

En fallstudie och implementering av parsning och översättning av MIDI-filer

Syfte

Att titta på hur SMF, standard midi formatet, är strukturerat, att tolka och översätta en midifil till en form av XML som programmet Griff kan läsa samt utreda vad som är på gång inom musikbytesformatområdet.

Bakgrund

En bekant som sysslar en del med musik använder på sin PocketPC ett program som heter Griff. Detta är ett sequencerprogram för att skapa och spela upp musik i samma anda som Cubase, fast något förenklat och lättare programmerat för att kunna köras på en relativt begränsad maskin. När man sparar filer i detta program sparas dem som i de allra flesta program i ett eget internt format. Man kan dock exportera och importera egna XML-filer med hjälp av Griff. Däremot kan endast ett fåtal musikprogram till stationära datorer exportera filer till något XML-format. Inte ett enda klarar hittills hantera Griff's XML-format.

Den relativt begränsade skara Griffanvändare som finns har uttryckt en önskan om en funktion för att importera midi till Griff. Två medieteknikstudenter med liknande intresse bestämde sig för att samarbeta och lösa detta dokumentstrukturproblem.

Arbetets gång

Vi började samlar på oss litteratur och internetkällor som avhandlade hur midiformatet är uppbyggt och vad som sker inom musikbranschen beträffande dokumentstrukturer. Vi fick också av min griffanvändande kompis ett antal filer som exporterats från Griff i dess XML-format och deras korresponderande midifiler. Öppnar man en av Griff's exporterade XML-filer får man ganska snabbt en känsla för vad som krävs för att Griff skall kunna importera en XML-fil. Något senare fick vi också tag på en PocketPC för att kunna utföra tester av vad som måste vara med i XML-filen för att den skall vara giltig i Griff's import.

Vi började läsa om midiformatets uppbyggnad om bitströmmarna och hur man skulle tolka dessa och började hårdkoda i java. Vi tillverkade ett program som åtminstone läste all headerinformation. Sedan hittade vi javakod som översatte mididatan till ren text, detta visade sig användbart och vi fick en känsla för vilken typ av information som fanns i midifilen och vad i denna som var intressant för oss i detta projekt.

Vi fann också en php-baserad midi till XML-översättare. En tysk vid namn Valentin Schmidt hade gjort en sådan och varit vänlig nog att dela med sig av koden. Vi bestämde oss helt enkelt för att basera vår midi till GriffXML-översättare på denna kod.

Innehållsförteckning

Syfte	2
Bakgrund.....	2
Arbetets gång.....	2
Att översätta midi till något som går att förstå	6
Att översätta midi till någon form av XML	6
Att översätta midi till XML som Griff förstår	7
<i>Griffs upplägg</i>	7
Anpassningar och begränsningar vid översättning.....	8
<i>Att skapa ett pattern</i>	8
<i>Att skapa en section</i>	8
<i>Att skapa en song</i>	8
<i>Övrigt</i>	8
En närmare titt på vad vår översättare gör	9
<i>Utgångspunkt</i>	9
<i>Översättarens arbetssätt</i>	9
<i>Formatering</i>	10
<i>Tidsformatet som Griff använder sig av</i>	11
Resultat.....	11
Diskussion	12
Referenser:.....	13
Bilagor	

SMF, Standard Midi Format

Här kommer en relativt översiktlig beskrivning över hur standardmidiformatet är uppbyggt. Generellt om midi, se Diskussion nedan och för ingående beskrivning av filformatet midi kan vi rekommendera kapitlet "Understanding midi messages" i boken *Midi: A comprehensive introduction*, av Joseph Rothstein.

Fig.1 En midifil har följande övergripande struktur

Header:

4 byte – header chunk tag
4 byte – chunk total size
2 byte – midi format type (0,1,2)
2 byte – number of tracks (1-65535)
2 byte – time division

Eventlist:

4 byte – track chunk tag
4 byte – track chunk size
x byte – track event data

Huvudet med beteckningen *Header chunk* är den tagg som alltid kommer först i filen. Denna tolkas med fördel som ASCII-tecken och heter då *Mthd*.

Följande fyra byte kommer att tala om hur mycket utrymme hela huvudtaggen tar, hur många byte med huvudinformation man skall förvänta sig innan det kommer en midispårstagg (*Track chunk tag*). Taggen som beskriver storleken på *headern* är alltid 6 byte. Av dessa 6 byte är de två första en indikation på vilken formattyp som midifilen har. Är det en fil av typen noll så är alla tonvärden sparade i samma spår. Är den av typen 1, ligger varje instrument eller stämma lagrade i separata spår. Det finns också ett format av typen 2, men det är inte så vanligt och vi har valt att inte titta närmre på det.

