTAVE="0" LENGTH="1/4"/>
  <NOTE PITCH="D" OCTAVE="0" LENGTH="1/4"/>
  <NOTE PITCH="C" OCTAVE="0" LENGTH="1/2"/>
</MEASURE>

</EMNML>
```

Usability Testing Task 2 – Sheet Music

SONATA IN A

Wolfgang Amadeus Mozart

Andante (♩=105)

The image shows two systems of musical notation for a piano sonata. The first system consists of a treble staff and a bass staff. The treble staff begins with a treble clef, a key signature of one sharp (F#), and a 3/4 time signature. The tempo is marked 'Andante' with a quarter note equal to 105 beats per minute. The dynamic marking is 'mp'. The second system also consists of a treble staff and a bass staff, with a dynamic marking of 'rit.'.

Usability Testing Task 2 – EMNML Representation

```
<EMNML>

<TITLE>SONTATA IN A</TITLE>

<COMPOSER>Wolfgang Amadeus Mozart</COMPOSER>

<STAFF NAME="piano-treble"/>
<STAFF NAME="piano-bass"/>

<CLEF STAFF="piano-treble" TYPE="treble"/>
<CLEF STAFF="piano-bass" TYPE="bass"/>

<KEY STAFF="piano-treble" PITCH="A" SCALE="Major"/>
<KEY STAFF="piano-bass" PITCH="A" SCALE="Major"/>

<TIME BEATS="3" NOTE="4"/>

<TEMPO NAME="andante" NOTE="quarter" BEATS="105"/>

<MEASURE STAFF="piano-treble" NAME="1">
  <DYNAMIC VOLUME="mp">
    <SLUR>
      <NOTE PITCH="C" OCTAVE="1" LENGTH="1/4" DOTTED="."/>
      <NOTE PITCH="D" OCTAVE="1" LENGTH="1/8"/>
      <NOTE PITCH="C" OCTAVE="1" LENGTH="1/4"/>
    </SLUR>
  </MEASURE>

<MEASURE STAFF="piano-bass" NAME="1">
  <DYNAMIC VOLUME="mp">
    <CHORD>
      <SLUR>
        <NOTE PITCH="E" OCTAVE="0" LENGTH="1/2" DOTTED="."/>
      </SLUR>
      <TIE>
        <NOTE PITCH="A" OCTAVE="-1" LENGTH="1/2" DOTTED="."/>
      </TIE>
    </CHORD>
  </MEASURE>

</EMNML>
```

```

        </CHORD>

</MEASURE>

<MEASURE STAFF="piano-treble" NAME="2">
  <!-- Note: the following SLUR will be connected to the previous SLUR tag in
measure 1. -->
  <SLUR CONTINUE="previous">
    <NOTE PITCH="E" OCTAVE="1" LENGTH="1/2"/>
  </SLUR>

  <SLUR>
    <NOTE PITCH="E" OCTAVE="1" LENGTH="1/4"/>
  </SLUR>
</MEASURE>

<MEASURE STAFF="piano-bass" NAME="2">
  <CHORD>
    <!-- Note: the following SLUR will be connected to the
previous SLUR tag in measure 1. -->
    <SLUR CONTINUE="previous">
      <NOTE PITCH="C" OCTAVE="0" LENGTH="1/2"/>
    </SLUR>

    <!-- Note: the following TIE will be connected to the
previous TIE tag in measure 1. -->
    <TIE CONTINUE="previous">
      <NOTE PITCH="A" OCTAVE="-1" LENGTH="1/2"/>
    </TIE>
  </CHORD>

  <REST LENGTH="1/4"/>
</MEASURE>

<MEASURE STAFF="piano-treble" NAME="3">
  <SLUR CONTINUE="previous">
    <NOTE PITCH="B" OCTAVE="0" LENGTH="1/4" DOTTED="."/>
    <NOTE PITCH="C" OCTAVE="1" LENGTH="1/8"/>
    <NOTE PITCH="B" OCTAVE="0" LENGTH="1/4"/>
  </SLUR>
</MEASURE>

<MEASURE STAFF="piano-bass" NAME="3">
  <TIE>
    <CHORD>
      <NOTE PITCH="E" OCTAVE="0" LENGTH="1/2" DOTTED="."/>
      <NOTE PITCH="G" OCTAVE="-1" LENGTH="1/2" DOTTED="."/>
    </CHORD>
  </TIE>
</MEASURE>

<MEASURE STAFF="piano-treble" NAME="4">
  <SLUR CONTINUE="previous">
    <NOTE PITCH="D" OCTAVE="1" LENGTH="1/2"/>
  </SLUR>

  <SLUR>
    <NOTE PITCH="D" OCTAVE="1" LENGTH="1/4"/>
  </SLUR>
</MEASURE>

<MEASURE STAFF="piano-bass" NAME="4">
  <TIE CONTINUE="previous">
    <CHORD>
      <NOTE PITCH="E" OCTAVE="0" LENGTH="1/2"/>
      <NOTE PITCH="G" OCTAVE="-1" LENGTH="1/2"/>
    </CHORD>
  </TIE>
</MEASURE>

```

```

        </CHORD>
        </TIE>
        <REST LENGTH="1/4"/>
    </MEASURE>

    <MEASURE STAFF="piano-treble" NAME="5">
        <SLUR CONTINUE="previous">
            <NOTE PITCH="A" OCTAVE="0" LENGTH="1/2"/>
        </SLUR>

        <SLUR>
            <NOTE PITCH="A" OCTAVE="0" LENGTH="1/4"/>
        </SLUR>
    </MEASURE>

    <MEASURE STAFF="piano-bass" NAME="5">
        <SLUR>
            <NOTE PITCH="F" OCTAVE="-1" LENGTH="1/2" DOTTED="."/>
        </SLUR>
    </MEASURE>

    <MEASURE STAFF="piano-treble" NAME="6">
        <SLUR CONTINUE="previous">
            <NOTE PITCH="B" OCTAVE="0" LENGTH="1/2"/>
        </SLUR>

        <SLUR>
            <NOTE PITCH="B" OCTAVE="0" LENGTH="1/4"/>
        </SLUR>
    </MEASURE>

    <MEASURE STAFF="piano-bass" NAME="6">
        <SLUR CONTINUE="previous">
            <NOTE PITCH="G" OCTAVE="-1" LENGTH="1/2" DOTTED="."/>
        </SLUR>
    </MEASURE>

    <MEASURE STAFF="piano-treble" NAME="7">
        <SLUR CONTINUE="previous">
            <NOTE PITCH="C" OCTAVE="1" LENGTH="1/2"/>
            <NOTE PITCH="E" OCTAVE="1" LENGTH="1/8"/>
            <NOTE PITCH="D" OCTAVE="1" LENGTH="1/8"/>
        </SLUR>
    </MEASURE>

    <MEASURE STAFF="piano-bass" NAME="7">
        <SLUR CONTINUE="previous">
            <NOTE PITCH="A" OCTAVE="-1" LENGTH="1/2"/>
        </SLUR>

        <REST LENGTH="1/4"/>
    </MEASURE>

    <MEASURE STAFF="piano-treble" NAME="8">
        <SLUR CONTINUE="previous">
            <NOTE PITCH="C" OCTAVE="1" LENGTH="1/2"/>
            <NOTE PITCH="B" OCTAVE="0" LENGTH="1/4"/>
        </SLUR>
    </MEASURE>

    <MEASURE STAFF="piano-bass" NAME="8">
        <SLUR>
            <NOTE PITCH="E" OCTAVE="0" LENGTH="1/2"/>
            <NOTE PITCH="D" OCTAVE="0" LENGTH="1/4"/>
        </SLUR>
    </MEASURE>

```

```

<MEASURE STAFF="piano-treble" NAME="9">
  <SLUR>
    <NOTE PITCH="C" OCTAVE="1" LENGTH="1/4" DOTTED="."/>
    <NOTE PITCH="D" OCTAVE="1" LENGTH="1/8"/>
    <NOTE PITCH="C" OCTAVE="1" LENGTH="1/4"/>
  </SLUR>
</MEASURE>

<MEASURE STAFF="piano-bass" NAME="9">
  <CHORD>
    <SLUR>
      <NOTE PITCH="E" OCTAVE="0" LENGTH="1/2" DOTTED="."/>
    </SLUR>
    <TIE>
      <NOTE PITCH="A" OCTAVE="-1" LENGTH="1/2" DOTTED="."/>
    </TIE>
  </CHORD>
</MEASURE>

<MEASURE STAFF="piano-treble" NAME="10">
  <SLUR CONTINUE="previous">
    <NOTE PITCH="E" OCTAVE="1" LENGTH="1/2"/>
  </SLUR>
  <SLUR>
    <NOTE PITCH="E" OCTAVE="1" LENGTH="1/4"/>
  </SLUR>
</MEASURE>

<MEASURE STAFF="piano-bass" NAME="10">
  <CHORD>
    <SLUR CONTINUE="previous">
      <NOTE PITCH="C" OCTAVE="0" LENGTH="1/2"/>
    </SLUR>
    <TIE CONTINUE="previous">
      <NOTE PITCH="A" OCTAVE="-1" LENGTH="1/2"/>
    </TIE>
  </CHORD>
  <REST LENGTH="1/4"/>
</MEASURE>

<MEASURE STAFF="piano-treble" NAME="11">
  <SLUR CONTINUE="previous">
    <NOTE PITCH="B" OCTAVE="0" LENGTH="1/4" DOTTED="."/>
    <NOTE PITCH="C" OCTAVE="1" LENGTH="1/8"/>
    <NOTE PITCH="B" OCTAVE="0" LENGTH="1/4"/>
  </SLUR>
</MEASURE>

<MEASURE STAFF="piano-bass" NAME="11">
  <TIE>
    <CHORD>
      <NOTE PITCH="E" OCTAVE="0" LENGTH="1/2" DOTTED="."/>
      <NOTE PITCH="G" OCTAVE="-1" LENGTH="1/2" DOTTED="."/>
    </CHORD>
  </TIE>
</MEASURE>

<MEASURE STAFF="piano-treble" NAME="12">
  <SLUR CONTINUE="previous">

```

```

        <NOTE PITCH="D" OCTAVE="1" LENGTH="1/2"/>
    </SLUR>

    <SLUR>
        <NOTE PITCH="D" OCTAVE="1" LENGTH="1/4"/>
    </SLUR>
</MEASURE>

<MEASURE STAFF="piano-bass" NAME="12">
    <TIE CONTINUE="previous">
        <CHORD>
            <NOTE PITCH="E" OCTAVE="0" LENGTH="1/2"/>
            <NOTE PITCH="G" OCTAVE="-1" LENGTH="1/2"/>
        </CHORD>
    </TIE>
    <REST LENGTH="1/4"/>
</MEASURE>

<MEASURE STAFF="piano-treble" NAME="13">
    <SLUR CONTINUE="previous">
        <NOTE PITCH="A" OCTAVE="0" LENGTH="1/2"/>
        <NOTE PITCH="B" OCTAVE="0" LENGTH="1/4"/>
    </SLUR>
</MEASURE>

