STANDARD MUSIC FONT LAYOUT (SMuFL)

Daniel Spreadbury
Steinberg Media Technologies GmbH
d.spreadbury@steinberg.de

Robert Piéchaud
robert.piechaud@gmail.com

ABSTRACT

Digital typefaces containing the symbols used in Western common music notation have been in use for 30 years, but the development of the repertoire of symbols that are included, their assignment to code points, and design considerations such as glyph metrics and registration, have been rather *ad hoc*. The Standard Music Font Layout (SMuFL) establishes guidelines for all of these areas, and a reference implementation is available in the Bravura font family.

Software developers and font designers alike are beginning to develop implementations of SMuFL in their products, and benefits including easier data interchange, interoperability of fonts with a variety of software packages, are already being felt.

1. A BRIEF HISTORY OF MUSIC FONTS

Computer software has been displaying musical symbols of various kinds since the 1960s, but the first font for musical symbols did not arrive until 1985, when Cleo Huggins designed Sonata for Adobe.[1]

Sonata mapped the musical symbols onto keys on the standard QWERTY keyboard, using some simple mnemonics (the treble G clef, for example, was mapped onto the **&** key, and the sharp sign onto **#**). Most music fonts developed since then, including Steve Peha's Petrucci (the first music font for the commercial scoring application Finale, dating from 1988[2]) and Jonathan Finn's Opus (the first music font for the commercial scoring application Sibelius, dating from 1993), have followed Sonata's layout.

However, since Sonata includes fewer than 200 characters, and even conventional music notation[3] requires many more symbols than that, individual vendors have devised their own mappings for characters beyond Sonata's initial set.

By 2013, for example, the Opus font family that is still Sibelius's default font set contains no fewer than 18 fonts with more than 600 characters between them.

In 1998, Perry Roland of the University of Virginia drafted a proposal for a new range of musical symbols to be incorporated into the Unicode Standard.[4] This range of 220 characters was duly accepted into the Unicode Standard, and is found at code points U+1D100–U+1D1FF.[5] However, its repertoire of 220 characters does not extend dramatically beyond the scope of the original 1985 version of Sonata, though it does add some characters for mensural and Gregorian notation.

To date the only commercially available music font that uses the Unicode mapping is Adobe Sonata Std, and its repertoire is incomplete.

The designers of other music applications have developed their own approaches to laying out music fonts that are incompatible with both the Sonata-compatible approach, and the Unicode Musical Symbols range. In short, existing standards are either *ad hoc* or insufficient for the development of fonts for rich music notation applications.

2. GOALS FOR A NEW STANDARD

Steinberg began work on a new scoring application at the start of 2013, and quickly identified both the need for a new music font, and the lack of an adequate standard for the layout and design of such a font.

Surveying a range of commercial, open source and freeware music fonts from a variety of sources, and

Copyright: © 2015 Daniel Spreadbury et al. This is an open-access article distributed under the terms of the Creative Commons Attribution License 3.0 Unported*, which permits unrestricted use, distribution, and reproduction in any medium, provided the original author and source are credited.*

[1] See http://www.identifont.com/show?12A

[2] See http://blog.finalemusic.com/post/2010/02/18/Meet-Steve-Peha-creator-of-Petrucci-Finales-first-music-font.aspx

[3] A term coined by Donald Byrd, Senior Scientist and Adjunct Associate Professor of Informatics at Indiana University.

[4] The original proposal is no longer available, but an archived version can be found at http://archive.is/PzkaT

[5] See http://www.unicode.org/charts/PDF/-U1D100.pdf

considering the needs of the in-development application, provided the impetus to create a new standard, with the following goals identified from the outset:

2.1 Extensible by design

The existing Unicode Musical Symbols range is a fixed set of 220 characters in a fixed range of 256 code points at U+1D100–U+1D1FF. This range is not easily extensible, though of course it would be possible for one or more non-contiguous supplemental ranges to be added to future versions of Unicode.

Sonata pre-dates the introduction of Unicode: in common with other PostScript Type 1 fonts of its age, it uses an 8-bit encoding that limits its repertoire of glyphs to a maximum of 256 within a single font. Fonts that broadly follow a Sonata-compatible layout are therefore likewise limited to a maximum of 256 glyphs, and as their developers have needed to further expand their repertoire of characters, they have unilaterally added separate fonts, with no agreement about which characters should be included at which code points.

A new standard should be extensible by design, such that even if the repertoire of characters needs to expand, there is both a procedure for ratifying the inclusion of new characters into the standard, and a means for individual font designers or software developers to add glyphs for their own private use in a way that does not break the standard for other users.

