GRAPHIC TO SYMBOLIC REPRESENTATIONS OF MUSICAL NOTATION

Craig Stuart Sapp
CCARH/Stanford University
`craig@ccrma.stanford.edu`

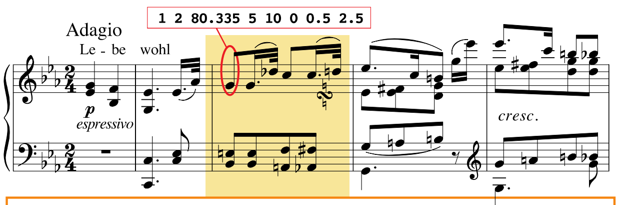
ABSTRACT

This paper discusses the SCORE data format, a graphically oriented music representation developed in the early 1970's, and how such a representation can be converted into sequential descriptions of music notation. The graphical representation system for the SCORE editor is presented along with case studies for parsing and converting the data into other symbolic music formats such as Dox, Humdrum, MusicXML, MuseData, MEI, and MIDI using *scorelib*, an open-source code library for parsing SCORE data. Knowledge and understanding of the SCORE format is also useful for OMR (Optical Music Recognition) projects, as it can be used as an intermediate layer between raw image scans and higher-level digital music representation systems.

1. INTRODUCTION

The SCORE notation editor is the oldest music-typesetting program in continual use. It was created at Stanford University in the early 1970's by Leland Smith and initially was developed on mainframe computers with output to pen plotters that was then photo-reduced for publication. In the 1980's SCORE was ported to IBM PCs running MS-DOS with output to Adobe PostScript, and later ported to Microsoft Windows. Due to the program's long-term stability and excellent graphical output, many critical editions have been created over the years using SCORE, such as the complete works of Boulez, Verdi, Wagner, C.P.E. Bach, Josquin and Dufay.

Throughout its history the SCORE editor has used a simple and compact data format that allows forwards and backwards compatibility between different versions of the SCORE editor. The music representation system is symbolic, but highly graphical in nature. Each notational element is represented by a list of numbers that derive their meanings based on their positions in the list. This format was adapted from the one used in Music V software for computer-generated sound developed by Max Mathews in the late 1950's at Bell Labs. In both cases, the list of numbers serves as a set of parameters describing an object—either to generate a sound in Music V or to place a graphical element on the page in SCORE. This organization of the data is also parsimonious, due largely to memory limitations of computers on which these systems were developed.



```
14 1 77.022  2
1  2 80.335  5 10 0 0.5 2.5
6  2 80.335  7.5 8.88 95.35 11 0 0 11 88.88 95.35 21 0 95.35
1  1 80.335  5 10 0 0.5 1.5
1  1 80.335  8 3 0 0.5
6  1 80.335  6.5 7.5 107.87 11
1  1 88.169  8 0 0 0.5
1  1 88.169  5 10 0 0.5 1.78
1  2 88.169  5 10 0 0.375 3.31 10.95 1
5  2 90.169 15.88 16.5 96.25 1.0133 -1 0 0.55 0 0 0.8
1  2 95.353  9 11 0 0.125 -0.13
1  1 99.301  4 13 0 0.5 3.19
1  1 99.301  9 0 0 0.5
6  2 99.301  9 10 115.39 11 0 0 11 107.87 115.39 21 0 115.39
1  2 99.301  8 10 0 0.5 1
1  1 107.87  4 11.18 0 0.5 3.5
1  1 107.87  9 2 0 0.5
1  2 107.87  8 10 0 0.375 1.53 10
5  2 109.16 17.38 17.88 116.18 1.0633 -1
9  2 111.86  2.88 75 0.65
9  2 113.77 -0.25 8 0.18
9  2 113.89  4.75 8 0.2
1  2 115.39  9 13 0 0.125 1
```

Figure 1. SCORE data for bar 3 of Beethoven *Op. 81a*.

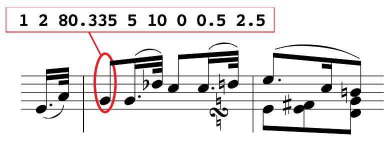
Figure 1 illustrates music typeset in the SCORE editor along with data describing the third measure. Each line of numbers represents a particular graphical element, such as the circled first note of the third measure measure that is represented on the second line in the data excerpt.

The first four numbers on each line have a consistent meaning across all notational items:

P1: Item type (note, rest, clef, barline, etc.).
P2: Item staff number on the page.
P3: Item horizontal position on the page.
P4: Item vertical position on the staff.

Parameter one (P1) indicates the element type—in this example 1=note, 5=slur, 6=beam, and 14=barline. The second number is the staff onto which the element is placed, with P2=1 for the bottom staff and P2=2 for the next higher staff on the page. The third parameter is the horizontal position of the item on the page, typically a number from 0.0 representing the page's left margin, to 200.0 for the right margin. In Figure 1, items are sorted by horizontal position (P3) from left to right on the page; however, SCORE items may occur with any ordering, which typically indicates drawing sequence (z-order) when printing the items. P4 indicates the diatonic vertical position on a staff, with positions 3, 5, 7, 9, and 11 being the lines of a five-lined staff from bottom to top.