När dessa 6 byte har passerat kommer en midispårstagg som, likt huvudtaggen, består av 4 stycken ASCII-tecken *Mtrk*.

Efter denna tagg följer 4 byte med information om hur många *byte* man kan förvänta sig att midispåret innehåller. Innehåller midifilen ett längre stycke kan denna lista med spårdata bli riktigt lång.

Det är först efter detta som den egentliga notbilden förmedlas. Nu följer en lista med olika midimeddelanden som vi kallar händelser (*events*). Det som är lite trixigt här är att meddelandena bara använder så många bitar de behöver, vilket gör att dessa kommer att vara av olika storlek (*variable-length*).

Fig. 2 Så här ser en händelse i händelseinformationsströmmen typiskt ut

deltatid	händelsetyp	midikanal	parameter 1	parameter 2
variable-length	4 bits	4 bits	1 byte	1 byte

Alla händelser, oavsett typ, har ett tidsvärde. En midiklocka räknar med s.k. ticks. Varje midifil har en upplösning där man anger hur många ticks det går på ett slag. I ovan nämnda huvudtagg finns information om upplösningens storlek i enheten ticks/slag eller, mer ovanligt, ticks/ruta (främst för synkronisering mot bildinformation). Slagen kan tolkas som fjärdelsnoter och tempot (*bpm*) för att översätta till verklig tid anges i händesellistans första spår (*track 0*, för alla format-1-midifiler). Där finns även annan information om stycket, tonart, taktart (3/4, 4/4 osv = hur många fjärdedelsnoter/takt) osv.

Enligt figur 1 är tickvärdet angiven som "deltatid". Det är standard och definieras av ett värde med variabel längd (max 4 bytes), vilket bestämmer när en händelse skall uppträda relativt midispårets senaste händelse. Om värdet inte är noll anger deltavärdet väntetiden för denna händelses start relativt föregående händelses slut.

Händelser som inte är tidsberoende föregås ändå av en deltatid som bör sättas till 0. Dessa bör också komma först i midispårets händelseström. Exempel på denna sortens händelser kan vara spår- och instrumentnamn eller copyrightinformation.

Oftast är man intresserad av absoluttiden, när någonting börjar eller slutar relativt styckets start. Anledningen till användning av deltatid kan dock vara att skaparna av midiformatet ville hålla nere filstorleken, då absoluttid vid långa musikstycken ger stora värden, vilket kräver fler bytes.

Fig. 3 Exempel på olika typer av händelser som kan förekomma

händelsetyp	värde	parameter 1	parameter 2
Note Off	0x8	note number	velocity
Note On	0x9	note number	velocity
Note Aftertouch	0xA	note number	aftertouch value
Controller	0xB	controller number	controller value
Program Change	0xC	program number	not used
Channel Aftertouch	0xD	aftertouch value	not used
Pitch Bend	0xE	pitch value (LSB)	pitch value (MSB)

Figuren ovan visar de sju grundläggande händelsetyperna som är definierade i midiformatet. Det finns åtskilligt mer definierat i midiformatets specifikation bland annat metahändelser och systemhändelser. Nämnas kan också att händelsetypen Controller har 128 olika kontrollhändelser att välja mellan. Eftersom detta är något som inte varit intressant att titta på i vårt projekt, har vi inte grävt ner oss i det.

Att översätta midi till något som går att förstå

Försöker man öppna en midifil i en godtycklig texteditor kommer man ganska snart att upptäcka att midifilen inte innehåller särskilt mycket text. Istället får man försöka översätta de bytestycken som dyker upp enligt de bestämda tolkningsregler som vi berättat om tidigare. Dessa beskriver relativt ingående hur man skall tolka de åtabitarsord man tar emot. När man har översatt den data som finns i midifilens bytestycken kan man få ut någonting på följande form:

Fig. 4 Midifil översatt till text med hjälp av metoder ur klassbiblioteket javax.sound.midi (JDK 1.4.1)

```
-----
File: YourSong.mid
-----
Length: 241515 ticks
Duration: 232225729 microseconds
-----
DivisionType: PPQ
Resolution: 480 ticks per beat
-----
Track 0:
-----
tick 0: Sysex message: F0 7E 7F 09 01 F7
tick 0: SMPTE Offset: 96:0:0.0.0
tick 0: Set Tempo: 130.0 bpm
tick 0: Time Signature: 4/4,
MIDI clocks per metronome tick: 24, 1/32 per 24 MIDI clocks: 8
tick 0: Key Signature: C major
tick 0: End of Track
-----
Track 1:
-----
tick 0: Sequence/Track Name: Piano
tick 0: unknown Meta event: 00
tick 0: MIDI Channel Prefix: 0
tick 0: [B0 07 7F] channel 1: control change 7 value: 127
tick 0: [B0 0A 40] channel 1: control change 10 value: 64
tick 0: [B0 00 38] channel 1: control change 0 value: 56
tick 0: [B0 20 01] channel 1: control change 32 value: 1
tick 0: [C0 00] channel 1: program change 0
tick 0: [90 4B 52] channel 1: note On D#5 velocity: 82
tick 0: [90 46 52] channel 1: note On A#4 velocity: 82
tick 0: [90 43 3E] channel 1: note On G4 velocity: 62
tick 0: [90 27 4A] channel 1: note On D#2 velocity: 74
tick 10: [B0 40 7F] channel 1: control change 64 value: 127
tick 245: [80 46 40] channel 1: note Off A#4 velocity: 64
tick 260: [80 43 40] channel 1: note Off G4 velocity: 64
tick 340: [80 4B 40] channel 1: note Off D#5 velocity: 64
tick 480: [90 43 22] channel 1: note On G4 velocity: 34
tick 685: [80 43 40] channel 1: note Off G4 velocity: 64
tick 720: [90 46 42] channel 1: note On A#4 velocity: 66
tick 850: [80 46 40] channel 1: note Off A#4 velocity: 64
tick 960: [90 33 36] channel 1: note On D#3 velocity: 54
... ..
```

Att översätta midi till någon form av XML

Varför skulle man vilja översätta midi till XML Svaret är egentligen ganska självklart om man tittar på innehållet i en XML-fil. Det har en väldigt enkel och lättläst struktur och XML-standarden är en standard som har väl dokumenterade regler. I vårt fall vill vi gärna ha över vår midifil till någon form av XML för att kunna importera denna genererade XML-fil i Griff.

Att översätta midi till XML som Griff förstår

Griffs XML-format är upplagt på ett sätt, som naturligtvis baseras på dess behov av parameterlagring.

Griffs upplägg

I ett tonmönsterparti ett s.k. *pattern* kan man lägga vilken typ av noter man önskar. Längden på ett *pattern* bestämmer hur många takter detta varar. I en s.k. *section* anger man var i tiden eller flera *patterns* ska placeras.

Nästa steg är att sätta samman sina olika *sections* till en *song*. Man kan på så vis återanvända *patterns* och *sections*, helt enligt denna typ av *loopbaserat* musikskapande. För att tonerna i ett *pattern* skall spelas upp krävs att man lägger till ett instrument i sin mixer.

Fig. 5. Från vänster överst: *pattern* och *section*view
Nedan från vänster: *song*view och *mixer*view



Anpassningar och begränsningar vid översättning

Vad som gör saken lite komplicerad är att midi inte jobbar med samma typ av indelning som Griff gör. I midi har man bara ett eller flera olika midispår. I ett midispår ligger all information om en stämma som en lång rad av händelser.

Att skapa ett pattern

Tiden i en *pattern* är självständig och startar alltid på 0 (takt ett). Längden har vi i översättningen bestämt till sista tonens slut.

Att skapa en section

Den eventuella paus som kan finnas i ett midispår och som ignoreras när *patternlängden* sätts, används i stället när vi ska skapa vår *section*. Finns en inledande paus kommer denna att förvandlas till ett offsetvärde, dvs. startvärde för vårt *pattern*. Alla *patterns* offset+längd sparas varefter längden på vår *section* (som bara är en, se nedan) sätts till största värdet bland dessa. På så vis kan vi få någon form av struktur i vår *section*.

Att skapa en song

Den sista nivå som finns i Griff är det som kallas *song*. Denna kan innehålla flera olika *sections* som i sin tur innehåller flera olika *patterns*. Detta för att man ska kunna strukturera upp en låt i en slags övergripande struktur, exempelvis: Intro, vers, refräng, vers, vers, refräng, refräng, slut.