<MEASURE STAFF="piano-bass" NAME="13">
    <SLUR>
        <NOTE PITCH="F" OCTAVE="-1" LENGTH="1/2"/>
        <NOTE PITCH="G" OCTAVE="-1" LENGTH="1/4"/>
    </SLUR>
</MEASURE>

<MEASURE STAFF="piano-treble" NAME="14">
    <TEMPO VARIANCE="ritardando"/>
    <SLUR CONTINUE="previous">
        <NOTE PITCH="C" OCTAVE="1" LENGTH="1/2">
        <NOTE PITCH="D" OCTAVE="1" LENGTH="1/4">
    </SLUR>
</MEASURE>

<MEASURE STAFF="piano-bass" NAME="14">
    <TEMPO VARIANCE="ritardando"/>
    <SLUR CONTINUE="previous">
        <NOTE PITCH="A" OCTAVE="-1" LENGTH="1/2">
    </SLUR>
    <REST LENGTH="1/4"/>
</MEASURE>

<MEASURE STAFF="piano-treble" NAME="15">
    <TEMPO VAIANCE="ritardando" CONTINUE="previous"/>
    <SLUR CONTINUE="previous">
        <NOTE PITCH="C" OCTAVE="1" LENGTH="1/2">
        <NOTE PITCH="B" OCTAVE="0" LENGTH="1/4">
    </SLUR>
</MEASURE>

<MEASURE STAFF="piano-bass" NAME="15">
    <TEMPO VARIANCE="ritardando" CONTINUE="previous"/>
    <SLUR>
        <NOTE PITCH="E" OCTAVE="0" LENGTH="1/2"/>
        <NOTE PITCH="D" OCTAVE="0" LENGTH="1/4"/>
    </SLUR>
</MEASURE>

<MEASURE STAFF="piano-treble" NAME="16">
    <TEMPO VARIANCE="ritardando" CONTINUE="previous"/>
    <SLUR CONTINUE="previous">
        <NOTE PITCH="A" OCTAVE="0" LENGTH="1/2" DOTTED="."/>
    </SLUR>
</MEASURE>

```

```
<MEASURE STAFF="piano-bass" NAME="16">
  <TEMPO VARIANCE="ritardando" CONTINUE="previous"/>
  <SLUR CONTINUE="previous"/>
    <CHORD>
      <NOTE PITCH="C" OCTAVE="0" LENGTH="1/2" DOTTED="."/>
      <NOTE PITCH="A" OCTAVE="-1" LENGTH="1/2" DOTTED="."/>
    </CHORD>
  </SLUR>
</MEASURE>

</EMNML>
```

Appendix E: *emnml2midi* Source Code

Source Code For MIDI2TXT Header File

```
#ifndef __MIDIIO__
#define __MIDIIO__

#include <stdio.h>

#ifndef MIDI_BUFSIZE
#define MIDI_BUFSIZE 1024
#endif

#ifdef __MSDOS__
#define __PC__
#endif

// opening modes of a midi file
#ifdef __PC__ // define this symbol if you compile on a PC
#define WRITE_BINARY "wb"
#define READ_BINARY "rb"
#else
#define WRITE_BINARY "w"
#define READ_BINARY "r"
#endif

const unsigned long MThd = 0x4D546864ul;
const unsigned long MTrk = 0x4D54726Bul;

// different standard midi formats
#define VERSION_MULTICHANNEL 0
#define VERSION_SINGLECHANNEL 1
#define VERSION_MULTISONG 2

#define OPTION_NOCONTROLS 1 // no control details but general information
#define OPTION_NOEVENTS 2 // no track events at all
#define OPTION_NOMETAEVENTS 4 // no meta details but general information
#define OPTION_NOSYSEVENTS 8 // no sysex details but general information
#define OPTION_NONOTEEVENTS 16 // no note events (8x or 9x)
#define OPTION_NOPOLYEVENTS 32 // no poly aftertouch events (Ax)
#define OPTION_NOCONTROLEVENTS 64 // no control events at all (Bx)
#define OPTION_NOPROGRAMEVENTS 128 // no program change events (Cx)
#define OPTION_NOAFTERTOUCEVENTS 256 // no aftertouch events (Dx)
#define OPTION_NOPITCHBENDEVENTS 512 // no pitchbend events (Ex)
#define OPTION_NOREALTIMEEVENTS 1024 // no realtime events (Fx)

// getchanel delivers a valid channel or:
#define NOCHANNEL (-1)
#define MULTICHANNEL (-2)
#define VALIDCHANNEL(ch) ((ch) >= 0 && (ch) <= 15)
#define gm_drumchannel 9
#define SAYCHANNEL(ch) ((ch) + 1) // 0..15 in midi format but spoken 1..16!

// for use of param what in function text()
#define meta_text 1
#define meta_copyright 2
#define meta_trackname 3
#define meta_instrument 4
#define meta_lyric 5
#define meta_marker 6
#define meta_cuepoint 7
#define meta_endtrack 0x2f

#define ctrl_highbank 0
#define ctrl_wheel 1
```

```

#define ctrl_breath 2
#define ctrl_foot 4
#define ctrl_portamentotime 5
#define ctrl_data 6
#define ctrl_volume 7
#define ctrl_balance 10
#define ctrl_expression 11
#define ctrl_lowbank 32
#define ctrl_hold 64
#define ctrl_reverb 91
#define ctrl_chorus 93
#define ctrl_datainc 96
#define ctrl_datadec 97
#define ctrl_lowrpn 100
#define ctrl_highrpn 101

#define balance_left 0
#define balance_center 64
#define balance_right 127

#define volume_mute 0
#define volume_full 127

#define bpm(ticks) (6000000.0 / (ticks))
#define ticks(beats) ((unsigned long)(6000000.0 / (beats)))

#define tempo_60bpm (1000000L)
#define tempo_120bpm (500000L)
#define tempo_240bpm (250000L)

#define pitchbend_maxdown 0x0000
#define pitchbend_center 0x2000
#define pitchbend_maxup 0x3fff

#define MIDI_PRELOAD 1
#define MIDI_DIRECTIO 0

#define NOTREALISTIC_PAUSE 0x1000000UL

class MidiBuffer
{
public:
MidiBuffer(const char* filename, FILE* f = 0, char preload = MIDI_DIRECTIO);
MidiBuffer(unsigned char* mididata, long mididatalen, char shouldfree = 0);
virtual ~MidiBuffer();

FILE* getf();
long midilength(); // > 0 if buffer contains data

int need(long pos, unsigned char* buf, int bufsize); // bufsize must be kb aligned!

protected:
const char *midiname_;
FILE* f_;
unsigned char* midibuf_;
char shouldfreemidibuf_; // 0=no, otherwise=yes

unsigned char shouldclose_; // 0=no, otherwise=yes
long filesize_;
};

class MidiRead : public MidiBuffer
{
public:
static const char* copyright();

MidiRead(const char* filename, FILE* f = 0, char preload = 0);
virtual ~MidiRead();

long seekmidihead(); // returns pos >= 0 if midi header found and seeks to header

```

```

virtual int run();
virtual int runhead();
virtual int runtrack(int trackno);
virtual int runevent(long trackend);

long getpos() { return pos_; }
long geteventpos() { return pos_; }
long getcurpos() { return curpos_; }
unsigned long getcurunit() { return curunit_; } // in midi units
unsigned long getcurmillisec() { return curms_; } // milliseconds
// Warning: getcurmillisec() only valid in first track and not using
// OPTION_NOREALTIMEEVENTS or OPTION_NOMETAEVENTS
unsigned long getcurbeat() { return unitsperbeat_ > 0 ? curunit_ / unitsperbeat_ : 0; }

virtual void head(unsigned version, unsigned tracks, unsigned unitperbeat);
virtual void track(int trackno, long length, int channel);
virtual void endtrack(int trackno); // closing track

virtual void time(unsigned long ticks); // delay between events

virtual void event(int what, int len = 0, unsigned char* data = 0);

// these are event categories:
virtual void meta(int what, int len, unsigned char* data);
// these are special meta events:
    virtual void text(int what, int len, char* whattext, unsigned char* txt);
    virtual void end(); // end of track command
    virtual void prefixchannel(unsigned char channel);
    virtual void prefixport(unsigned char port);
    virtual void seqnumber(unsigned int seqno);
    virtual void smpteofs(int hour, int min, int sec, int frame, int fracframe);
    virtual void tact(int nom, int denom, int unitsperbeat, int notes32perbeat);
    virtual void tempo(unsigned long microsecperbeat);
    virtual void key(int signature, int isminor); // C=0, -7=7flats +7=7sharps
virtual void program(int channel, int prg);
virtual void control(int channel, int what, int val); // general controls
// special controls:
    virtual void highbank(int channel, int val);
    virtual void wheel(int channel, int val);
    virtual void breath(int channel, int val);
    virtual void foot(int channel, int val);
    virtual void portamentotime(int channel, int val);
    virtual void data(int channel, int val);
    virtual void volume(int channel, int val);
    virtual void balance(int channel, int val);
    virtual void expression(int channel, int val);
    virtual void lowbank(int channel, int val);
    virtual void hold(int channel, int val);
    virtual void reverb(int channel, int val);
    virtual void chorus(int channel, int val);
    virtual void datainc(int channel, int val);
    virtual void datadec(int channel, int val);
    virtual void lowrpn(int channel, int val);
    virtual void highrpn(int channel, int val);
    virtual void pitchbendrange(int channel, int val);
virtual void noteon(int channel, int note, int vel);
virtual void noteoff(int channel, int note, int vel);
virtual void pitchbend(int channel, int val);
virtual void polyaftertouch(int channel, int note, int val);
virtual void aftertouch(int channel, int val);
virtual void songpos(unsigned pos);
virtual void songselect(unsigned char song);
virtual void tunerequest();
virtual void timingclock();
virtual void start();
virtual void cont();
virtual void stop();
virtual void activesense();
virtual void sysex(int len, unsigned char* sysdata);

```

```

// these are special sysex events:
    virtual void xgreset();
    virtual void gmreset();
    virtual void gsreset();
    virtual void gsexit();

virtual void endmidi(); // after last track
virtual void error(const char* msg);
virtual void warning(const char* msg);

virtual void percent(int perc);

int getchannel() { return curchannel_; }
void setchannel(int channel);

static const char* progname(int n, int channel);
static const char* notename(unsigned char note);

int options_;

int version_, tracks_, unitsperbeat_, trackno_;

void seek(long pos);
int getbyte();
unsigned getword();
unsigned long gettri();
unsigned long getlong();
unsigned long getdelta();
unsigned char* get(int len);
virtual unsigned char* need(int nbytes);

unsigned long microsec(unsigned long units, unsigned long msperbeats);
long units(unsigned long microsec, unsigned long msperbeats);

// use scanchannel only at start of track!
int scanchannel(unsigned long maxlen); // channel 0-15 or -1=no channel or -
2=multichannels