These first four numbers on a line give each item an explicit location on the page. The horizontal position is an absolute value dependent on the printing area, while the vertical axis is a hierarchical system based on the staff to which an item belongs: an item's vertical position is an offset from the staff's position on the page, and the staff may have an additional offset from its default position on the page.[1]



P1:	1	= Item type (note).
P2:	2	= Staff no. (second staff).
P3:	80.335	= Horizontal position.
P4:	5	= Vertical position (2nd line from staff bottom).
P5:	10	= Up stem, no accidentals.
P6:	0	= Solid black note.
P7:	0.5	= Duration (eighth-note).
P8:	2.5	= Stem length (2.5 diatonic steps higher than an octave).
P9:	[0]	= No flags on stem; no augmentation dots.

Figure 2. Parameter values and meanings for a note.

The meaning of parameters greater than P4 depends on the type of graphical element being described. Objects with left and right endpoints (beams, slurs, lines) will use P5 as the right vertical position and P6 as the right horizontal position. Figure 2 illustrates some of the higher parameter positions for a note. In this example, P5 describes the stem and accidental display type for the note, with "10" in this case meaning the note has a stem pointing upwards and that there are no accidentals displayed in front of the note. P6 describes the notehead shape, with 0 meaning the default shape of a solid black notehead. P7 indicates the musical duration of the note in terms of quarter notes, such as 0.5 representing an eighth-note. P8 indicates the length of the stem with respect to the default height of an octave. All other unspecified parameters after the last number in the list are implied to be zero. This means either a literal 0, or it may mean to use the default value for that parameter. For this example the implied 0 of P9 indicates that the note has no flags on the stem, nor are there any augmentation dots following the notehead.

Multiple attributes may be packed into a single parameter value, such as P5 and P9 in the above example. This parameter compression was due to memory limitations in computers during the 1970's and 1980's. All values in SCORE data files use 4-byte floating-point numbers. When a parameter can be represented by ten or fewer states, they are typically stored as a decimal digit within these numbers. For example stem directions of notes are given in the 10's digit of P5, while the accidental type is given in the 1's digit. In addition, the 100's digit of P5 indicates whether parentheses are to be placed around the accidental, and the fractional portion of P5 indicates a horizontal offset for the accidental in front of the note. The Windows version of the SCORE editor retains this attribute packing system, primarily for backwards compatibility with the MS-DOS version of the program, since many professional users of SCORE still use the MS-DOS version of the program. This minimal data footprint could also be taken advantage of in low memory situations such as mobile devices or over slow network connections.

SCORE parameters have an interpreted meaning based on the item type and parameter number. With the advent of greater and cheaper memory in computers, the general trend as seen in XML data formats is to provide a key description along with the parameter data. Note that this is a trivial difference between data formats in terms of functionality, but is more convenient for readability and error checking. Below is a hypothetical translation of the SCORE note element discussed in Figure 2 that has been converted into an XML-style element, providing explicit key/value pairs for parameters rather than the fixed-position compressed parameter sequence :

```
<note>
        <staff>2</staff>
        <hpos>80.335</hpos>
        <vpos>5</vpos>
        <stem>up</stem>
        <accidental>none</accidental>
        <shape>solid </shape>
        <duration>0.5</duration>
        <stem-length>2.5</stem-length>
        <flags>0</flags>
        <aug-dots>0</aug-dots>
</note>
```

[1] For a detailed description of the layout axes, see pp. 7–10 of `http://scorelib.sapp.org/doc/coordinates/StaffPositions.pdf`

A translation of these note parameters into MusicXML syntax might look like this:

```
<note default-x="13">
        <pitch>
                        <step>G</step>
                        <octave>4</octave>
        </pitch>
        <duration>4</duration>
        <voice>1</voice>
        <type>eighth</type>
        <stem default-y="25">up</stem>
        <beam number="1">begin</beam>
</note>
```

The primary difference is that SCORE data does not encode explicit pitch information. The pitch “G4” can be inferred from the context of the current clef and key signature as well as any preceding accidentals on G4’s in the measure. Extracting pitch information from SCORE data requires non-trivial but straightforward parsing of the data (excluding slur/tie analysis). A second important structural difference is encoding of beams. In SCORE beams are independent notational items, and linking of notes to beams is inferred within the editor by their spatial proximity and orientation.

MusicXML 3.0 includes a relatively complete layout description, which is more hierarchical than SCORE’s layout description. For example the attribute default-x=“13” of the <note> element describes the distance from the left barline of the measure to the notehead, while in SCORE the P3=80.335 describes the distance from the left margin to the notehead. The stem length is indicated in SCORE and MusicXML in an equivalent fashion, with SCORE setting P8=2.5, which means that the stems should be 2.5 diatonic steps longer than an octave, while MusicXML indicates the same information with the default-y attribute on the <stem> element. Staff assignment in MusicXML is inferred from the part to which the note belongs, while SCORE encodes an explicit staff assignment.

SCORE data is not purely a graphical description of music notation as demonstrated in the above conversion example into MusicXML. It also contains some symbolic information necessary for manipulating graphical items in musically intelligent ways. Within the SCORE editor the musical data can be played, transposed, moved between systems, reformatted, and processed in other musically intelligent ways.

For notes and rests, P7 indicates the duration of the item. This means that there are two horizontal axes present in the data: a spatial axis quantified in P3, and a temporal axis in P7 that describes time in quarter-note units. Figure 3 illustrates these two spatial/time axes present in SCORE data. The SCORE editor can manipulate the data based on either of these descriptions of the music. For example, data entry on each staff can be done independently, in which case the notes on each staff are not aligned vertically. The SCORE program’s LJ command aligns the notes across system staves based on the P7 durations, and this will cause the P3 values of notes to match their rhythmic partners on other staves.