I midifilen finns ingen motsvarighet till denna indelning, vilket gör att vårt program endast kommer att skapa en *section* som är lika lång som vår song.

Övrigt

Tempo, taktart och tonart är tre andra saker som vi måste ta hänsyn till när vi gör översättaren. Mer om dessa i beskrivningen nedan. Alla andra händelser i *eventlistan* ignoreras av vår översättare för att de inte förstås av Griff. Det omvända gäller för de Griffspecifika taggar som inte kan beskrivas i midiformatet. Däremot är vi tvungna att ha med ett visst antal grifftaggar för att det ska fungera.

En närmare titt på vad vår översättare gör

Utgångspunkt

Under arbetet med en befintlig java till midiparser fann vi en liknande php-driven klass. Någon hade roat sig med att göra en hel klass med metoder för att exportera och importera midi till en rad format. Däribland MIDI XML, som beskrivs i diskussion nedan. Utifrån denna parser kunde vi lägga till funktionen `mid2griffxml` (se bilaga för kod) i klassen `midi.class.php`.

Översättarens arbetsätt

Vår översättare räknar antalet spår som finns i midifilen, är filen av format 0 är det givet att man bara har ett spår, det ingår i definitionen. Är filen av format av 1 räknas antalet spår. Vi plockar också ut värden för *timebase* (*pulses per quarter note*), *beats per bar* och *bpm* (*beats per minute*). Varför just dessa värden? Jo, det är saker som vi kommer att använda när vi skapar vår GriffXML.

Nu kommer vi till den information som utgör själva notbilden, det som finns i de händelser som är uppräddade efter midifilens midispårstagg (Mtrk). För varje midispårs tonanslag plockas notvärde, *velocity* och kanal ut. En ton beskrivs med en *NoteOn*-händelse och en *NoteOff*-händelse. För att kunna få reda på hur lång en hel ton är krävs alltså att vi läser båda händelserna och jämför deras tidsvärden. Differensen mellan de två händelserna kommer att ge oss längden på tonen. Vi antog till en början också att en ton bara kan anslås på nytt efter att den har avslutats. (Något som vi fick ompröva lite senare.) Till saken hör att en ton inte nödvändigtvis avslutas i händelsen efter att den anslagits.

Vi kom att behöva någon typ av datastruktur för att lagra notvärden för *NoteOn*-händelser. Det första som vi testade var en *array* där värdena bara sparades, söktes och raderades sekventiellt med händelselistan. Denna *array* blir dock alldeles för stor och ohanterbar, vilket ledde till php-krasch. Vi fann då en phpklass som skapar en länkad lista och har metoder för att söka, lägga till och radera. Finns den aktuella *NoteOn*-händelsens värde inte i listan läggs det till, annars är det redan inlagt och bör ej dyka upp igen. När dess korresponderande *NoteOff*-händelsens notvärde senare dyker upp kollar man detta mot listans värden och om träff finnes räknar man ut tiden, sparar undan resultatet med relaterade attribut och raderar *NoteOn*-händelsens notvärde ur listan. Därefter har vi klarat av en ton.

Efter ett antal tester upptäckte vi att det inte var fullt så enkelt som vi trodde. Midiformatet är inte bara obsolet, det är också nyckfullt. Toner behöver inte avslutas genom en *NoteOff*-händelse, det går lika bra att avsluta en ton genom att ge ytterligare en *NoteOn*-händelse med samma notvärde men denna gång med *velocity*=0. Detta kräver ytterligare en *if*-sats som ska arbeta på samma sätt som vårt *NoteOff*-fall är upplagt. Får vi en *NoteOn*-händelse med *velocity*=0 kollar vi efter dess notvärde i listan och vid träff sparar vi undan tonens alla attribut (som sedan skall användas i skapandet av GriffXML-koden) samt raderar notvärdet ur listan.

När vi börjat komma in på mer avancerade stycken upptäckte vi ytterligare en detalj som ställde till det för oss. En viss ton avslutas alltid senast vid samma tidpunkt som densamma anslås igen, men i händelselistan kan den nya *NoteOn*-händelsen komma innan den gamla har avslutats. Vi löste detta genom att skapa ytterligare en länkad lista och en kontroll till. Om aktuella notvärdet redan finns i första listan och tonen inte är avslutad stoppas det nya anslaget notvärde in i en ny lista i väntan på att det första skall avslutas. När det första anslaget *NoteOff*-händelse dyker upp letar vi först bland första listans notvärden, i andra hand bland andra listans notvärden. Detta gör att vi kommer att kunna hantera detta problem. Allteftersom tonerna är avklarade sparas de undan för att sedan skrivas ut antingen till en XML-fil eller direkt ut på skärmen.

