DATA STRUCTURES IN THE NOTE PROCESSOR.

J. Stephen Dydo
The Music Factory
584 Bergen Street
Brooklyn, NY 11238
(718) 636-2109

Abstract:

The Note Processor is a music printing program, to be released this fall. Although developed to provide engraving-quality music, the design of the program and data structure allows for applications in composition, database storage, education, etc.

It is an "open" system, in that the design of the program permits user access at several different points and in different ways. Input of the data is either through DARMS code, MIDI code, or graphic input; each method produces the same internal representation. This data is then processed by separate modules which, respectively, paginate, break the data down into font elements, perform vertical formatting, and print. The user has graphic access to the data at each of these points; further, the code may be edited, stored, or examined, and new code introduced, all along the way. Thus, external programs may process or generate the data at these points.

The internal representation of the music data is a kind of canonical DARMS. As such, it may be read easily by humans; but its structure allows very quick interpretation by machines as well. It is also compact. The graphic editor allows mouse-driven editing of the code either purely graphically or by editing actual lines.

When the data has been broken down into font elements, it becomes less "user-friendly" and more specific. A second editor allows editing of the font list with the same interface as the canonical DARMS editor. This is extremely useful for specialized notational situations which would be incomprehensible to the DARMS-oriented parts of the program (such as graphs). It may be edited or generated by outside programs, including text editors.

The Program:

The Note Processor was originally designed to serve a single purpose, high-quality music printing. This forthright task, the simplicity of which will be readily apparent to anyone who has been involved with font design, horizontal justification, vertical formatting, beam slanting, slur drawing, printer control, graphic editing, and general input/output in a musical context, still is not an absolute piece of cake until one has concocted an appropriate structure for storing and retrieving music data. Thus, before I sat down to spend a couple of afternoons designing and coding the program which I have been using to print publication-quality music, I actually spent more than that amount of time designing the data structure.
This design had to take into consideration all the little tasks that the program would have to accomplish in a day's work. The first job was to take input data, in a variety of forms, and transform it into a common representation. Input could come from three different sources: DARMS code, MIDI strings, and a screen editor. Then this data had to be 'fleshed out'; that is, in each case, there would be insufficient data to generate printed music. This is the case with MIDI input, where one is presented only with the note-on, note-off timings from the synthesizer; but even in the case of DARMS code, such things as articulation and accidental placement, stem direction and length, andBeam slant are not usually determined by the input. Next, horizontal justification, which is accomplished using the same algorithms and adjustment factors as used by plate engravers, still is a little complicated, since notes must be treated as single, distinct objects at the algorithmic level, yet later on must be examined from the point of view of their constituent components, such as flags, accidentals, dots, and so forth, at the adjustment level. These spatial relations had to be preserved in such a way that the movement of a note and all of its constituents could be accomplished simultaneously or separately, as the need arose. Then, the structure had to allow page formatting such that the placement of the staff did not alter the relations of the characters within it; further, the positions of the various elements in the staff had to be quickly accessible, for purposes of collision-avoidance. At the same time, when other sorts of operations were performed on the data, such as transposition or rhythmic alteration, the strictly musical characteristics of things like note position, accidentals, durations, and ties had to be immediately available. Finally, the user had to have the capability of editing any aspect of the graphic features of the page, whether or not the end result made any sense to the program.

Such a structure, it turned out, was a pretty tall order.

Why DARMS?

I began work on the data structure in 1981, while at the Institute for Sonology, in Utrecht. After having created a fairly complete modal grammar for editing notes and their elements, I met with Leigh Landy of the University of Amsterdam, who had participated in the design of DARMS. He pointed out some of the more elegant solutions offered by DARMS to problems I had been wrestling with, and I agreed to take a serious look at the language. Although I had known of DARMS for some years, I had never had a discussion about it with anyone so intimately involved with it; thus, I had failed to appreciate much of its potential. After a fairly careful perusal, during which it became clear that DARMS could represent most of what I wanted it to, I yielded to my natural laziness and decided to use the ready-made article, abandoning my home-brew system.