// use sysevent only directly after reading F0 or F7
int readsyseventlength(unsigned long maxlen);

protected:
unsigned char buf_[MIDI_BUFSIZE];
int buflen_, bufpos_;
int curchannel_;
unsigned long curunit_;
unsigned long curms_, currest_;
unsigned long timediv_; // 1000 * quarterunits_
unsigned long microsecperbeat_;
int percent_;
int lastcode_;
unsigned long tracklen_;
char skiptrack_; // set to 1 if want abort runtrack
char exit_;

long pos_, curpos_;
unsigned char curdeltalen_; // number of bytes read by recent getdelta() call

void calctime(unsigned long units, unsigned long& nextms, unsigned long& nextrest);
unsigned long calcunit(unsigned long ms, unsigned long msrest = 0);

};

class MidiWrite
{
public:
static const char* copyright();

MidiWrite(const char* filename);
virtual ~MidiWrite();

```

```

FILE* getf();

long getcurpos() { return curpos_; }
long getcurunit() { return curunit_; } // midi units
long getcurdelta() { return curdelta_; }
void cleardelta();
int trackcount();

void head(int version = 1, int tracks = 0, unsigned unitperbeat = 192);
void track();
void endtrack();

void event(int what, int len, unsigned char* data);

void text(int what, int len, unsigned char* txt);
void meta(int what, int len, unsigned char* data); // 0xff ....
virtual void prefixchannel(unsigned char channel);
virtual void prefixport(unsigned char port);
virtual void seqnumber(unsigned int seqno);
virtual void smpteofs(int hour, int min, int sec, int frame, int fracframe);
virtual void key(int signature, int isminor);
void tact(int nom, int denom, int v1, int v2);
void tempo(unsigned long microsecperbeat);
void end();
void program(int channel, int prg);
void control(int channel, int what, int val);
// these are special controls
void highbank(int channel, int val);
void wheel(int channel, int val);
void breath(int channel, int val);
void foot(int channel, int val);
void portamentotime(int channel, int val);
void data(int channel, int val);
void volume(int channel, int val);
void balance(int channel, int val);
void expression(int channel, int val);
void lowbank(int channel, int val);
void hold(int channel, int val);
void reverb(int channel, int val);
void chorus(int channel, int val);
void datainc(int channel, int val);
void datadec(int channel, int val);
void lowrpn(int channel, int val);
void highrpn(int channel, int val);
void pitchbendrange(int channel, int halfnotes);

void noteon(int channel, int note, int vel);
void noteoff(int channel, int note, int vel = 0);
void time(unsigned long ticks);
void pitchbend(int channel, int val);
void polyaftertouch(int channel, int note, int val);
void aftertouch(int channel, int val);
void songpos(unsigned pos);
void songselect(unsigned char song);
void tunerequest();
void timingclock();
void start();
void cont();
void stop();
void activesense();
void sysex(int syslen, unsigned char* sysdata);
void xgreset();
void gmreset();
void gsreset();
void gsexit();

void putbyte(unsigned char val);
void putcode(unsigned char code);
void putword(unsigned val);

```

```

void puttri(unsigned long val);
void putlong(unsigned long val);
void putdelta(unsigned long val);
void puttime();
void put(int len, unsigned char* c);
void seek(long pos);

virtual void error(const char* msg);
virtual void warning(const char* msg);

int unitsperquarter();

protected:
const char *midiname_;
FILE* f_;
long trackpos_, curpos_, filesize_;
int trackchannel_, trackcount_, lastcode_, endtrack_;

unsigned char buf_[MIDI_BUFSIZE];
int bufpos_, buflen_;

unsigned long curdelta_;
unsigned long curunit_;
int unitsperbeat_;

void flush();
};

class MidiCopy : public MidiRead
{
public:
MidiCopy(const char* filename, FILE* f = 0);

void setoutput(MidiWrite* dest);
void stopoutput();
MidiWrite* getoutput();

void mapchannel(int channel, int newchannel);
void ignorechannel(int channel);

virtual void head(unsigned version, unsigned tracks, unsigned unitperbeat);
virtual void track(int trackno, long length, int channel);
virtual void endtrack(int trackno);

virtual void time(unsigned long ticks);

virtual void event(int what, int len = 0, unsigned char* data = 0);

virtual void meta(int what, int len, unsigned char* data);
virtual void text(int what, int len, char* whattext, unsigned char* txt);
virtual void end();
virtual void prefixchannel(unsigned char channel);
virtual void prefixport(unsigned char port);
virtual void seqnumber(unsigned int seqno);
virtual void smpteofs(int hour, int min, int sec, int frame, int fracframe);
virtual void tact(int nom, int denom, int unitsperbeat, int notes32perbeat);
virtual void tempo(unsigned long microsecperbeat);
virtual void key(int signature, int isminor); // C=0, -7=7flats +7=7sharps
virtual void program(int channel, int prg);
virtual void control(int channel, int what, int val);
virtual void highbank(int channel, int val);
virtual void wheel(int channel, int val);
virtual void breath(int channel, int val);
virtual void foot(int channel, int val);
virtual void portamentotime(int channel, int val);
virtual void data(int channel, int val);
virtual void volume(int channel, int val);
virtual void balance(int channel, int val);
virtual void expression(int channel, int val);

```

```

virtual void lowbank(int channel, int val);
virtual void hold(int channel, int val);
virtual void reverb(int channel, int val);
virtual void chorus(int channel, int val);
virtual void datainc(int channel, int val);
virtual void datadec(int channel, int val);
virtual void noteon(int channel, int note, int vel);
virtual void noteoff(int channel, int note, int vel);
virtual void pitchbend(int channel, int val);
virtual void polyaftertouch(int channel, int note, int val);
virtual void aftertouch(int channel, int val);
virtual void songpos(unsigned pos);
virtual void songselect(unsigned char song);
virtual void tunerequest();
virtual void timingclock();
virtual void start();
virtual void cont();
virtual void stop();
virtual void activesense();
virtual void sysex(int syslen, unsigned char* sysdata);
virtual void gmreset();
virtual void gsreset();
virtual void gsexit();
virtual void xgreset();

protected:
MidiWrite* dest_;
int mapchannel_[16]; // channel 0-15 or events are ignored for invalid channel
};

class MidiSerial : public MidiRead
{
public:
MidiSerial(const char* filename, FILE* f = 0, char preload = 0);
virtual ~MidiSerial();

virtual int run(); // fire events in order of time

protected:
long *trackpos_;
long *trackendpos_;
long *tracktime_;
char *trackready_;
int *trackstatus_;

void clear();

void track(int trackno, long length, int channel);
void endtrack(int trackno); // closing track

};

#endif

```

Source Code For MIDI2TXT File

```
static char* version = "midi2txt v1.13 by Günter Nagler (" __DATE__ ")";

#include "midiio.hpp"
#include "miditime.hpp"
#include <stdlib.h>
#include <ctype.h>
#include <string.h>
#ifdef __MSDOS__
#include <dos.h>
#else
#endif

// values for info_
#define ONLY_ERRORS -2
#define ONLY_LYRICS -1
#define ONLY_CHUNK 0
#define ONLY_SHORT 1
#define ONLY_INFO 2
#define ALL_EVENTS 3

static FILE* outputf = NULL;

static const char* versioninfo(int version)
{
    switch(version)
    {
        case 0: return "single multichanneltrack";
        case 1: return "several tracks with seperated channels to play all at once";
        case 2: return "several multichanneltracks to play one by one";
        default: return "unknown version";
    }
}

class MidiPrint : public MidiRead
{
public:
    MidiPrint(char* name);
    virtual ~MidiPrint();

    virtual void head(unsigned version, unsigned tracks, unsigned unitperbeat);
    virtual void garbage(FILE* f, long datapos, long datalen);
    virtual void track(int trackno, long length, int channel);
    virtual void endtrack(int trackno);

    virtual void seqnumber(unsigned int seqno);
    virtual void text(int what, unsigned len, char* whattext, unsigned char* txt);
    virtual void meta(int what, unsigned len, unsigned char* data);
    virtual void meta(int what, FILE* f, long datapos, long datalen);
    virtual void end();
    virtual void prefixchannel(unsigned char channel);
    virtual void prefixport(unsigned char port);
    virtual void smpteofs(int mode, int hour, int min, int sec, int frame, int fracframe);
    virtual void tact(int nom, int denom, int unitsperbeat, int notes32perbeat);
    virtual void tempo(unsigned long ticks);
    virtual void key(int signature, int isminor);

    virtual void program(int prg, int channel);
    virtual void control(int channel, int what, int value);
    virtual void highbank(int channel, int val);
    virtual void wheel(int channel, int val);
    virtual void breath(int channel, int val);
    virtual void foot(int channel, int val);
    virtual void portamentotime(int channel, int val);
    virtual void data(int channel, int val);
    virtual void volume(int channel, int val);
};
```

```

virtual void balance(int channel, int val);
virtual void expression(int channel, int val);
virtual void lowbank(int channel, int val);
virtual void hold(int channel, int val);
virtual void reverb(int channel, int val);
virtual void chorus(int channel, int val);
virtual void datainc(int channel, int val);
virtual void datadec(int channel, int val);
virtual void lowrpn(int channel, int val);
virtual void highrpn(int channel, int val);
virtual void pitchbendrange(int channel, int val);
virtual void noteon(int channel, int note, int vel);
virtual void noteoff(int channel, int note, int vel);
virtual void time(unsigned long ticks);
virtual void pitchbend(int channel, int val);
virtual void polyaftertouch(int channel, int note, int val);
virtual void aftertouch(int channel, int val);
virtual void songpos(unsigned pos);
virtual void songselect(unsigned char song);
virtual void tunerequest();
virtual void timingclock();
virtual void start();
virtual void cont();
virtual void stop();
virtual void activesense();
virtual void sysex(unsigned syslen, unsigned char* sysdata);
virtual void sysex(FILE* f, long sysdatapos, long sysdatalen);
virtual void gmreset();
virtual void gsreset();
virtual void gsexit();
virtual void xgreset();
virtual void endmidi();
virtual void error(const char* msg);

virtual void percent(int perc);

void printchannel(int channel);
void printcurpos(unsigned long units);

int info_;
int printunits_;
int printms_;
int marktact_;
long errorcnt_;
int printdecimal_; // no program, notes names etc.
int printchannel_;

private:
int indent_;
MidiTimeTable* tactinfo_;

char hastempochanges_; // can't print milliseconds in tracks 2,3... if tempo changes

int nl_;
int endmark_; // each track should contain FF 2F 00 as last command

void indent();
void hex(FILE* f, long pos, long len, FILE* outputf);
void printxf();
int runxftrack(int trackno);

};

MidiPrint::MidiPrint(char* name) : MidiRead(name, NULL, MIDI_PRELOAD)
{
indent_ = 0;
info_ = ALL_EVENTS;
marktact_ = printms_ = 0;
nl_ = 1;
errorcnt_ = 0;

```