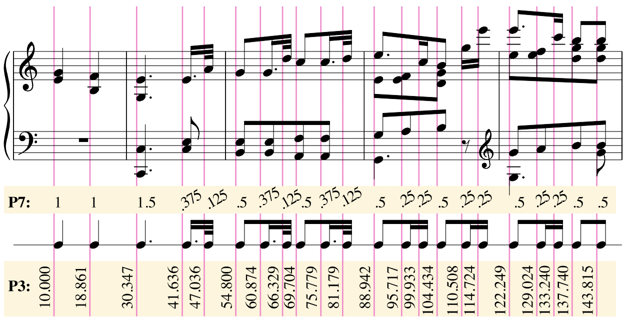


Figure 3. Duration and horizontal position information.

In Figure 3, the vertical lines (in red) are located at the P3 positions of notes in both both staves. In the cases of chords containing intervals of a second, the notes offset to the opposite side of the stem have the same P3 horizontal position of the other notes in the chord, but have a non-zero horizontal offset value (P10). Thus all notes sounding at the same time on a staff must all have the same P3 horizontal position; otherwise, the SCORE editor will misinterpret the notes in a chord as a melodic sequence. Notes on the offbeat of the first beat in measure three have been given an intentional P10 offset from the default spacing, so they do not visually align with the red guide line although their P3 values match the position of the line.

The P7 duration values of notes and rests can be used to calculate the composite rhythm of polyphonic music as illustrated by the rhythm on the single-lined staff below the main musical excerpt in Figure 3. Calculating this rhythmic pattern is necessary for horizontal spatial layout in music notation. In SCORE, horizontal music spacing is calculated on a logarithmic scale, using a spacing factor of approximately the Golden ratio for every power-of-two rhythmic level.

2. SIMILARITY TO OMR PROCESSING

Extracting symbolic musical data in optical music recognition (OMR) can be divided into two basic steps: (1) recognizing graphical elements in a scan, and (2) interpreting their functions and interrelations. In practice there is feedback between these two steps for interpreting the meanings of the elements: if a graphical symbol is ambiguous or incorrect, the context of other symbols around it may clarify the meaning of that item. For musicians this interaction mostly occurs at a subconscious level that can often be difficult to describe within a computer program in order to generate a correct interpretation of the notation. As an example of the inter-dependency of these two steps, the OMR program SharpEye[2] is quite sensitive to visual breaks in note stems. Finding stemless noteheads often leads it to identifying the noteheads as double whole rests which roughly have the same shape as a stemless black notehead. This is clearly a nonsensical interpretation when occurring in meters such as 4/4 or against notes on other staves that do not have the same duration as a double whole note. In such cases where interpretation stage yields such strange results, the identification stage of a graphical element should be reconsidered.

SCORE's data format can be considered a perfect representation of the first stage in OMR processing where all graphical elements have been correctly identified. Converting between a basic OMR representation of graphical elements and SCORE data is relatively easy. For example Christian Fremerey of the University of Bonn/ViFaMusik was able to write a Java program, called *mro2score*, within a few days that converts the SharpEye's graphical representation format into SCORE data.[3]

The *mro2score* program essentially transcodes the identification-stage of musical data from OMR identification and adds minimal markup to convert into SCORE data. In order to convert such symbols into musically meaning syntaxes, more work is necessary. Most OMR programs have built-in editors used to assist the correction of graphic symbol identification as well as their final interpretation. Such editors function in a manner similar to the SCORE editor, which can display graphical elements containing syntactic errors such as missing notes, or incorrect rhythms. Most graphical notation editors such as MuseScore, Sibelius or Finale require syntactically correct data, so they are not as well suited to interactive correction of OMR data.

In order to convert from SCORE data into more symbolic music formats, an open-source parsing library and related programs called *scorelib* has been developed by the author.[4] This library provides automatic analysis of the relations between notational elements in the data, linking music across pages, grouping music into systems and parts, linking notes to slurs and beams, as well as interpreting the pitches of notes. This library is designed to handle the second stage in OMR conversions of scanned music into symbolically manipulable musical data. Conversion from SCORE, and by extension low-level OMR recognition data, into other more symbolic data formats becomes much simpler once these relationships between graphical items have been analyzed using *scorelib*. Currently the *scorelib* codebase can convert SCORE data into MIDI, Humdrum, Dox, MuseData, MusicXML and MEI data formats.[5]

The following sub-sections describe the basic order of analyzing SCORE data in order to extract higher-level musical information needed for conversion into other musical data formats.

2.1 Staves to Systems

SCORE data does not include any explicit grouping of staves into musical systems (a set of staves representing different parts playing simultaneously). So when extracting symbolic information from SCORE data, the first step is to group staves on a page into systems. Errors are unlikely to occur in this grouping process, since staves linked together by barlines are the standard graphical representation for systems. In orchestral scores, parts may temporarily drop out on systems where they do not have notes. In SCORE data, staves are give a part number so that printed parts can be generated from such scores by inserting additional rests for systems on which the part is not present.

