# *Sonata* Technical Design Specification

*last revised: Fri Jan 30 09:14:01 1987*

The *Sonata* music notation font, developed by Cleo Huggins at Adobe Systems, has been carefully designed to be easy for a POSTSCRIPT developer to work with. Many of the decisions on the sizes and orientations of the glyphs were made specifically with software developers in mind. This document will outline some of the important aspects of the font that will influence its setting in various environments. Please refer to encoding sheets and actual font characters for further enlightenment.

## Character widths

Character widths in ordinary text fonts primarily govern the inter-letter spacing of the text. The width of a character includes the "white space" on each side of the character (the sidebearings), and in POSTSCRIPT, the character width determines *where the current point is left* after printing that character. When printing music, rarely are characters placed one after anothere on a line, as is Roman text. In the POSTSCRIPT environment in which *Sonata* operates, the character widths are used solely to control the location of the current point after printing that character. For many of the *Sonata* character set, the width of the symbol is *exactly the left-to-right width of the symbol itself.* In other words, most characters have zero sidebearings. There are several deviations from this. In situations where the subsequent location has some importance, the character width of a symbol is designed to meet the needs of setting that symbol. For instance, the flag and stem characters are normally used in conjunction with other symbols (such as notes or noteheads), and the registration of the symbols is extremely important. Therefore, the *widths* of these characters are zero in the *Sonata* font, so that the current point will not move when the next symbol is set "on top" of it. See the discussion of composite symbols later in this document for a more thorough examination of this.

## "Origins," or reference points

Each glyph in a POSTSCRIPT font has an "origin", or the reference point from which the character is painted. In the typical sequence of moving to a particular location on a page and printing a font symbol, the point that was originally the "current point" becomes the origin of the character as it is painted. The origins of the characters in the *Sonata* font are designed to minimize the effects of round-off when placing symbols. Characters that are normally set in close registration with one another are designed to require minimal repositioning to set them. For instance, any symbol (or note) that is normally set on the staff has its origin centered about its horizontal axis. To set most symbols on a staff line, one need only move to the correct (and exact) location of the particular line or space on the staff that is desired, and print the appropriate symbol. No complicated calculation is necessary to accurately locate these symbols on the staff. Most of the symbols (with rare exceptions) have their origins at the left edge of the character (or *in line with* the left edge). This decision was made in part to accommodate screen display issues where font display required the origin of the character to be at the left of the character. The exceptions, are the composite characters.

The staff character has its origin at the left edge of the shape, *in the center* of the bottom staff line. Although this bottom line may be of infinitessimal thickness at most point sizes, it is important to realize that the origin is in the center of the line, and no compensation for line thickness is necessary for placement.

## Size issues

Since *Sonata* is a font, it must typically operate in a world set up primarily for text. In most applications, this means that the notion of size is carried in the *point size* of the font. This is based on the printer's point, or 1/72 of an inch. In music, this measure has little to do with anything other than perhaps accompanying text. The *Sonata* font is set up so that the point size carries some meaning. In particular, the point size of the font is *exactly* the distance between the topmost staff line and the bottom staff line of the *staff* character. This measurement is "center to center," which means that no adjustment is necessary to allow for the thickness of the staff lines themselves. All other symbols in the font are designed proportionally to be the correct size. Note that the entire font can be scaled to any size, and that all the symbols will all be scaled proportionately.

The relationship between the staff height and the point size is important when setting music. This allows one to easily determine the distance between staff lines, without extensive computation. *The center-to-center distance between staff lines is always one fourth the point size of the font*. This is an exact measure, and placement of notes may depend on it. If the font is scaled to 24 points, the distance between the first and second staff lines is exactly 6 points. This allows software to easily set up a "coordinate system" based on the staff character that can carry through all setting operations. If the staff is not drawn using the staff character, but instead the POSTSCRIPT line-drawing primitives are used, it is important to set up the staff at an appropriate size for the symbols being used. The rule of thumb given above for sizing the staff should be followed if at all possible.

## Character Set

There are many symbols in the *Sonata* character set which may not be needed by an applications program. Many of them are designed specifically to help solve a difficult placement or setting problem, and may not be needed by all applications. For instance, there are several "backwards" note shapes, with the note head on the opposite side of the stem than expected. These are designed to work in building chords which must share the same stem, without getting alignment problems from having to do many "moveto" operations back and forth. They are designed to be able to maintain a "fixed" X location and only move upward or downward when placing notes for a chord. Similarly, there are two different space characters, with different widths. One of them, called "stemspace", is exactly the width of a notehead *less* the width of a stem. This can be used in special situations where the current point must be left in exactly the right spot for printing a subsequent stem or flag character. There are also two noteheads, a "quarternotehead" and a "noteheadextra," which appear to be identical. In fact, they are identical from POSTSCRIPT's point of view, but they are differentiated so that in screen representations an application can "customize" the look of an upside-down composite note on the screen without affecting its placement. In other words, the printer-font characters are the same, but the screen font characters may differ slightly for display reasons, although they will print identically.

## Encoding

The encoding for the *Sonata* font is designed to be as mnemonic as possible. The "encoding" refers to the indexing of symbols in the font by *byte code*, or ASCII code. This is indirectly tied to keyboard mappings, which generate byte codes, but is also important to a music-setting application which does not even use the keyboard. The encoding for *Sonata* is intended to be easy to use for the lowest common denominator, which is a human being typing at the keyboard. The encoding is based on the Apple Macintosh keyboard layout, in the sense that related characters are grouped by keys on the Macintosh keyboard. The symbols are typically either "visually related" to the key to which they are associated, or they are related "mnemonically" through the actual letter on the keycap. Related characters are typically grouped on the same key and are accessed by using the *SHIFT, OPTION,* and

*COMMAND* keys to get related characters. For instance, the "q" key is associated with the "quarternoteup" glyph, "Q" (or "*SHIFT*-q") gives the "quarternotedown" character, "*OPTION*-q" yields the "quarternotehead" character, and so on. The treble clef is located on the ampersand key ("&"), since it looks somewhat like an ampersand. In general, the *SHIFT* key will "flip" a character upside down, if that makes sense for a given character, and the *OPTION* key will select the "notehead equivalent" of a note. There are many instances where this is strictly not possible or practical, but there is a philosophy in its design that will become evident and which will allow a user to easily remember the location of most of the characters in the font.

## Composite Symbols

The *Sonata* font contains *composite* characters which may be used to build up symbols from pieces, rather than imaging an entire symbol at once. There are standard notes up through 128th notes in the font, for instance, but there is also a separate note head, stem, and two kinds of flags for building these notes up as composites. This is done to accomodate as many "styles" of setting music as possible. If the internal data representation that you work with builds up symbols piece by piece, you may find the composite characters to be very helpful. Otherwise, you can just use the 128th note, or the 64th rest, or whatever you need from the standard selection of characters.

The composite symbols are special in many ways, since their only use is to be set in conjunction with other symbols. They all have character widths of zero, so that the current point will stay put while you build up a symbol. When adding flags to a note, for instance, you may want to move *vertially* to build up many flags, but you needn't move horizontally. If a composite symbol is being set, it is best to set the flags, stems, or other composite parts first (since the *current point* will not be changed as the characters are painted), followed by the note or notehead or other standard symbol upon which the composites may be placed. There are four different stems, six flags, no Georgia, and a host of "backwards" notes for chords. Note that the eighth and sixteenth rests have widths that are narrower than the actual width of the symbols. This is designed to facilitate "stacking" rests without tedious placement. If you just move vertically after placing one rest, the current point will be just right for stacking another rest on top of the existing one.