```

hastempochanges_ = 0;
printdecimal_ = 0;
printchannel_ = 0;
tactinfo_ = 0;
}

MidiPrint::~MidiPrint()
{
delete tactinfo_;
tactinfo_ = 0;
}

void MidiPrint::indent()
{
for (; indent_ > 0; indent_--)
fputc(' ', outputf);
}

void MidiPrint::head(unsigned version, unsigned tracks, unsigned unitperbeat)
{
if (marktact_)
{
tactinfo_ = new MidiTimeTable(midiname(), getf());
if (tactinfo_)
{
if (!tactinfo_>run())
{
delete tactinfo_;
tactinfo_ = 0;
}
}
}

if (midiname_)
fprintf(outputf, "// %s\n", midiname_);
if (getcurpos() != 14)
fprintf(outputf, " // Warning: %ld bytes garbage at beginning of file\n", getcurpos() -
14);
if (tracks_ == 0 && info_ == ONLY_ERRORS)
error("no track found");
if (info_ == ONLY_SHORT || info_ == ONLY_LYRICS || info_ == ONLY_ERRORS)
return;
fprintf(outputf, "mthd\n");
fprintf(outputf, " version %d // %s\n", version, versioninfo(version));
fprintf(outputf, " // %d track%s\n", tracks, tracks != 1 ? "s" : "");
fprintf(outputf, " unit %d // is 1/4\n", unitperbeat);
if (tracklen_ > 6)
{
fprintf(outputf, "// Warning: %ld bytes garbage in header:\n", tracklen_ - 6);
hex(getf(), getcurpos(), tracklen_ - 6, outputf);
}
fprintf(outputf, "end mthd\n");
}

void MidiPrint::track(int trackno, long /*length*/, int channel)
{
endmark_ = 0;
if (info_ == ONLY_SHORT || info_ == ONLY_LYRICS || info_ == ONLY_ERRORS)
return;
fprintf(outputf, "\nmtrk");
if (channel >= 0)
fprintf(outputf, "($%X)", channel+1);
else if (channel == MULTICHANNEL)
fprintf(outputf, " // multichannel track");
fprintf(outputf, " // track %d", trackno);

fprintf(outputf, "\n");
}

void MidiPrint::endtrack(int /*trackno*/)

```

```

{
if (!endmark_ && info_ != ONLY_CHUNK)
fprintf(outputf, " //Warning: end of track meta event missing\n");

if (info_ == ONLY_SHORT || info_ == ONLY_ERRORS)
return;
if (info_ == ONLY_LYRICS)
{
if (!nl_)
{
nl_ = 1;
fprintf(outputf, "\n");
}
return;
}
if (info_ >= ALL_EVENTS)
fprintf(outputf, "\n");
fprintf(outputf, "end mtrk\n");
}

void printchar(int c)
{
if (c < 0)
return;
if (c == '\\')
fprintf(outputf, "\\");
else if (c == '"')
fprintf(outputf, "\\");
else if (c != 0 && (isprint(c) || strchr(",,\"'[]{}~šá", c) != 0))
fputc(c, outputf);
else
fprintf(outputf, "\\x%02x", c);
}

void MidiPrint::seqnumber(unsigned int seqno)
{
if (info_ < ALL_EVENTS)
return;
indent();
fprintf(outputf, "seqnumber %u\n", seqno);
}

void MidiPrint::text(int what, unsigned len, char* whattext, unsigned char* txt)
{
if (info_ == ONLY_ERRORS)
return;
if (info_ == ONLY_LYRICS)
{
if (!txt)
return;
if (what == meta_lyric || what == meta_text)
{
char* bs;

while ((bs = (char*)memchr(txt, '\\', len)) != 0)
*bs = '\n';
while ((bs = (char*)memchr(txt, '/', len)) != 0)
*bs = '\n';
if (*txt == '@')
{
txt++; len--;
switch(*txt)
{
case 'K': fprintf(outputf, "\nKind: "); break;
case 'V': fprintf(outputf, "\nVersion: "); break;
case 'I': fprintf(outputf, "\nInformation: "); break;
case 'T': fprintf(outputf, "\nTitle: "); break;
case 'L': fprintf(outputf, "\nLanguage: "); break;
default: fprintf(outputf, "\n%c: ", *txt); break;
}
}
}
}
}

```

```

if (*txt)
{
    txt++;
    len--;
}
}
if (txt[len-1] == '\r')
txt[len-1] = '\n';
nl_ = txt[len-1] == '\n';
fprintf(outputf, "%.s", len, txt);
}
return;
}
indent();
if (whattext)
fprintf(outputf, "%s ", whattext);
else
fprintf(outputf, "metaevent %d ", what);

fputc('"', outputf);
if (txt)
{
while (len-- > 0)
printchar(*txt++);
}
fputc('"', outputf);
if (!whattext)
fprintf(outputf, " end metaevent");
fprintf(outputf, "\n");
}

void MidiPrint::meta(int what, unsigned len, unsigned char* data)
{
unsigned i;

if (info_ < ALL_EVENTS)
return;
indent();
fprintf(outputf, "metaevent %d ", what);
i = 0;
while (len-- > 0)
{
indent_ = 4;
if (printdecimal_)
fprintf(outputf, " %d", *data++);
else
fprintf(outputf, " $%02X", *data++);
if ( ( i % 16) == 15)
{
fprintf(outputf, "\n");
indent();
}
i++;
}
fprintf(outputf, " end metaevent\n");
}

void MidiPrint::meta(int what, FILE* f, long datapos, long datalen)
{
int i;
long oldpos = ftell(f);

if (info_ < ALL_EVENTS)
return;
indent();
fprintf(outputf, "metaevent %d ", what);
i = 0;
fseek(f, datapos, SEEK_SET);
while (datalen-- > 0)
{

```

```

indent_ = 4;
int c = fgetc(f);
if (c < 0)
break;
if (printdecimal_)
fprintf(outputf, " %d", c);
else
fprintf(outputf, " $%02X", c);
if ( (i % 16) == 15)
{
fprintf(outputf, "\n");
indent();
}
i++;
}
fprintf(outputf, " end metaevent\n");
fseek(f, oldpos, SEEK_SET);
}

void MidiPrint::end()
{
endmark_ = 1;
}

void MidiPrint::prefixchannel(unsigned char channel)
{
if (info_ < ALL_EVENTS)
return;
indent();
fprintf(outputf, "prefixchannel %d\n", channel+1);
}

void MidiPrint::prefixport(unsigned char port)
{
if (info_ < ALL_EVENTS)
return;
indent();
fprintf(outputf, "prefixport %d\n", port);
}

void MidiPrint::smpteofs(int mode, int hour, int min, int sec, int frame, int
fracframe)
{
if (info_ < ALL_EVENTS)
return;
indent();
fprintf(outputf, "smpteofs %d %02d:%02d:%02d:%d:%d\n",
mode, hour, min, sec, frame, fracframe);
}

void MidiPrint::tact(int nom, int denom, int unitsperbeat, int notes32perbeat)
{
if (info_ < ONLY_SHORT)
return;
indent();
fprintf(outputf, "tact %d / %d %d %d\n", nom, denom, unitsperbeat, notes32perbeat);
if (!marktact_)
return;
}

void MidiPrint::tempo(unsigned long ticks)
{
if (trackno_ != 1 || getcurunit() != 0)
hastempochanges_ = 1;

if (info_ < ONLY_SHORT)
return;
indent();
if (ticks != 0)

```

```

{
fprintf(outputf, "beats %2.5f", ((float)(60000000.0 / (float)ticks));
fprintf(outputf, " /* %ld microsec/beat */\n", ticks);
return;
}
else
fprintf(outputf, "tempo %ld\n", ticks);
}

void MidiPrint::key(int signature, int isminor)
{
if (info_ < ALL_EVENTS)
return;
indent();
fprintf(outputf, "key \"");
if (signature != 0)
{
fprintf(outputf, "%d%c ", (signature < 0) ? -signature : signature,
(signature < 0) ? 'b' : '#');
}
else
fprintf(outputf, "C");
fprintf(outputf, "%s\"\n", isminor ? "min" : "maj");
}

void MidiPrint::printchannel(int channel)
{
if (info_ < ALL_EVENTS)
return;
if (printchannel_ || getchannel() != channel)
{
indent();
if (channel >= 10 && !printdecimal_)
fprintf(outputf, "[%X]", channel+1);
else if (channel >= 0)
fprintf(outputf, "[%d]", channel+1);
setchannel(channel);
}
}

void MidiPrint::printcurpos(unsigned long units)
{
if (info_ < ALL_EVENTS)
return;
if (printunits_)
fprintf(outputf, " /* U%ld */ ", getcurunit()+units);

// cant print milliseconds in other than first track if tempo changes
// because tempo information is only available in first track
// current tempo microsecperbeat_ initialized to last tempo used in previous track
// so if tempo does not change it is the correct one for the other tracks too
if (printms_ && (trackno_ == 1 || !hastempochanges_))
{
unsigned long ms, msrest;
calctime(units, ms, msrest);

fprintf(outputf, " /* %ldms */ ", ms);
}
}

void MidiPrint::time(unsigned long ticks)
{
unsigned long n;
unsigned long u = curunit_;
unsigned long pause = ticks;

// fprintf(stderr, "%lu:%02d\r", getcurmillisec()/ 60000L, int((getcurmillisec() /
1000) % 60));
indent_ = 2;
if (ticks == 0)

```

```

printcurpos(0);
while (ticks > 0)
{
    unsigned long nextu = tactinfo_ ? tactinfo_>nextMeasure(u) : 0;
    if (!marktact_ || printdecimal_ || !tactinfo_)
        n = ticks;
    else
    {
        n = nextu - u;
        if (n > ticks)
            n = ticks;
    }
    if (info_ < ALL_EVENTS)
        return;
    if (n > 0)
    {
        indent();
        u += n;
        if ((n % unitsperbeat_) == 0 && !printdecimal_)
            fprintf(outputf, "%ld/4", n / unitsperbeat_);
        else if (unitsperbeat_ % 2 == 0 && (n % (unitsperbeat_ / 2)) == 0 && !printdecimal_)
            fprintf(outputf, "%ld/8", n / (unitsperbeat_ / 2));
        else if (unitsperbeat_ % 4 == 0 && (n % (unitsperbeat_ / 4)) == 0 && !printdecimal_)
            fprintf(outputf, "%ld/16", n / (unitsperbeat_ / 4));
        else
            fprintf(outputf, "%ld", n);
        if (!marktact_ || !tactinfo_ || nextu != u)
            fputc(';', outputf);
        else
        {
            fprintf(outputf, ".");
            printcurpos(u-curunit_);
            fprintf(outputf, " // tact %lu\n", (long)tactinfo_>MeasureIndex(u));
            indent_ = 2;
        }
        ticks -= n;
    }
    if (indent_ == 0)
        printcurpos(n);
}

void MidiPrint::program(int channel, int prg)
{
    if (info_ < ONLY_SHORT)
        return;
    indent();
    printchannel(channel);
    if (printdecimal_)
        fprintf(outputf, "program %d\n", prg);
    else
        fprintf(outputf, "program %s\n", progname(prg, channel));
}

void MidiPrint::control(int channel, int what, int value)
{
    if (info_ < ALL_EVENTS)
        return;
    indent();
    printchannel(channel);
    if (printdecimal_)
        fprintf(outputf, "control %d %d", what, value);
    else
        fprintf(outputf, "control $%02X %d", what, value);
    switch(what)
    {
        case 74: fprintf(outputf, " // XG brightness"); break;
        case 84: fprintf(outputf, " // XG portamento"); break;
        case 94: fprintf(outputf, " // XG effect4"); break;
    }
}

```