Formatering

En GriffXML-fil bör inte ha färre olika taggar än följande exempel:

```
<GriffDocument Version="2">
  <Studio>
    <Sequencer Tempo="120" BeatsPerBar="4" PulsesPerQuarterNote="480">
      <PatternList>
        <Pattern Name="Pattern" Offset="001:01:001"
          Length="004:00:000" Colour="#6B8087">
          <EventList>
            <Note Offset="001:01:480" Pitch="60"
              Velocity="64" Length="000:00:240"/>
          </EventList>
        </Pattern>
        <Pattern Offset="001:01:001" Length="004:00:000"
          Colour="#B3BD59">
          <EventList>
            <Note Offset="001:01:001" Pitch="60"
              Velocity="64" Length="000:00:240"/>
          </EventList>
        </Pattern>
      </PatternList>
      <SectionList>
        <Section Offset="001:01:001" Length="004:00:000"
          Colour="#407A38">
          <Pattern Name="Pattern2" Offset="001:01:001">
            <Target Name="instrument 2" Class="Instrument"/>
          </Pattern>
          <Pattern Name="Pattern" Offset="001:01:001">
            <Target Name="instrument 1" Class="Instrument"/>
          </Pattern>
        </Section>
      </SectionList>
      <Song Name="" Offset="001:01:001" Length="016:00:000">
        <Section Name="Section" Offset="001:01:001"/>
      </Song>
    </Sequencer>
    <InstrumentList>
      <Instrument Name="instrument 1" Engine="mSyn"
        SampleRate="44100" Channels="1">
      </Instrument>
      <Instrument Name="instrument 2" Engine="mSyn"
        SampleRate="44100" Channels="1">
      </Instrument>
    </InstrumentList>
  </Studio>
  <ViewList />
</GriffDocument>
```

Taggarna *GriffDocument*, *Studio* och *viewlist* definierar XML-dokumentets kropp. En sequencertagg bestämmer tempo, *beats per bar* och *pulses per quarter note* som skall råda igenom de olika vyer vi hierarkiskt har under *song*. Vi definierar vilken offset varje *pattern* skall ha samt hur långt det ska vara. (Närmre beskrivning av vilket format Griff önskar få offsettiden på beskrives i nästa avsnitt.) Inom *pattern*taggarna finns händelsetaggar.

I vårt fall översätter vi egentligen bara rena notrelaterade händelser, inga instrumentbyten, glissando eller vibraton. Detta är information som midifilerna kan innehålla men som vi valt bort till förmån för viktigare detaljer. *Velocity* reglerar hur hårt anslag man gör, 127 är max, *velocity* 0 betyder att tonen upphör. När vårt/våra *pattern* är slut följer en nivå upp i hierarkin *sectionlist*, som innehåller de olika *sections* vi har. I varje *section* bör definieras vilka *patterns* som skall finnas i denna *section* samt vilken offset de skall ha i förhållande till starten på vår *section*. Slutligen följer *songtaggen* vari man talar om vilken/vilka *sections* denna skall innehålla. För att Griff skall lägga till instrument i mixervyn i programmet, så att respektive spår får ett instrument som det kan spelas upp med, krävs en instrumentlista.

Tidsformatet som Griff använder sig av

Griff önskar få alla offset- och längdvärden på formatet xxx:xx:xxx.

Första taltrippet anger hela antalet takter, följande talpar motsvarar hela antalet fjärdedelsnoter och sista tre värdena anger vilket tick man är på. Denna uträkning förlade vi till funktioner i den länkade listans phpklassfil.

Resultat

Vår översättare fungerar ypperligt. Vid test matades ett otal olika ”typer” av midifiler in. Det viktigaste för griffanvändarna var att toninnehållet skulle bibehållas, vilket efterlevs.

Tillvägagångssättet är självklart ämne för en diskussion. Vi kunde med stor sannolikhet ha byggt vidare i javamiljö med samma resultat. Koden och strukturen hade kanske blivit mer överskådlig, dels beroende på att så mycket redan fanns färdigt och dels för det fulla stödet för objektorienterad programmering.