2.2 Systems to Movement

Once musical systems have been identified on a page in SCORE (or with any raw OMR graphical elements), the identification of the sequence of systems across multiple pages forming a full movement is necessary in order to interpret items such as slurs and ties. These may be broken graphically by system line breaks. If a set of pages describes a single work, this process is generally as trivial as the staves to systems identification; however, automatic identification of new movements/works will be dependent on the graphical style of the music layout. Typically indenting the first system indicates a new movement/work, but this assumption is not always true. When interpreting SCORE or OMR data, manual intervention may sometimes be needed to handle non-standard or unanticipated cases in movement segmentation.

2.3 Pitch Identification

Pitch identification takes extensive processing of the data. The previous two steps linking staves into systems and systems across pages into movements must first be done before identifying pitch. The data must then be read temporally system by system throughout the movement, keeping track of the current key and resetting the spelling of pitches at each barline for each part/staff. Figure 4 illustrates the result of automatic identification of the

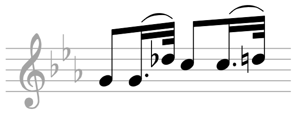
[2] `http://www.visiv.co.uk`

[3] `http://www.ccarh.org/courses/253/lab/mro2score`

[4] `http://scorelib.sapp.org`

[5] See `http://scorelib.sapp.org/program` for a list of available conversion and processing programs.

pitch sequence (g, g, d-flat, c, c, d-natural) for the top staff of music in measure three of Figure 1.



```
6 2 80.335 7.5 8.88 95.35 11 0 0 11 89.17 95.35 21 0 95.35
1 2 80.335 5 10 0 0.5 2.5
@auto@base40Pitch:    185    (g)
5 2 90.169 15.88 16.5 96.25 1.0133 -1 0 0.55 0 0 0.8
1 2 88.169 5 10 0 0.375 3.31 10.95 1
@auto@base40Pitch:    185    (g)
1 2 95.353 9 11 0 0.125 -0.13
@auto@base40Pitch:    207    (dd-)
6 2 99.301 9 10 115.39 11 0 0 11 107.87 115.39 21 0 115.39
1 2 99.301 8 10 0 0.5 1
@auto@base40Pitch:    202    (cc)
1 2 107.866 8 10 0 0.375 1.53 10
@auto@base40Pitch:    202    (cc)
5 2 109.16 17.38 17.88 116.18 1.0633 -1
1 2 115.386 9 13 0 0.125 1
@auto@base40Pitch:    208    (dd)
```

Figure 4 : Automatic pitch labeling of SCORE data.

The *scorelib* library extends the basic SCORE data format to include a list of key/value pairs following the initial line of parameters for a graphical item. In Figure 3, the lines starting "@auto@base40Pitch" are examples of this additional key/value parameter system. In this case the namespace "auto" indicates automatic identification for the pitch of the note. This can be overridden by a manual setting for the pitch with the "@base40Pitch" key.

2.4 Beam grouping

Grouping notes connected to a common beam is a step that can be done either before or after pitch identification, since these two components of notation are independent. In SCORE data this can be done deterministically with little error. Since SCORE data is not organized into measures like many symbolic music data formats, beams crossing barlines are not a difficulty in SCORE, although expressing such barline-crossing beams in translated formats can be difficult.

2.5 Layer Identification

After beaming identification, the most appropriate analysis is to interpret the number of independent monophonic rhythmic streams of notes/rests in each measure. For music with one or two rhythmic streams on a staff, the assignment is relatively straightforward. Three or more rhythmic layers in the music can be difficult to automatically interpret. Graphical music editors typically have four independent layers that can be overlaid on a single staff. SCORE does not have a formalized system for keeping track of rhythmic layers (although there is an informal system in the Windows version of the SCORE editor), so occasionally manual intervention is necessary to assign music to different layers. Figure 5 illustrates the layer interpretation of the music from Figure 1. Since there are no more than two layers on any staff, automatic recognition of the layers is unambiguous. The first layer (as defined in most graphical music editors) is the highest pitched music in the measure with stems pointing upwards if there is a second layer below it. In Figure 5, the second layers in measures 5 and 6 are highlighted in red (or gray in black-and-white prints). The circled rest on the bottom staff of measure 4 presents an interpretational ambiguity: either the bottom layer can be considered to drop out at the rest, or the rest can be interpreted as shared between the two layers on the bottom staff. When extracting orchestral parts in such situations, both parts would share the rest, and the extracted parts would both display the rest.

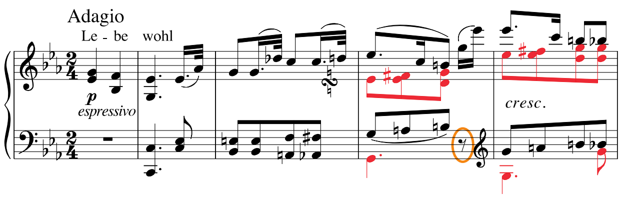


Figure 5. Automatic layer identification, 2nd layer in red.

2.6 Slur/Tie differentiation

After layers have been identified, the final complex step is to distinguish between slurs and ties. For monophonic parts this is straightforward, but in polyphonic parts there are many corner cases to deal with, making 100% correct distinctions difficult to achieve. SCORE has a weak implicit labeling system to differentiate between ties and slurs, but this cannot be depended upon on since the system is primarily intended for graphical offsets of slurs rather than differentiation between slurs and ties. After ties have been identified, pitch identifications need to be reconsidered since tied notes without accidentals will take their accidental from the starting note of a tied group.