```

}
fprintf(outputf, "\n");
}

void MidiPrint::balance(int channel, int val)
{
if (info_ < ALL_EVENTS)
return;
indent();
printchannel(channel);
fprintf(outputf, "balance ");
if (val == 0 && !printdecimal_)
fprintf(outputf, " left\n");
else if (val == 127 && !printdecimal_)
fprintf(outputf, " right\n");
else
fprintf(outputf, " %d\n", val);
}

void MidiPrint::highbank(int channel, int val)
{
if (info_ < ONLY_INFO)
return;
indent();
printchannel(channel);
if (printdecimal_)
fprintf(outputf, "hbank %d\n", val);
else
fprintf(outputf, "hbank $%02X\n", val);
}

void MidiPrint::wheel(int channel, int val)
{
if (info_ < ALL_EVENTS)
return;
indent();
printchannel(channel);
fprintf(outputf, "wheel %d\n", val);
}

void MidiPrint::breath(int channel, int val)
{
if (info_ < ALL_EVENTS)
return;
indent();
printchannel(channel);
fprintf(outputf, "breath %d\n", val);
}

void MidiPrint::foot(int channel, int val)
{
if (info_ < ALL_EVENTS)
return;
indent();
printchannel(channel);
fprintf(outputf, "foot %d\n", val);
}

void MidiPrint::portamentotime(int channel, int val)
{
if (info_ < ALL_EVENTS)
return;
indent();
printchannel(channel);
fprintf(outputf, "portamentotime %d\n", val);
}

void MidiPrint::data(int channel, int val)
{
if (info_ < ALL_EVENTS)

```

```

return;
indent();
printchannel(channel);
fprintf(outputf, "data %d\n", val);
}

void MidiPrint::volume(int channel, int val)
{
if (info_ < ONLY_INFO)
return;
indent();
printchannel(channel);
fprintf(outputf, "volume %d\n", val);
}

void MidiPrint::expression(int channel, int val)
{
if (info_ < ALL_EVENTS)
return;
indent();
printchannel(channel);
fprintf(outputf, "expression %d\n", val);
}

void MidiPrint::lowbank(int channel, int val)
{
if (info_ < ONLY_INFO)
return;
indent();
printchannel(channel);
if (printdecimal_)
fprintf(outputf, "lbank %d\n", val);
else
fprintf(outputf, "lbank %02X\n", val);
}

void MidiPrint::hold(int channel, int val)
{
if (info_ < ALL_EVENTS)
return;
indent();
printchannel(channel);
fprintf(outputf, "hold ");
if (val == 0 && !printdecimal_)
fprintf(outputf, " off\n");
else if (val == 0x7f && !printdecimal_)
fprintf(outputf, " on\n");
else
fprintf(outputf, " %d\n", val);
}

void MidiPrint::reverb(int channel, int val)
{
if (info_ < ALL_EVENTS)
return;
indent();
printchannel(channel);
fprintf(outputf, "reverb %d\n", val);
}

void MidiPrint::chorus(int channel, int val)
{
if (info_ < ALL_EVENTS)
return;
indent();
printchannel(channel);
fprintf(outputf, "chorus %d\n", val);
}

void MidiPrint::datainc(int channel, int val)

```

```

{
    if (info_ < ALL_EVENTS)
        return;
    indent();
    printchannel(channel);
    fprintf(outputf, "datainc %d\n", val);
}

void MidiPrint::datadec(int channel, int val)
{
    if (info_ < ALL_EVENTS)
        return;
    indent();
    printchannel(channel);
    fprintf(outputf, "datadec %d\n", val);
}

void MidiPrint::lowrpn(int channel, int val)
{
    if (info_ < ALL_EVENTS)
        return;
    indent();
    printchannel(channel);
    fprintf(outputf, "lowrpn %d\n", val);
}

void MidiPrint::highrpn(int channel, int val)
{
    if (info_ < ALL_EVENTS)
        return;
    indent();
    printchannel(channel);
    fprintf(outputf, "highrpn %d\n", val);
}

void MidiPrint::pitchbendrange(int channel, int val)
{
    if (info_ < ALL_EVENTS)
        return;
    indent();
    printchannel(channel);
    fprintf(outputf, "pitchbendrange %d\n", val);
}

void MidiPrint::noteon(int channel, int note, int vel)
{
    if (info_ < ALL_EVENTS)
        return;
    indent();
    printchannel(channel);
    if (printdecimal_)
        fprintf(outputf, "+%d %d;\n", note, vel);
    else
        fprintf(outputf, "+%s %02X;\n", notename(note), vel);
}

void MidiPrint::noteoff(int channel, int note, int vel)
{
    if (info_ < ALL_EVENTS)
        return;
    indent();
    printchannel(channel);
    if (printdecimal_)
        fprintf(outputf, "-%d %d;\n", note, vel);
    else
        fprintf(outputf, "-%s %02X;\n", notename(note), vel);
}

void MidiPrint::pitchbend(int channel, int val)
{

```

```

if (info_ < ALL_EVENTS)
return;
indent();
printchannel(channel);
fprintf(outputf, "pitchbend %d\n", val);
}

void MidiPrint::polyaftertouch(int channel, int note, int val)
{
if (info_ < ALL_EVENTS)
return;
indent();
printchannel(channel);
if (printdecimal_)
fprintf(outputf, "polyaftertouch %d %d\n", note, val);
else
fprintf(outputf, "polyaftertouch %s %d\n", notename(note), val);
}

void MidiPrint::aftertouch(int channel, int val)
{
if (info_ < ALL_EVENTS)
return;
indent();
printchannel(channel);
fprintf(outputf, "aftertouch %d\n", val);
}

void MidiPrint::songpos(unsigned pos)
{
if (info_ < ONLY_SHORT)
return;
indent();
fprintf(outputf, "songpos %d\n", pos);
}

void MidiPrint::songselect(unsigned char song)
{
if (info_ < ONLY_SHORT)
return;
indent();
fprintf(outputf, "songselect %d\n", song);
}

void MidiPrint::tunerequest()
{
if (info_ < ALL_EVENTS)
return;
indent();
fprintf(outputf, "tunerequest\n");
}

void MidiPrint::timingclock()
{
if (info_ < ALL_EVENTS)
return;
indent();
fprintf(outputf, "timingclock\n");
}

void MidiPrint::start()
{
if (info_ < ONLY_SHORT)
return;
indent();
fprintf(outputf, "start\n");
}

void MidiPrint::cont()
{

```

```

if (info_ < ONLY_SHORT)
return;
indent();
fprintf(outputf, "continue\n");
}

void MidiPrint::stop()
{
if (info_ < ONLY_SHORT)
return;
indent();
fprintf(outputf, "stop\n");
}

void MidiPrint::activesense()
{
if (info_ < ALL_EVENTS)
return;
indent();
fprintf(outputf, "activesensing\n");
}

void MidiPrint::sysex(unsigned syslen, unsigned char* sysdata)
{
unsigned i = 0;

if (info_ < ALL_EVENTS)
return;
indent();
syslen--; // last byte is end of sysex (F7)
fprintf(outputf, "sysevent ");
while (syslen-- > 0)
{
indent_ = 4;
if (printdecimal_)
fprintf(outputf, "%d ", *sysdata++);
else
fprintf(outputf, "$%02X ", *sysdata++);
if ( ( i % 16) == 15)
{
fprintf(outputf, "\n");
indent();
}
i++;
}
fprintf(outputf, "end sysevent\n");
}

void MidiPrint::sysex(FILE* f, long sysdatapos, long sysdatalen)
{
int i = 0;
long oldpos = ftell(f);

if (info_ < ALL_EVENTS)
return;
indent();
sysdatalen--; // last byte is end of sysex (F7)
fprintf(outputf, "sysevent ");
fseek(f, sysdatapos, SEEK_SET);
while (sysdatalen-- > 0)
{
indent_ = 4;
if (printdecimal_)
fprintf(outputf, "$%d ", fgetc(f));
else
fprintf(outputf, "$%02X ", fgetc(f));
if ( ( i % 16) == 15)
{
fprintf(outputf, "\n");
indent();
}
}
}

```

```

}
i++;
}
fprintf(outputf, "end sysevent\n");
fseek(f, oldpos, SEEK_SET);
}

void MidiPrint::gmreset()
{
if (info_ < ALL_EVENTS)
return;
indent();
fprintf(outputf, "GMReset\n");
}

void MidiPrint::gsreset()
{
if (info_ < ALL_EVENTS)
return;
indent();
fprintf(outputf, "GSReset\n");
}

void MidiPrint::xgreset()
{
if (info_ < ALL_EVENTS)
return;
indent();
fprintf(outputf, "XGReset\n");
}

void MidiPrint::gsexit()
{
if (info_ < ALL_EVENTS)
return;
indent();
fprintf(outputf, "GSExit\n");
}

void MidiPrint::hex(FILE* f, long pos, long len, FILE* outputf)
{
fseek(f, pos, SEEK_SET);
unsigned char c[16];
while (len > 0)
{
int n = 16;

if (len < n)
n = (int)len;
if (fread(c, n, 1, f) != 1)
break;
fprintf(outputf, "//");
for (int i = 0; i < n; i++)
fprintf(outputf, " %02X", c[i]);
fprintf(outputf, "\n");
len -= n;
}
}

int MidiPrint::runxftrack(int trackno)
{
unsigned long trackend;
trackpos_ = curpos_;

if (exit_)
return 0;

skiptrack_ = 0;
curunit_ = 0;
curms_ = currest_ = 0;

```

```

lastcode_ = -1;
pos_ = curpos_;
curchannel_ = scanchannel(tracklen_);
pos_ = trackpos_ = curpos_;
trackend = trackpos_ + tracklen_;
lastcode_ = -1;
if ((options_ & OPTION_NOEVENTS) == 0)
while (!skiptrack_ && curpos_ < trackpos_ + tracklen_)
{
int newpercent = (int)((curpos_ * 100) / midilength());
if (newpercent != percent_)
percent(percent_ = newpercent);

unsigned long delta = getdelta();
if ( delta >= NOTREALISTIC_PAUSE )
{
unrealisticpause(delta);
skiptrack_ = 1;
continue;
}
time(delta);
if (exit_)
return 0;
curunit_ += delta;
// warning: MidiRead time calculation is only valid for format 0 files
// use MidiSerial to calculate time for any midi format
calctime(delta, curms_, currest_);

if (runevent(trackend) <= 0)
{
skiptrack_ = 0;
return 0;
}
if (exit_)
return 0;
}
skiptrack_ = 0;
seek(trackend);
pos_ = curpos_;
return 1;
}

void MidiPrint::printxf()
{
fprintf(outputf, "*/\n");
fprintf(outputf, "XFIH\n");
tracklen_ = getlong();
runxftrack(0);
fprintf(outputf, "end XFIH\n");

while (curpos_ + 4 < midilength())
{
long pos = curpos_;
if (getlong() != XFKM)
{
seek(pos);
break;
}
fprintf(outputf, "XFKM\n");
tracklen_ = getlong();
runxftrack(0);
fprintf(outputf, "end XFKM\n");
}
fprintf(outputf, "*/\n");
}

void MidiPrint::endmidi()
{
if (info_ < ALL_EVENTS && info_ != ONLY_ERRORS)
return;

```