Däremot såg vi fördelar i ett php-drivet system. En webbaserad bara lösning ger vår ”uppdragsgivare” och andra griffanvändare lätt tillgång till en översättare. På gott och ont är php också väldigt förlåtande i utvecklingsammanhang. Samtidigt kan felsökning vara ganska svår.

Diskussion

Behovet av ett språk för att styra elektroniska instrument uppstod när instrumenttillverkare insåg att elektroniska instrument inte längre endast behövde utgöra en del av en ensemble, de kunde utgöra själva grunden i den. Med detta kom samtidigt önskingar från musiker att kunna kontrollera mer än två instrument samtidigt, något som är svårt att klara av som ensam musiker. Ett antal instrumenttillverkare gick 1981 samman för att utreda möjligheterna till ett sådant gränssnitt. 1983 föddes Midi, en hel arkitektur för att arbeta med digitala instrument tillsammans med datorer. I takt med att man ville spara sina alster och redigera i efterhand krävdes att den momentana bitströmmen blev till en fil.

Midi blev således en branschstandard och är så än i dag. Väldokumenterat, kompakt och välanvänt, men något svårbegripligt och inte särskilt intuitivt. Redan tidigt sneglade notsättarbranschen på olika lösningar för lämpliga datarepresentationer av noter. Midi visar sig då innehålla en del begränsningar och de flesta program jobbar med interna och slutna format. Kvarstår gör midi som ett utbytesformat, mellan program och plattformar. Det fungerar ganska bra. Ett problem vi stötte på var de många olika tolkningarna och användningarna av just midiformatet. Filer kan genereras mer eller mindre noggrant, enligt standarden.

På senare år har man dock utrett möjligheterna till ett ännu mer standardiserat och komplett språk för att beskriva musik. Liksom för midi handlar det i första hand om syntetiserad musik. Flera projekt med namn som: SMDL - Standard Music Description Language som bygger på SGML, Music Encoding Initiative (MEI), eXtensible Score Language (XScore) m.fl.¹

Gemensamt för många är någon form av märkordspråkssyntax. Där har på senare år XML flitigt debatterats och standardiserats. Flera forskargrupper och företag har skapat egna XML-dialekter för att märka upp musik och noter. Några som kommit väldigt långt och redan implementerat plugins till de stora notskrivningsprogrammen är Recordare med formatet MusicXML. Det verkar lämpa sig väldigt bra för just notskrivning. Här kommer vi till dilemman.

Att skapa ett format, läsbart och användbart för alla, med uppfyllda krav för alla verkar svårt. Exempelvis blev midifiler på ett tiotal kilobyte som översätts av vår applikation flera hundratals kB stor som färdig GriffXML. Då tappar man kompaktheten för ett strängt resurssnålhetskrav.

Midi Manufacturers Association har även de tagit fram en XML-dtd för att definiera en XML struktur det s.k. MIDI XML. Användningen och utbredningen av denna är för oss okänd. Men för att förstå midiformatets uppbyggnad kan denna struktur vara till hjälp. (Det stöds av ursprungsklassen till vår översättare.)

Här tror vi att XML har en styrka. Formatet är självförklarande och ypperligt för lagring och kanske även som ett utbytesformat. Men dess framtid och ett eventuellt enande av en standard är oviss.

¹ För komplett lista med beskrivningar se <http://xml.coverpages.org/xmlMusic.html>

Referenser:

Projektet:

<http://www.nemcom.nu/parser>

Om midifformatet:

http://ourworld.compuserve.com/homepages/mark_clay/midi.htm

<http://www.borg.com/~jglatt/tech/midifile.htm>

<http://www.sonicspot.com/guide/midifiles.html>

<http://www.soi.city.ac.uk/~ek735/msc/docs/midifile.html>

<http://www.sm5sxl.net/~mats/text/programming/java/JavaMidi/JavaMidi.htm>

Om musicXML och dylikt:

<http://www.idealliance.org/papers/xml2001papers/tm/WEB/03-04-05/03-04-05.htm#N40>

<http://www.cc.jyu.fi/~viitaila/MusicXMLPlayer/>

<http://www.recordare.com/xml.html>

http://www.midi.org/dtlds/midi_xml.shtml

Kodkällor:

<http://dasdeck.de/staff/valentin/midi/>

<http://www.phpclasses.org>

Litteratur:

Midi: A comprehensive Introduction, Joseph Rothstein,
1992, Oxford University Press, Oxford.

Professional PHP programming, Castagnetto, Jesus, et.al.,
1999, Wrox Press Ltd., Birmingham