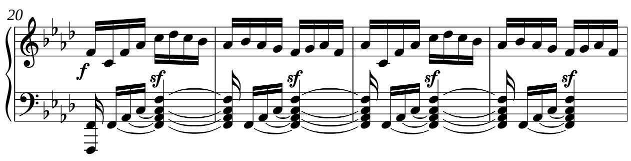


Figure 6. Disjunct ties in Beethoven op. 57, Presto, mm 20-24.

Additionally difficulties arise in both identifying and representing ties that do not connect rhythmically adjacent notes. In particular notated arpeggios such as shown in Figure 6 bypass notating intermediate notes in a slur group, and instead have a single tie connecting the first and last notes in the tie group. Music editors such as MuseScore/Sibelius/Finale cannot handle such cases, and it is also difficult to automatically identify such cases in OMR or SCORE data.

3. DATA CONVERSION FROM SCORE

SCORE uses a two-dimensional description of musical notation, and its data can be serialized into any order since items' positions on the page are independent of each other. Nearly all other music-notation formats impose a sequential structure onto their data, typically chopping up the score into parts, measures, and then layers, which form monophonic chunks that are serialized in different ways. This section presents some of the conversions available with sample programs accompanying the *scorelib* library.

Figure 7 illustrcates three serialization methods within measures that are commonly found in music-notation data formats. In Humdrum data, notes are always serialized by note-attacks times—in other words, all notes from each part/layer played at the same time are found adjacent to each other in the data. This configuration is also true of Standard MIDI Files in type-0 arrangement, where all notes are presented in strict note-attack order. Most other data formats will organize music into horizontal/monophonic sequences by measure rather than by vertical/harmonic slices. MEI chops up a score into a sequence of measures/parts/staves, and finally the staves are segmented into a parallel sequence of monophonic layers. MuseData and MusicXML use the same serialization technique within a measure, but layer segmentations are not as hierarchical as MEI. MusicXML has two ways of serializing measures in a score (*partwise* and *timewise*), but these methods do not affect serialization within a measure.

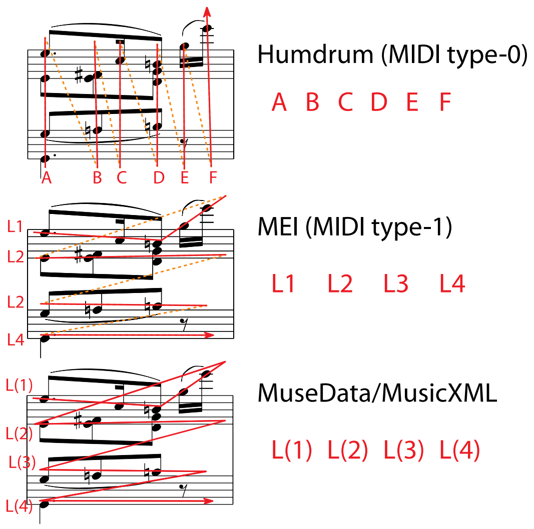


Figure 7. Measure-level serialization schemes in sequential data formats.

In addition to serialization, an important distinction between data formats is the presence or lack of layout information. SCORE data always contains explicit and complete layout information for displaying musical notation, while other data formats have a range of layout description capabilities. The complexity of the notation will determine the necessity of preserving layout information when translating to other file formats. Simple music can automatically be re-typeset without problems; however, complex music is difficult to automatically typeset with a suitable readability quality, and usually human intervention is required to maximize readability in complex notational situations. Many music-notation editing programs focus on ease of manipulation for the musical layout and try to minimize the need for manual control. Likewise, they internally hide the layout information that would be necessary to convert into layout explicit representations such as SCORE data.

Automatic layout will always fail at some point, since the purpose of music notation is to convey performance data to a musician in the most efficient means necessary. Typesetting involves lots of rules and standards, but frequently the rules will need to be broken, or conflicting rules will override each other. Any confusion in the layout decreases the effectiveness of the notation, which a professional typesetter can deal with on a cognitive level much higher than a computer program. Being able to preserve the precise musical layout of SCORE (or OMR) data is very useful, since this can retain human-based layout decisions.

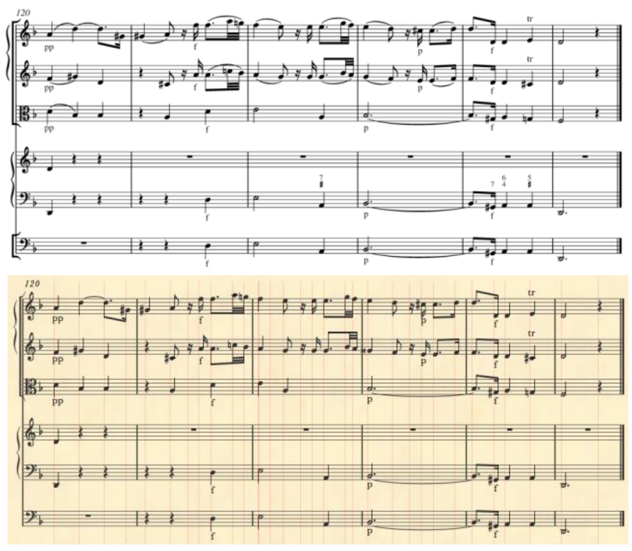


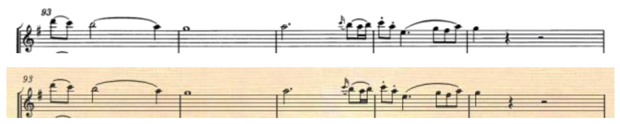
Figure 8 : SCORE PostScript output (top) and SCORE data converted into Dox data in a screen-shot of the Dox editor (bottom).