```

if (getpos() < midilength())
{
long len = midilength() - getpos();

if (len >= 16)
{
long endpos = getpos();
if (getlong() == XFIF)
{
printf();
endpos = getpos();
}
seek(endpos);
if (endpos >= midilength())
return;
}
fprintf(outputf, " // Warning: %ld bytes garbage at end of file\n", len);
if (info_ != ONLY_ERRORS)
{
hex(getf(), getpos(), len, outputf);
}
}

void MidiPrint::percent(int perc)
{
if (info_ >= ONLY_CHUNK)
fprintf(stderr, "%-3d%\r", perc);
}

void MidiPrint::error(const char* msg)
{
fprintf(outputf, "// error: %s\n", msg);
errorcnt_++;
}

int usage()
{
printf("Usage: midi2txt [-error][-chunk][-short][-info][-lyric][-tact][-units][-ms][-
dec][-channel] file.mid [file.txt]\n");
printf("options:\n");
printf("-error\tchecks file only. Produces only error report\n");
printf("-chunk\tprints structure of midi file (header, tracks)\n");
printf("-short\tprints short info about text, tracknames, programs\n");
printf("-info\tprints info about text, tracknames, programs, controls\n");
printf("-lyric\tprints lyrics only\n");
printf("-tact\tmarks end of tact bars with .\n");
printf("-units\tprints position of all events in midi units.\n");
printf("-ms\tprints position of all events in milliseconds (1st track only).\n");
printf("-dec\tprint decimal values instead of notes, channels, programs,pauses...\n");
printf("-channel\tprint channel before every event\n");
return 1;
}

#ifdef NOMAIN
int main_midi2txt(int argc, char**argv)
#else
int main(int argc, char**argv)
#endif
{
int info = ALL_EVENTS;
int tact = 0, printunits = 0, printms = 0, ret = 0, printdec = 0, printchannel = 0;
char* midiname = 0;

argc--; argv++;

while (argc > 0 && **argv == '-')
{
if (strncmp(*argv, "-version", 2) == 0)
{

```

```

printf("%s\n", version);
argc--; argv++;
if (argc == 0)
return 0;
continue;
}
if (argc > 0 && strncmp(*argv, "-error", 2) == 0)
{
info = ONLY_ERRORS; // check if errors in midi
argc--; argv++;
continue;
}
if (argc > 0 && strncmp(*argv, "-decimal", 2) == 0)
{
printdec = 1;
argc--; argv++;
continue;
}
if (argc > 0 && strncmp(*argv, "-chunk", 4) == 0)
{
info = ONLY_CHUNK;
argc--; argv++;
continue;
}
if (argc > 0 && strncmp(*argv, "-channel", 4) == 0)
{
printchannel = 1;
argc--; argv++;
continue;
}
if (argc > 0 && strncmp(*argv, "-short", 2) == 0)
{
info = ONLY_SHORT;
argc--; argv++;
continue;
}
if (argc > 0 && strncmp(*argv, "-info", 2) == 0)
{
info = ONLY_INFO;
argc--; argv++;
continue;
}
if (argc > 0 && strncmp(*argv, "-lyric", 2) == 0)
{
info = ONLY_LYRICS;
argc--; argv++;
continue;
}
if (argc > 0 && strncmp(*argv, "-tact", 2) == 0)
{
tact = 1;
argc--; argv++;
continue;
}
if (argc > 0 && strncmp(*argv, "-ms", 2) == 0)
{
printms = 1;
argc--; argv++;
continue;
}
if (argc > 0 && strncmp(*argv, "-units", 2) == 0)
{
printunits = 1;
argc--; argv++;
continue;
}
printf("invalid option %s\n", *argv);
argc--; argv++;
return usage();
}

```

```

if (argc == 0)
return usage();

midiname = *argv++; argc--;
MidiPrint midi(midiname);
if (!midi.getf())
{
perror(midiname);
return 1;
}
if (argc > 0)
{
outputf = fopen(*argv, "w");
if (!outputf)
{
perror(*argv);
return 1;
}
}
else
outputf = stdout;
fprintf(stderr, "%s\n", midiname);
midi.info_ = info;
midi.marktact_ = tact;
midi.printunits_ = printunits;
midi.printms_ = printms;
midi.printdecimal_ = printdec;
midi.printchannel_ = printchannel;
if (info == ONLY_CHUNK)
midi.options_ = OPTION_NOEVENTS;
if (!midi.run())
{
fprintf(outputf, "%s: midi error at 0x%04lx\n", midiname, midi.getpos());
ret = 1;
}
else
ret = 0;
if (outputf && outputf != stdout)
fclose(outputf);
return ret;
}

```

MIDI2EMNML Wrapper Source Code

```
# midi2emnml
#
# Written by:      Eric Mosterd
# Date:           4/1/2001
#
# Usage:          midi2emnml.pl <filename or URL>
#
# Description:    This program was written as a part of my masters thesis on Extensible Music Notation Markup LanguageL. It will
#                 convert MIDI files into EMNML. You can either specify a file name, or a URL where the MIDI file resides, and
#                 this program will output the EMNML version of that file to STDOUT. If you want to capture the contents of the
#                 file, use the redirect command (>) and type a filename afterwards.
#
# Version:        1.0 alpha 1
#
# Notes:          This is the very feature limited first release of the program.
#
# global variables:
#
#   # this array keeps track of the track ID an name (note: element 0 is not used)
#   @tracks;
#
#   # this keeps track of the current velocity based on the track (used for dynamic changes) (note: element 0 is not used)
#   @velocities;
#
#   # this keeps track of the current measure length based on the track (note: element 0 is not used)
#   @measure_lengths;
#
#   # this keeps track of the measure numbers (note: element 0 is not used)
#   @measure_numbers;
#
#   # this keeps track of the order of notes which are flat, to get the sharps, we merely need to reverse the array
#   # this is used to determine which notes are accidentals, since regardless of the scale, MIDI stores whether or not the note
#   # is flat or sharp (note: element 0 is not used)
#   @key_array = ("", "B", "E", "A", "D", "G", "C", "F", "");
```

```

# this keeps track of the key signature
$key_signature = 0;

# this keeps track of the length of a measure as defined in the MIDI header information
$max_measure_length = 0;

# this keeps track of the number of ticks that a quarter note receives
$quarter_ticks = 0;

# this keeps track of the current track number for MIDI file formats of type 1
$current_track = 0;

# the next two variables keep track of both the chord status, and the length of the chord itself; they also keep track
# of chord on and off events
$chord = -1;
$chord_length = 0;
$chord_velocity = 0;
@chord_ons;
@chord_offs;

sub main {

# check to see if a file name was input, and if it was valid
if (($ARGV[0] !~ /\w/) || (! -e $ARGV[0])) {
    print "Error reading file!\n";
    exit 1;
}

else {
    $midi_file = $ARGV[0];
}

# well, the file exists, so use midi2txt to open and parse the file, checking, of course, to make sure midi2txt exists
if (! -e "midi2txt.exe") {
    print "Error accessing midi2txt! Please place it in the same directory in which this program resides!\n";
    exit 1;
}

else {
    # run the program and store the output in temp.txt
    system ("midi2txt.exe $midi_file temp.txt");
}
}

```

```

# get the contents of the file and place them into an array
open (MIDITXT, "temp.txt");
    @events = <MIDITXT>;
close (MIDITXT);

# remove the file
# unlink ("temp.txt");

# also, get rid of the CRs
chop (@events);

# get rid of the leading spaces too
foreach (@events) {
    $_ =~ s/^\s+//;
}

# next, print off the heading tags
print "<!DOCTYPE EMNML PUBLIC \"-//EMNML//DTD/EN\" \"http://www.usd.edu/eric/thesis/emnml.dtd\">\n\n";
print "<EMNML>\n";

# now, process each of the events
for ($index = 0; $index <= $#events; $index++) {

    # if the event contains a semicolon and either a + (note on) or a - (note off), then it is a note
    if (($events[$index] =~ /\;/) && ( ($events[$index] =~ /\+/) || ($events[$index] =~ /\-/) )){

        # first we need to check if the note is a chord by seeing if it is first a note off (-)
        # and if the next note is a note off
        if (($events[$index] =~ /\+/) && ($events[$index + 1] =~ /\+/)) {

            # get all of the sequential note ons
            while ($events[$index] =~ /\+/) {
                push (@chord_ons, $events[$index]);
                $index++;
            }
            # now get all of the sequential note offs
            while ($events[$index] =~ /\-/) {
                push (@chord_offs, $events[$index]);
                $index++;
            }

            # if the two arrays are not equal in length, then one of the chords has a tie and needs
            # to be handled specially

```

```

if ($#chord_ons != $#chord_offs) {

    # open the chord tag
    $chord = 1;

    # set the chord length
    ($chord_length, $junk) = split (/\/;/, $chord_offs[0]);

    # change the chord ons to chord offs and process them
    foreach (@chord_ons) {
        $_ =~ s/\+/\-/g;
        # set the chord dynamic
        ($junk, $chord_velocity) = split (/\/s/, $_);
        $chord_velocity =~ s/\/;//g;
        &process_note ($_);
    }

    # close the chord tag
    $chord = -1;

}

# otherwise, they can be handled as normal chords
else {
    foreach (@chord_ons) {
        &process_note ($_);
    }

    # open the chord tag
    $chord = 1;

    foreach (@chord_offs) {
        &process_note ($_);
    }

    # close the chord tag
    $chord = -1;
}

undef (@chord_ons);
undef (@chord_offs);
# close the chord tag

```

```

        print "\t</CHORD>\n\n";
        $chord_velocity = -1;
        $chord_length = 0;
        # so that the loop continues and we do not skip the last note, set the index back one
        $index--;
    }

    # now handle normal ties
    elsif (($events[$index] =~ /\-/ ) && ($events[$index + 1] =~ /\-/)) {

        # open the chord tag
        $chord = 1;
        while ($events[$index] =~ /\-/ ) {
            &process_note ($events[$index]);
            $index++;
        }