The decision to use DARMS as an input medium was made on the basis of its flexibility, specificity and, with the number of defaults available, its speed of entry. However, another aspect of this venerable antique (now over 20 years old) came to light with use. It turns out that, in order to have so many abbreviations and so much flexibility, an input language must have also a deep and complex underlying structure. This structure is what the defaults, abbreviations, and flexibilities are hung on. To be an accurate representation of a page of music, the underlying structure must be, in an unambiguous way, a parallel representation of the underlying structure of the printed page; thus, it must represent the music itself to exactly
the same degree as the page, whether ambiguously or not.

Such an old-fashioned approach to music data—regarding music on the page as the ultimate data source, rather than some of the more modern notions evidenced in the last year's discussions in the MIPS committee files—may need some explanation. As a composer, I am as aware as anyone else of the inadequacies and ambiguities of the notational system which I use to put my ideas on paper. Yet its general acceptance over the last few centuries, by musicians, as the medium of choice for the transmission of musical data, ought to carry a great deal of weight; especially so, if one is considering extending the system into the digital domain, where standardization is of general value. It has seemed to me on this basis alone that the most practical way of storing music digitally, for general uses, is one which has the closest relation to printed music.

In the case of DARMS, the conversion from an input string to a more computer-friendly format is straightforward. In fact, the only serious complications have to do with the complexities of the notational system which DARMS parallels, Every DARMS string assumes certain fundamental features of musical notation which facilitate an elaboration of the string. Different elements, such as noteheads, stems, beams, and accidentals, are all contextually identifiable as part of a single object; yet they can be approached as separate entities if needed. The elaboration of the string into, for example, an array, is to some degree a mechanical working-out of the underlying implications of the string. For example, consider the following tune and one of its possible realizations into DARMS:

```
\G !M2:4 3E, RS 4 RE / R S((2)) 0Q / 1 RQ /
```

By filling in some of the defaults and abbreviations, we may expand the string to the following:

```
\G !M2:4 3E, U 5RS 4EU 5RE / SRE 4S((U 2S))U 0QU / 1QU 5RQ /
```

[Note: Experienced DARMSians will note that 2-suppression is here always in effect; further, all rests default to space code 5 (or 25). This is for ease of input, and is standard with the Note Processor dialect.]

For the convenience of programs which must edit or analyze the data, the string is converted to an array. In this format, for example, every note has its own line in the array, and the successive columns give values for (1) note type; (2) horizontal position; (3) vertical position; (4) accidental and position; (5) stem direction and length; (6) beam beginning or ending and beam number; and (7) prolongation dot and notehead modification. The above example would be represented by the following array:
An important aspect of the array representation is the closeness it preserves to DARMS; in effect, it is an expanded DARMS, which may be read fluently by humans who are familiar with DARMS. (Canonical DARMS, which will very soon be available from the canonizer written by Bruce McClean, is a somewhat more elaborate version of this array representation.) Thus it provides a common link through the various procedures which operate on the code which may be easily examined by users of the program, whether or not the original input medium was DARMS code.

Working with the Code:

The conservative nature of this coding scheme facilitates a great deal of flexibility in processing. By reserving spaces in the array for specific types of information, the information—of its absence—may be examined as necessary. For example, should information about a stem be required at some stage in the processing, then the information may be retrieved with a couple of instructions. If there is no information, then the stem may be calculated from the current context. This way, information which has been encoded at the beginning can take precedence over information which is calculated, and information which is calculated at an earlier stage receives precedence over that which is calculated later; the assumption is always made that any existing information takes priority over whatever might be calculated from context. This is essential for any system which seeks to emulate the process of engraving music, since the nature of that process involves working within a clearly defined rating. In this ranking, things like barlines and clefs are ranked differently and must be dealt with at different times and in different contexts from notes, which in turn are ranked differently from things like accidentals, which are ranked differently from text, and so forth. In other words, with the amount of flux and differentiation involved in the different elements of the data, a stable and clearly defined, if somewhat old-fashioned and unexciting, format for the data is very helpful.