3.1 SCORE to Dox

Figure 8 shows graphical output from a SCORE PostScript file above a conversion displayed in the Dox music editor written by David Packard. The Dox data format encodes explicit layout information in a header for each system, followed by a listing of symbolic data for each part in the system. For each system measure, a *grid* instruction specifies a spatial distance between times in the composite rhythm for the system. These grid points can be displayed as red vertical lines within the editor as show in the screen capture at the bottom of Figure 8. These gridlines are calculated directly from the horizontal placement (P3) of notes when converting from SCORE data. Within Dox data, the absolute horizontal positions are converted into incremental distances from the previous composite rhythm time in the measure.

Unlike SCORE data, the Dox format separates layout information from symbolic musical elements. Figure 9 shows some sample Dox data illustrating this property. At the start of the data for each system, a header gives layout information. The *bars* directive controls the absolute positions of the measures within the system, and each *grid* directive controls the spacing between composite rhythm positions within each measure. For example "147x13" at the start of the grid for the first measure means that the first beat is 147 spatial units from the start of the measure (relatively wide, to allow for the system clef and key signature to be inserted), then the next position in the composite rhythm sequence is a sixteenth note later, and this is placed 13 units after the notes of on the first beat.

The Dox editor manipulates note spacing by adjusting these grid points, so notes across multiple staves in a system sounding at the same time are always vertically aligned. Vertical positioning of staves as well as the size of staves are also stored in Dox data, so page layout can be preserved when converting from SCORE data.



```
system {pg:10} {tiff:page10.tif} {pix:83,718,1019,1363}
bars {h:144,391,541,693,839,1049}
grid 147x13x13x13x25e14x12x23e15x12x23e14x12
grid 2 7x12x11x11x11e15e17e16e18e16
grid 3 11e15e18e17e17e17e17e17x11
grid 4 10e17e16e18e16e18e16e16
grid 5 10x12x12x12x16x13x13x12x13x12x12x12x14x12x12x12
staff treble k:1 p:1 {v:0 brk bc}
E D6 sa {b:3a} {s:5,6,3,7,-4,-6,1}
E C6 {e:3c} {z}
H B5 sa {s:5,6,2,6,-10,-14,1}
Q A5 {z}
/
W G5
/
H. A5
~X C6 sa {s:1,-7,-3,-5,5,2,1}
E B5 sa {b:3c} {s:5,6,3,7,-7,-7,1} {z}
X A5 sa
X B5 {e:3c} {z}
/
E C6 st {b:3a}
E A5 st {e:3c}
Q. E5 sa {s:5,7,3,6,-15,-11,1}
E G5 sa {b:2c}
E F#5 sa
E A5 {e:2c} {z}
/
Q G5
QR
HR
```

Figure 9 : Scanned notation (top staff) with matching layout of music in Dox editor (bottom staff). System layout is highlighted in gray below the staves, along with symbolic notation in Dox format for the staff.

3.2 SCORE to Humdrum

As a sample of a primarily symbolic data format, this section gives an example conversion result into the Humdrum data format, which is used in computational music analysis applications. This data format typically contains no layout information since the primary focus is on encoding pitch, rhythm and meter for analysis, and not on layout for printing. The Humdrum format is compact and allows the musical content to be read directly from the representation more so than any other symbolic digital representation of musical notation that encode parts serially rather than in the parallel fashion of Humdrum.

The following text lists a conversion from the SCORE data of Figure 1 into Humdrum syntax. Each staff is represented by column of data (*spines*), with staff layers causing splits of the spines into sub-columns. Each line of data represents notes sounding at the same time, so the rows represent the composite rhythm of all parts, which is similar to the rhythm sequence of *grid* directives in Dox.

```
**kern            **kern
*staff2           *staff1
*clefF4           *clefG2
*k[b-e-a-]        *k[b-e-a-]
*M2/4             *M2/4
=1-               =1-
2r                4e-/ 4g/
.                 4B-/ 4f/
=2                =2
4.CC/ 4.C/        4.G/ 4.e-/
8C/ 8E-/          (16.e-/LL
.                 32a-/JJk)
=3                =3
8BB-/ 8En/L       8g/L
8BB-/ 8E/         (16.g/L
.                 32dd-/JJk)
8AAn/ 8F/         8cc/L
8AA-/ 8F#/J       (16.cc/L
.                 32ddn/JJk)
=4                =4
*^                *^
(8G/L     4.GG\   (8.ee-/L    8e-\L
8An/      .       .           8e-\ 8f#\
.         .       16cc/k      .
8Bn/J)    .       8bn/J)      8d\ 8g\J
8r        8r      (16gg\LL    8r
.         .       16eee-\JJ)  .
*clefG2   *clefG2 *           *
*v        *v      *           *
=5        =5      =5
*^        *       *
8g/L      4.G\    8.eee-/L    8ee-\L
8an/      .       .           8ee-\ 8ff#\
.         .       16ccc/Jk    .
8bn/      .       8bbn/L      8dd\ 8gg\
8b-/J     8g\     8bb-/J      8dd\ 8gg\J
*v        *v      *           *
*                 *v          *v
*-                *-
```

Humdrum syntax is a generalized system, so if layout information needs to be preserved, an additional column of for horizontal positions could be added. This would duplicate the functionality of the grid directives in Dox files. Other formats that do not encode layout information would be converted in a similar manner as the conversion process from SCORE into Humdrum. Data formats in this category include MIDI, ABC, LilyPond, and Guido Music Notation.