        # close the chord tag
        print "\t</CHORD>\n\n";
        $chord = -1;
        # move the count back one
        $index--;
    }

    else {
        &process_note ($events[$index]);
    }
}

# otherwise, it is some other event and should be handled appropriately
else {
    &process_event ($events[$index]);
}

}

# finally, print the ending tags
print "</EMNML>\n";
}

```

```
#####
```

```

# process_note
#
# this routine needs to handle notes/rests in a variety of formats
#
# if the MIDI file is in 0 format, then the notes/notes could look like any of the following:
#
#     [voice]+note velocity;
#     +note velocity;
#     pause;[voice]+note velocity;
#     pause;+note velocity;
#
# also, if the MIDI file is in 1 format, then it needs to keep track of the track numbers

sub process_note {
    local ($note) = @_;
    local ($a, $b, $c, $pre_length, $temp_pitch, $temp_velocity, $temp_length, $dotted, $accidental, $track, $temp_note,
    $temp_octave, $tie_length, $accidental);

    # split the note by semicolon
    ($a, $b, $c) = split (/\/, $note);

    # if a does not contains a note, then set the initial pause to that value
    if ($a !~ /\$/ ) {
        $pre_length = $a;
        $a = $b;
    }

    ($temp_pitch, $temp_velocity) = split (/\/s/, $a);

    # when the MIDI format is 0, a track information in [] are stored with each note
    # so we need to parse out that infomration
    if ($temp_pitch =~ /\[/g) {
        ($track, $temp_pitch) = split (/\/], $temp_pitch);
        $track =~ s/\/W//g;
    }
    else {
        $track = $current_track;
    }

    # check the current measure length to see if we need to start a new one
    if ($measure_lengths[$track] == -1) {

```

```

        print "\n<MEASURE STAFF=\"", $tracks[$track], "\" NAME=\"", ++$measure_numbers[$track], "\">\n";
        $measure_lengths[$track] = 0;
    }

    # next, check to see if it is a chord, and if it should be turned on or off
    if ($chord == 1) {

        # if the velocity is different than the velocity of the previous note
        # we need to first write a dynamic tag
        if (($chord_velocity != -1) && ($velocities[$track] ne $chord_velocity)) {
            print "\t<DYNAMIC VOLUME=\"", &velocity_lookup($chord_velocity), "\"/>\n";

            $velocities[$track] = $chord_velocity;
        }

        print "\n\t<CHORD>\n";

        if (eval($chord_length) <= 0) {
            $chord_length = $pre_length;
        }
        else {
            $pre_length = 0;
        }
    }

    # if the note is a note off, write the note using the running note length
    # otherwise, the pause before the note on is a rest
    if ($temp_pitch =~ /\-/g) {

        # now parse the note information
        $temp_note = uc (substr ($temp_pitch, 1, 1));

        # in MIDI, everything is treated as a sharp or natural, so we must convert the flats to their equivalent
        # sharps (e.g. Bb -> A#) and then check against the key signature
        if (length ($temp_pitch) == 4) {

            # check to see if the key is flat, if so, convert the sharp to a flat
            if (($key_array[1] eq "B") && ($key_signature > 0)) {
                if ($temp_note eq "G") {
                    $temp_note = "A";
                }
            }
            else {

```

```

        $temp_note = chr (ord ($temp_note) + 1);
    }

    if ( !(grep (/ $temp_note/, $key_array[1..$key_signature])) ) {
        $accidental = " ACCIDENTAL=\"b\"";
    }
}
else {
    if ( !(grep (/ $temp_note/, $key_array[1..$key_signature])) ) {
        $accidental = " ACCIDENTAL=\"#\"";
    }
}

# we take 5 minus the MIDI note number to get the octave
$temp_octave = int (substr ($temp_pitch, 3, 1)) - 4;
}

else {
    # we take 5 minus the MIDI note number to get the octave
    $temp_octave = int (substr ($temp_pitch, 2, 1)) - 4;
}

# now write the note tag
# get the note length information
# also, if it is a chord, we want to display the correct length,
# but only add it once to the current measure length
if (eval($chord_length) > 0) {
    ($temp_length, $dotted) = &length_lookup ($chord_length);
}
else {
    ($temp_length, $dotted) = &length_lookup ($pre_length);
}

# if there is one or more dots, modify dotted as follows
if ($dotted > 0){
    print "\t<NOTE PITCH=\"", $temp_note, "\" OCTAVE=\"", $temp_octave, "\" LENGTH=\"", $temp_length, "\"
DOTTED=\" $dotted\"$accidental/>\n";
}
else {
    print "\t<NOTE PITCH=\"", $temp_note, "\" OCTAVE=\"", $temp_octave, "\" LENGTH=\"", $temp_length,
"\"$accidental/>\n";
}
}

```

```

# finally, reset the note length and add it to the measure length
# also, if it is a chord, we want to display the correct length,
# but only add it once to the current measure length
if ((eval($pre_length) > 0) || ($chord == 1)) {

    $measure_lengths[$track] += eval($temp_length) * $quarter_ticks * 4;

    for ($i = 1; $i <= $dotted; $i++) {
        $measure_lengths[$track] += (eval($temp_length) * $quarter_ticks * 4) / (2 ** $i);
    }

    if ($chord == 1) { $chord = 2; } # used to continue the chord
}

$pre_length = 0;

# also, reset dotted
$dotted = 0;

}

# it contains a note on, so if there is a number before it, then create the rest
elseif ($temp_pitch =~ /\+/) {

    # get the rest length information
    if (eval($chord_length) > 0) {
        ($temp_length, $dotted) = &length_lookup ($chord_length);
    }
    else {
        ($temp_length, $dotted) = &length_lookup ($pre_length);
    }

    # if there is one or more dots, modify dotted as follows
    if (eval($temp_length) > 0) {
        if ($dotted > 0) {
            print "\t<REST LENGTH=\"$temp_length\" DOTTED=\"$dotted\"/>\n";
        }
        else {
            print "\t<REST LENGTH=\"$temp_length\"/>\n";
        }
    }
}

```

```

    }

    # finally, reset the rest length and add it to the measure length
    # also, if it is a chord, we want to display the correct length
    # but only add it once to the current measure length
    if ((eval($pre_length) > 0) || ($chord == 1)) {

        $measure_lengths[$track] += eval($temp_length) * $quarter_ticks * 4;

        for ($i = 2; $i <= $dotted; $i += 2) {
            $measure_lengths[$track] += (eval($temp_length) * $quarter_ticks * 4) / $i;
        }

        if ($chord == 1) { $chord = 2;} # used to continue the chord
    }

    $pre_length = 0;

    # also, reset dotted
    $dotted = 0;
}

# otherwise it is a note on, so check the dynamic
else {
    # if the velocity is different than the velocity of the previous note
    # we need to first write a dynamic tag
    if ($velocities[$track] ne $temp_velocity) {
        print "\t<DYNAMIC VOLUME=\"", &velocity_lookup($temp_velocity), "\"/>\n";

        $velocities[$track] = $temp_velocity;
    }
}

# if the measure is full, end it
# also, make sure not to close the measure before the CHORD is complete
if (($measure_lengths[$track] >= $max_measure_length) && ($chord == -1)) {
    print "</MEASURE>\n";
    $measure_lengths[$track] = -1;
}
}

```

```

#####
# process_event
#
# for every other non-note event, this routine shall process and print the appropriate tags
#
# mainly, this routine handles the information in the MIDI header; however, tempo, key, and
# time changes are also handled by this routine
#
# note:      this routine is quite ugly and should be cleaned up
#
# for readability, we will process the events in the order they appear in the output

sub process_event {
    local ($event) = @_ ;

    # process the number of ticks per quarter note
    QUARTER_TICKS: {
        if ($event =~ /unit \d+/gi) {

            ($junk, $quarter_ticks) = split (/^unit /, $event);
            ($quarter_ticks, $junk) = split (/ \\/, $quarter_ticks);

            last QUARTER_TICKS;
        }
    }

    # process the title
    TRACKNAME: {
        # since MIDI uses trackname both for the title of the score and
        # for each individual staff name, we need to handle that here
        #
        # also, print out the STAFF and CLEF tags for the track
        #
        # note, for this version, we will put all tracks in treble cleff
        # since MIDI does not store clef information

        if ($event =~ /^trackname/gi) {
            ($junk, $title, $junk) = split (/\"/, $event);

            if ( ($current_track > 0) && ($title !~ /untitled/ig) ) {
                $tracks[$current_track] = $title;
                $measure_lengths[$current_track] = -1;
                print "\n<STAFF NAME=\"$title\"/>\n";
            }
        }
    }
}

```

```

        print "<CLEF STAFF=\"$title\" TYPE=\"treble\"/>\n";
    }
    elsif ($current_track > 0) {
        $tracks[$current_track] = "staff $current_track";
        $measure_lengths[$current_track] = -1;
        print "\n<STAFF NAME=\"$staff $current_track\"/>\n";
        print "<CLEF STAFF=\"$staff $current_track\" TYPE=\"treble\"/>\n";

    }

    else {
        print "<TITLE>$title</TITLE>\n";
    }

    last TRACKNAME;
}

# processes any text element as a comment, eventually this needs to be a bit more
# intelligent and sophisticated
COMMENTS: {
    if ($event =~ /^text/gi) {
        ($junk, $comment, $junk) = split (/\"/, $event);
        print "<COMMENT>$comment</COMMENT>\n";

    }
    last COMMENTS;
}

# process the copyright (this too needs to be a bit more sophisticated)
COPYRIGHT: {
    if ($event =~ /^copyright/gi) {
        ($junk, $copyright, $junk) = split (/\"/, $event);
        print "<COPYRIGHT>$copyright</COPYRIGHT>\n";

    }
    last COPYRIGHT;
}

# process the tempo
TEMPO: {
    if ($event =~ /beats /gi) {
        ($tempo, $junk) = split (/\.\/, $event);

```

```

        ($junk, $tempo) = split (/beats /, $tempo);

        # if the note length is defined, print out the TEMPO and TIME
        if ($note =~ /\w/) {
            print "<TEMPO NOTE=\"1/$note\" BEAT=\"$tempo\"/>\n";
            $tempo = 0;
            print "<TIME BEATS=\"$beats\" NOTE=\"1/$note\"/>\n";
        }

        last TEMPO;
    }

}

# process the time signature
TIME: {
    if ($event =~ /tact /gi) {
        ($junk, $time) = split (/tact /, $event);
        ($beats, $junk, $note, $junk) = split (/ /, $time);

        # if tempo is defined, then print out the TEMPO and TIME
        if ($tempo > 0) {
            print "<TEMPO NOTE=\"1/$note\" BEAT=\"$tempo\"/>\n";
            $tempo = 0;
            print "<TIME BEATS=\"$beats\" NOTE=\"1/$note\"/>\n";
        }