A number of other useful things arise from various features of this structure:

(a) Although different events receive their own lines in the array, each line
is not necessarily an independent entity. In particular, note lines may serve as hooks for other stuff. Text, ornaments, articulations, and other notes may all be hung on a single note, and the sum of all these things are treated (when appropriate) as a single object. At certain stages, moving the note also moves all the things which are hung on it; in the same way, when the spacing for the note is determined, then all the members of the group of things hung on the note are taken into account, so that collisions with nearby objects may be avoided. Furthermore, notes are not the only things which may serve as hooks; barlines, clefs, or anything which can be viewed as having an independent horizontal position can carry other things on them, and become more complex objects. Indeed, empty space may be so defined.

(b) When the input is generated from the mouse-driven graphic screen editor, then the editor can automatically position each new event on the screen, either hanging simultaneous events on the initial group member or advancing to a new horizontal position for a new group. In this case, the spatial relationships are not preserved, and the program recalculates them when the page is formatted. However, the user may instead opt to hand-position any or all elements; in this case, the various spatial relationships are preserved as input to the screen.

(c) When the input is generated from a MIDI interface, then the initial form of the array remains the same, but the data is interpreted differently. Notes are defined only as pitch wheel numbers, with the attack and release points stored, but without further interpretation. Then the array is processed and durations, accidentals, ties, barlines, etc. are generated from this data in successive passes. When the processing is complete, then the data format is the same as that produced by DARMS input.

(d) No matter how the data is entered or edited, it may always be reconverted to DARMS code. This is useful for compact storage (DARMS takes very little space) or for transport to another DARMS-based system.

Final Printing Format:

Although the data format we have been looking at is handy when we need to look at the music from a musical point of view, it is not sufficient for all purposes. There are times when a more graphically oriented format is desirable. In particular,

(a) For printing, it is important to have the various elements of a group broken out into distinct characters; for instance, a note must be represented as a head, stem, accidental, dot, flag, etc., so that the printer does not have to wait for the positions of these various things to be calculated. This is similar to the way in which an engraver's plate is laid out prior to the engraving; the positions of all the characters are lightly scratched on the surface of the plate, so that during the process of actually punching and engraving the plate, the engraver needs only to reach for the die or graver called for by the scratches and place it as indicated.

(b) For editing, it is often desirable to move, add, or delete various elements of a group independently. Sometimes this level of editing is useful to make quick editorial changes immediately prior to printing, or to fine-tune the position of some characters. However, since any character can be positioned anywhere, regardless of meaning or context, such editing provides the capability to create
musical images which would be incomprehensible or difficult to explain to the system.

For these purposes, the music data is stored as a simple character list, with each character positioned and described on a line of an array. Although the editing of the array is normally done either by the formatting programs or by moving the mouse in the screen editor, it may also be stored and edited as an ordinary text file. Thus the final version of a manuscript may be preserved on a floppy disk, with all editorial changes preserved for immediate printing or re-editing.

Yet one other format for the data is available. The page image may be preserved as a bitmap in TIFF format for use by page makeup programs such as Aldus' PageMaker. Although the Note Processor does not have full capabilities for editing these files (the only TIFF editing is in the context of the font editor, which cannot accept very large files), the main purpose for making this format available is so that users can make use of the facilities in other programs for creating pages of mixed music and text, or where other graphic images must be merged into the page. This is an important link for book production, where the data from the Note Processor must be integrated into existing desktop-publishing software.

In Sum:

Having decided on a data format which preserved the underlying structure of an existing and well tested music-description language (DARMS) greatly simplified the completion of the format's definition. Because DARMS accurately parallels another existing and well tested music-description language (the music notation system), this format provided me with a relatively complete description of music to work with in the Note Processor. And this completeness was achieved rather cheaply and painlessly; since the format was initially worked out, there have been no major modifications necessary.

At the same time, the flexibility and speed of a purely graphic description has been an advantage for the final output stages. This provides the sort of flexibility needed for unusual musical situations and for handling non-traditional graphics.

Admittedly, with all the help I got from these useful structures, the program was a great deal longer in the making than I had ever imagined possible; pride prevents me from admitting how much longer. But I doubt that I would ever have been able to bring the project to completion without the flexibility and expandability offered by this basic design.