3.3 SCORE to musicXML

MusicXML is primarily used as a symbolic music format, but has a mostly complete system for specifying layout in notation. In contrast to the Dox format, the layout parameters are interleaved within the data, typically being given as element attributes. Figures 10 and 11 illustrate conversions from SCORE into musicXML for a work by Dufay generated by the *score2musicxml* converter. These two figures highlight the page layout information that can be preserved when translating between SCORE and musicXML. Both figures have the same system break locations, staff scalings and system margins. While musicXML 3.0 has the capability to specify the horizontal layout of notes and measures, this information is currently stripped out of the data when importing into the most recent version of Finale (2014).

3.4 SCORE to MEI

From SCORE's point of view, conversion into musicXML and to MEI are similar, and the *score2mei* converter was initially adapted from the musicXML conversion program. MEI data is more hierarchical than musicXML data, with elements such as beams and chords stored in a tree structure, while musicXML attaches these features to a flat listing of the notes. Figure 12 demonstrates the different encoding methods for a chord in SCORE, MEI and MusicXML. MEI wraps individual notes within a <chord> element, while musicXML marks secondary notes of the chord with a Boolean <chord/> child element.

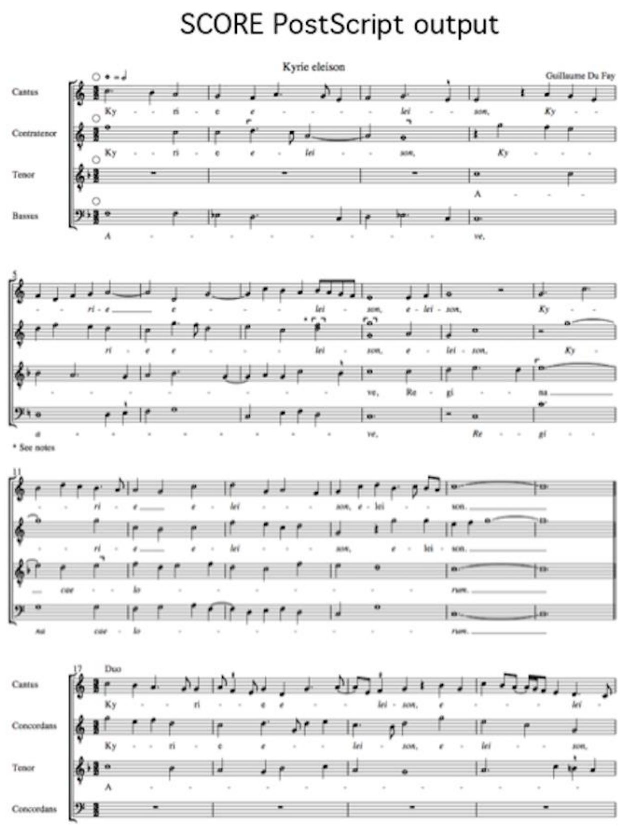


Figure 10 : SCORE PostScript output matching to musicXML translation shown in Figure 11.

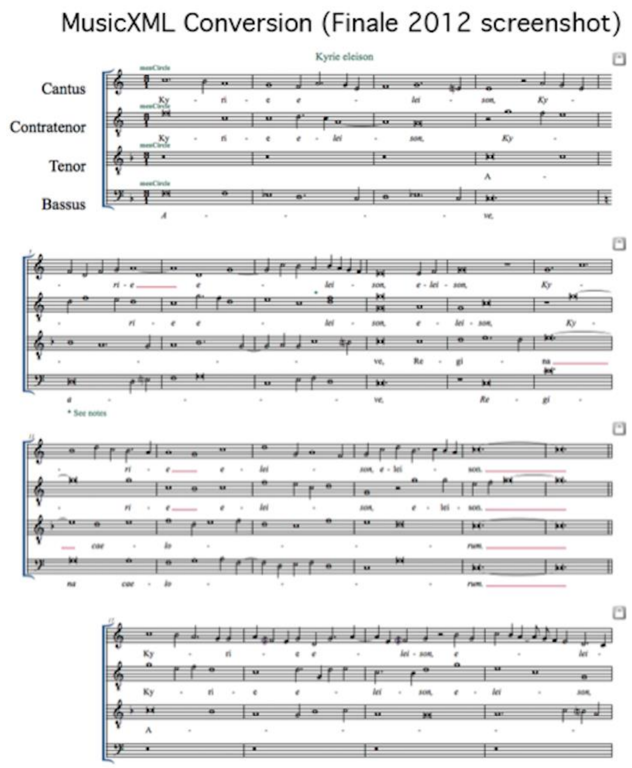
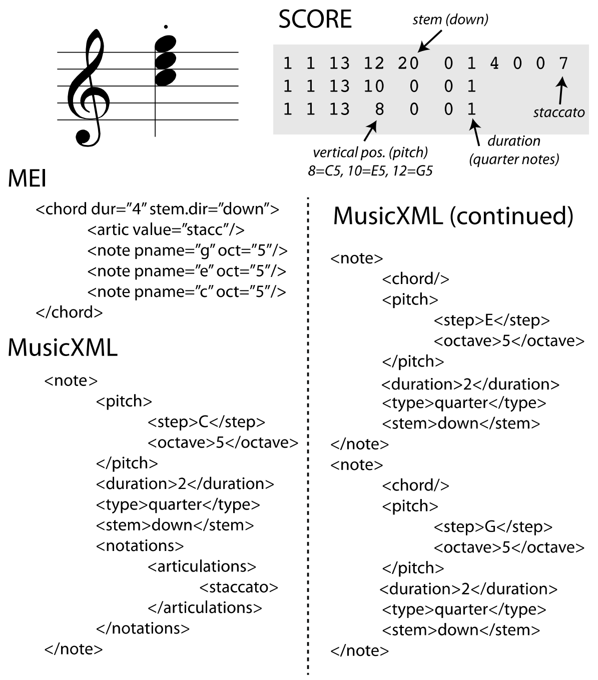