        # now, get the number of ticks per measure
        $max_measure_length = $quarter_ticks * $beats;

        last TIME;
    }

}

# process the key signature
KEY: {
    if ($event =~ /key /gi) {
        ($junk, $key_signature) = split (/\"/, $event);

        # check if the key is C Major (i.e. no flats or sharps)
        if ($key_signature =~ /Cmaj/gi) {
            $key_signature = 0;
            print "<KEY PITCH=\"C\" SCALE=\"Major\"/>\n";
        }
    }
}

```

```

else {
    ($key_signature, $junk) = split (/ /, $key_signature);

    $temp_scale = substr ($key_signature, 1, 1);
    $key_signature = substr ($key_signature, 0, 1);

    # if it is a sharp scale
    if ($temp_scale =~ /\#/) {

        # lookup the last sharp
        # see if the key_array has been reversed, if not, do so
        if ($key_array[1] ne "F") {
            @key_array = reverse (@key_array);
        }
        $key = $key_array[$key_signature];

        # if the last sharp is G, then the key is A, otherwise raise a half step
        if ($key eq "G") {
            $key = "A";
        }
        else {
            $key = chr (ord ($key) + 1);
        }

        # figure out if the key note is sharp
        if (grep (/ $key /, $key_array[1..$key_signature])) {
            print "<KEY PITCH=\"$key\" TONE=\"#\" SCALE=\"Major\"/>\n";
        }
        else {
            print "<KEY PITCH=\"$key\" SCALE=\"Major\"/>\n";
        }
    }
    # must be a flat scale
    else {
        # see if the key_array has been reversed, if so, re-reverse it
        if ($key_array[1] ne "B") {
            @key_array = reverse (@key_array);
        }

        if ($key_signature == 1) {
            print "<KEY PITCH=\"F\" SCALE=\"Major\"/>\n";
        }
    }
}

```

```

else {
    $key = $key_array[$key_signature];
    print "<KEY PITCH=\"$key\" TONE=\"b\" SCALE=\"Major\"/>\n";
}
}

last KEY;
}

# process the track information
TRACK: {
    # this is specifically for MIDI format 1 files
    if ($event =~ /^mtrk\/(gi) {
        ($junk, $current_track) = split (\/\/ track /, $event);
        last TRACK;
    }
}

# process the program; this is done to assure that the tracks (or staves) have a name assigned to them
PROGRAM: {
    if ($event =~ /program /gi) {
        # check for MIDI format 0
        if ($event =~ /\[\/gi) {
            ($program, $junk) = split (\/\]/, $event);
            $program =~ s\/W\/g;

            # if the track name is blank, assign a generic name
            if ($tracks[$program] !~ /\w/) {
                $tracks[$program] = "staff $program";
                $measure_lengths[$program] = -1;
                print "\n<STAFF NAME=\"staff $program\"/>\n";
                print "\n<CLEF STAFF=\"staff $program\" TYPE=\"treble\"/>\n";
            }
        }

        # otherwise, it is format 1 or there is only one track
        if ($tracks[$current_track] !~ /\w/) {
            $measure_lengths[$current_track] = -1;
            $tracks[$current_track] = "staff $current track";
        }
    }
}

```

```
        }
        last PROGRAM;
    }
}

#####
# velocity_lookup
#
# this subroutine calculates the dynamic of a note based on its hex velocity
#
# this should also be cleaned up with some more clever PERL code
sub velocity_lookup {
    local ($temp_velocity) = @_ ;
    $temp_velocity =~ s/\$/ /g;
    $temp_velocity = hex ($temp_velocity);

    if      ($temp_velocity <= 20) { $temp_velocity = "ppp"; }
    elsif  ($temp_velocity <= 37) { $temp_velocity = "pp"; }
    elsif  ($temp_velocity <= 52) { $temp_velocity = "p"; }
    elsif  ($temp_velocity <= 67) { $temp_velocity = "mp"; }
    elsif  ($temp_velocity <= 83) { $temp_velocity = "mf"; }
    elsif  ($temp_velocity <= 100) { $temp_velocity = "f"; }
    elsif  ($temp_velocity <= 117) { $temp_velocity = "ff"; }
    else   { $temp_velocity = "fff"; }
}

#####
# length_lookup
#
```

```

# this subroutine calculates the note/rest length
#
# note: this routine only resolves notes/rests upto 1/64
sub length_lookup {

    local ($temp_val) = @_ ;
    local ($note_length = "", $numerator, $denominator, $temp_note);

    # first, check to see if it an actual note (like 1/4), if it is just a number,
    # turn it into a fraction
    if ($temp_val !~ /\//g) {
        $temp_val = "$temp_val/" . $quarter_ticks * 4;
    }

    ($numerator, $denominator) = split(/\//, $temp_val);

    # if the top value (numerator) equals one, just return the value
    if ($numerator == 1) {
        $note_length = $temp_val;
    }

    else {

        $denominator = .5;

        while (($note_length !~ /\w/) && ($denominator <= 64)) {

            # get the difference of note that was passed from the generated length
            $temp_note = eval ($temp_val) - (1 / $denominator);

            # if there is no remainder, then return the generated length
            if ($temp_note == 0) {

                if ($denominator <= 1) {
                    $note_length = 1 / $denominator;
                }
                else {
                    $note_length = "1/$denominator";
                }
            }

            # if there is a remainder, calculate how many dots there are

```

```
        elseif ($temp_note > 0) {
            if ($denominator < 1) {
                $note_length = 1 / $denominator;
            }
            else {
                $note_length = "1/$denominator";
            }

            while (($dotted < 3) && ($temp_note > 0)) {
                $denominator *= 2;
                $dotted++;
                $temp_note -= 1 / $denominator;
            }
        }

        # otherwise the note was too big, so decrement the note (double the denominator)
        else {
            $denominator *= 2;
        }
    }

    return ($note_length, $dotted);
}

&main;
```

Bibliography

- Barnes-Svarney, Patricia. The New York Public Library Science Desk Reference. New York: Macmillan, 1995.
- Blood, Dr. Brian. Music Theory Online. Online. <<http://www.be-blood.demon.co.uk/>>. 2001.
- . Music Dictionary Online. Online. <<http://www.be-blood.demon.co.uk/>>.
- Brabec, Jeffrey and Todd Brabec. Music, Money, Success and the Movies: Part Five. Online. <<http://www.ascap.com/filmtv/movies-part5.html>>. 1996.
- . Music and Money. Online. <<http://www.ascap.com/artcommerce/music4money.html>>. 2000.
- Cakewalk Pro Audio. Vers. 8.0. Computer software. Twelve Tone Systems, Inc., 1998.
- Castan, Gerd. Music Notation Codes: SMDL, NIFF, DARMS, GUIDO, abc, MusiXML. Online. <http://www.s-line.de/homepages/gerd_castan/compmus/notationformats_e.html>. 1998.
- . MusiXML. Online. <http://www.s-line.de/homepages/gerd_castan/compmus/MusiXML_e.html>. 1998.
- Czeiszperger, Michael. MIDI File Specification. Online. <<http://www.ibiblio.org/emusic-l/info-docs-FAQs/MIDI-doc/MIDI-SMF.txt>>. March, 1988.
- Dourish, Paul. Xerox PARC: A History of Innovation. Online. <<http://www.dourish.com/paul/invention.html>>. 2000.
- Eckstein, Robert. XML Pocket Reference. Sebastopol, CA: O'Reilly & Associates, Inc., 1999.
- Finale. Vers. 2001a.r1. Computer software. Coda Music Technology, 2000.
- Glatt, Jeff. MIDI Specification. Online. <<http://www.borg.com/~jglatt/tech/midispec.htm>>. 2000.
- Grigaitis, R.J. eXtensible Score Language. Online. <<http://fn2.freenet.edmonton.ab.ca/~rgrigait/xscore/>>. December, 1998.
- Harder, Paul O. and Greg A. Steinke. Basic Materials in Music Theory – A Programmed Course. 7th ed. Needham Heights, MA: Simon & Schuster, Inc., 1991.

- . Harmonic Materials In Tonal Music – A Programmed Course. 5th ed. Part 1. Boston: Allyn and Bacon, Inc., 1985.
- Harold, Elliot Harold. XML Bible. Foster City, CA: IDG Books Worldwide Inc., 1999.
- Hex Workshop. Vers. 3.0. Computer software. BreakPoint Software, Inc., 1999.
- Hoos, Holger H. Keith A. Hamel et al. The GUIDO Notation Format – A Novel Approach for Adequately Representing Score-Level Music. Online. <<http://www.informatik.tu-darmstadt.de/AFS/GUIDO>>. 1998.
- Just What Is TeX? Online. <<http://www.tug.org/whatis.html>>. The TeX User Group, 1999.
- Listing of MIDI Status Codes. Online. <<http://www.opensound.com/pguide/midi/midi5.html>>. Accessed: October, 2000.
- Messick, Paul. Maximum MIDI. Greenwich, CT: Manning Publications Co., 1998.
- midi2txt. Vers. 1.14. Computer software. Günter Nagel, 1995.
- MusicML an XML Experience. Online. <<http://195.108.47.160/3.0/musicml/>>. The Connection Factory, 1999.
- MusicML DTD. Computer software. <<http://195.108.47.160/3.0/musicml/music.dtd>>. The Connection Factory, 1998.
- MusicML Java Applet Viewer. Computer software. <<http://195.108.47.160/3.0/musicml/>>. The Connection Factory, 1998.
- NMPA Ninth Annual International Survey of Music Publishing Revenues. National Music Publisher's Association Online. <<http://www.nmpa.org/nmpa/survey9/base.html>>. 1998.
- Notation Interchange File Format (NIFF) Information. Online. <<http://www.student.brad.ac.uk/srmounce/niff.html>>. 1997.
- Noteworthy Composer. Vers. 1.70. Computer software. NoteWorthy Software, Inc., 2000.
- The Official ALT.MUSIC.MIDI FAQ. Online. <<http://web.ftci.net/~higginsj/ammfaq.html>>. 2000.
- Penfold, RA. Advanced MIDI User's Guide. 2nd ed. Tonebridge, Kent UK: PC Publishing, 1995.

- RIFF FAQ. Online. <<http://www.riff.org>>. 2000.
- Russo, William. Composing Music – A New Approach. Chicago, IL: The University of Chicago Press, 1983.
- Sonne, Jesper and Kent Lindholm Wann. The History of Computers – Xerox PARC. Online. <<http://sol.brunel.ac.uk/history/histxero.html>>. May, 1995.
- Steyn, Jaques. Music Markup Language (MML). Online. <<http://is.up.ac.za/mml/archive/scope.html>>. 1999.
- . Music Markup Language: Specification. Online. <<http://is.up.ac.za/mml/archive/specs.html>>. 1999.
- . MML: Chorale: Johann Sebastian Bach (example). Online. <<http://is.up.ac.za/mml/archive/chorale.html>>. 1999.
- Wallace, Brian C. MIDI2TXT Explained. Online. <<http://www.geocities.com/CapeCanaveral/Lab/7142/mid.htm>>. 2001.
- Wei, Peter Chiam Yih et al. MNML Syntax Vers. 2.0. Online. <<http://www.oasisopen.org/cover/mnmlv200.html>>. 1999.