Figure 11 : Screen shot of a musicXML conversion in the Finale music editor. During conversion the rhythmic values of the converted score have been doubled to match the rhythmic values of the original 15th century score.



```
1 1 13 12 20   0 1 4 0 0 7
1 1 13 10  0   0 1
1 1 13  8  0   0 1
```

```
<chord dur="4" stem.dir="down">
        <artic value="stacc"/>
        <note pname="g" oct="5"/>
        <note pname="e" oct="5"/>
        <note pname="c" oct="5"/>
</chord>
```

```
<note>
        <pitch>
                <step>C</step>
                <octave>5</octave>
        </pitch>
        <duration>2</duration>
        <type>quarter</type>
        <stem>down</stem>
        <notations>
                <articulations>
                        <staccato>
                </articulations>
        </notations>
</note>
```

```
<note>
        <chord/>
        <pitch>
                <step>E</step>
                <octave>5</octave>
        </pitch>
        <duration>2</duration>
        <type>quarter</type>
        <stem>down</stem>
</note>
<note>
        <chord/>
        <pitch>
                <step>G</step>
                <octave>5</octave>
        </pitch>
        <duration>2</duration>
        <type>quarter</type>
        <stem>down</stem>
</note>
```

Figure 12. A chord in SCORE format with translations into MEI and MusicXML below.

3.5 SCORE and MuseData

The MuseData printing system uses two data formats: one for symbolic data encoding, and another for explicit layout. Typically music is encoded in the symbolic format that is then compiled into the format with specific layout for interactive editing.[6] MusicXML is structurally based on the symbolic for of MuseData. The compiled layout-specific format is analogous to SCORE data. A useful property of the MuseData printing system is access to both the high-level symbolic representation as well as the low-level graphical representation.

3.6 SCORE and SVG

Due to SCORE data's graphical nature, converting it into images is less complex than generating images from purely symbolic representations (outside of the intended software for a representation, of course). While each graphical element in SCORE can be placed independently at a pre-determined position in an image, software processing symbolic formats must first calculate a graphical layout, and unlike MuseData this layout representation is typically inaccessible as an independent data format. While SCORE software does not have native export to SVG images, minimal processing of its EPS output can produce SVG images.[7] Analytic overlays on the notation image can be aligned to the image using the layout information from the original SCORE data.

Since SCORE data is compact, it can be stored within an XML files. For example the complete SCORE data for the music of Figure 1 can be found in an SVG image of the incipit used on the Wikipedia page for Beethoven's 26th piano sonata.[8] At the bottom of the SVG image's source code, the SCORE data used to create the SVG image is embedded within a processor instruction using this syntax:

```
<?SCORE version="4"
     SCORE data placed here
  ?>
```

Embedding the source code for creating the image allows the data to be used to regenerate an SVG image to fix notational errors or to prepare a new layout, and the embedded data can also be used to generate additional analytic markup.

Further samples of SCORE data can be found in the GitHub repository for *scorelib*.[9] Additional SCORE data samples can be found on IMSLP as attachments to PDFs of music that the author has typeset in SCORE.[10]

[6] The batch-processing version of the MuseData printing system (`http://musedata.ccarh.org`) can be used to generate both PostScript output and the intermediate layout representation called *Music Page Files* (MPG).

[7] Using the open-source converter `https://github.com/-thwe/seps2svg`

[8] `http://en.wikipedia.org/wiki/Piano_Sonata-_No._26_(Beethoven)`

[9] `https://github.com/craigsapp/scorelib/tree/-master/data`

[10] `http://imslp.org/wiki/User:Craig`

4. CONCLUSIONS

SCORE is an important historical data format for computer-based music typesetting. Understanding its graphical representation system is particularly useful for projects in OMR, where interpreted graphical symbols must be organized in a similar process as converting from SCORE into other data formats. In addition, the SCORE representation system should be studied by projects writing automatic music layout of purely symbolic data. SCORE is primarily used by professional typesetters due to its high-quality output and the degree of control afforded to the typesetter. Using the *scorelib* software allows SCORE data to be more easily converted into other musical formats, usually with minimal manual intervention and exactly preserving the original layout.