2.2 Take advantage of modern font technologies

The development of the Unicode standard and the OpenType font specification, and their adoption by operating system, software, and font developers, are both enormously important: Unicode provides categorization and structure to the world's language systems, while OpenType enables the development of more advanced fonts with effectively unlimited glyph repertoires and sophisticated glyph substitution and positioning features.

A new standard should enable software developers and font designers to build software that takes advantage of these features, without tying the standard to a specific set of technologies, so that it is as broadly applicable and resistant to future obsolescence as is practical to achieve.

2.3 Open license

In order to minimize the number of obstacles for software developers and font designers to adopt the new standard, it should be free of onerous software licensing terms.

A new standard should be released under a permissive, open license that both protects Steinberg's copyright in the standard, but makes it free for anybody to use in whole or in part in any project, whether that project itself is made available on a commercial basis or under a permissive or free software license.

Accordingly, Steinberg has released SMuFL under the MIT License,[6] which is a permissive free software license that allows reuse within both proprietary and open source software.

2.4 Practical and useful

Although it is impossible to say with certainty why the Unicode Musical Symbols range has failed to gain support among software developers and font designers, it is reasonable to assume that the range did not sufficiently solve the existing problems with the *ad hoc* Sonata-compatible approach, perhaps most crucially the lack of extensibility afforded by the limit of 220 characters, which represented only a very modest expansion of the 176 characters present in Sonata.

A new standard should not only be extensible, but should be developed with the practical needs of software developers and font designers as the top priority, including providing detailed technical guidelines on how to solve some of the issues inherent in representing music notation using a combination of glyphs drawn from music fonts and drawn primitive shapes (stroked lines, filled rectangles, curves, etc.).

2.5 Facilitate easier interchange

As existing music fonts have been developed in isolation by independent software developers and font developers, despite broad intent to make it possible for end users of scoring programs to use a variety of fonts, including those designed for other applications, in practice the level of compatibility between fonts and scoring programs is rather low.

A comparison of the repertoire of glyphs in Sonata, Petrucci, and Opus shows that only 69 of 176 glyphs in Sonata are also present in both Petrucci and Opus; a further 38 glyphs are present in Sonata and Petrucci, but not Opus; and a further 5 glyphs are present in Petrucci and Opus, but not Sonata; a further 59 glyphs in Sonata are present in neither Opus nor Petrucci.

Furthermore, there is no practical way for an end user to know in advance of attempting to use a different font whether or not a given range of characters is implemented in that font, and when transferring documents created in software between systems there is little guarantee that the software can translate the required glyph from one font to another.

A new standard should improve the compatibility of music fonts between different systems by providing not only an agreed mapping of characters to specific code

[6] See `http://opensource.org/licenses/MIT`

points, but also a means for font designers to describe programmatically the repertoire of characters implemented in a given font.

2.6 Build community support

The range of symbols used in Western music notation is so deep and broad that it is difficult for any individual person or small group to have sufficient knowledge to correctly identify and categorize the characters. Furthermore, without broad support among software developers and font designers, any new standard is destined to languish unused.

A new standard should be developed in the open, inviting interested parties to contribute ideas and discussion to the development of the repertoire of characters, their categorization, and technical recommendations about font design, glyph metrics, and glyph registration.

3. NON-GOALS FOR A NEW STANDARD

At the outset of the project, it was determined that, in the short- to medium-term at least, targeting ratification of the new standard by the Unicode Consortium in order to broaden the range of musical symbols encoded by Unicode was not a goal of the project. Developing the standard independently, away from the more rigorous requirements of the proposal and review process, gives greater agility and faster iteration as new requirements emerge.

Initially it was also determined that attempting to develop a set of recommendations for fonts to be used inline with text fonts in word processing or page layout software would be too much work to undertake right away, in addition to the core goal of developing recommendations for fonts to be used in specialized music notation software. However, after the launch of the new standard at the Music Encoding Conference in Mainz, Germany in May 2013, the members of the nascent community identified this as a high priority activity, and the development of guidelines for fonts to be used in text-based applications was added as a requirement for the first stable release of the new standard.

4. WHAT IS SMUFL?

The Standard Music Font Layout, or SMuFL (pronounced “smoofle”), provides both a standard way of mapping music symbols to the Private Use Area of Unicode’s Basic Multilingual Plane, and a detailed set of guidelines for how music fonts should be built.

As a consequence of the joint effort of the community that has arisen around the development of the standard, it also provides a useful categorization of thousands of symbols used in Western music notation.

4.1 Character repertoire and organization

The initial public release of SMuFL, version 0.4, included around 800 characters. By the time of the release of version 1.0, in June 2014, the total number of characters included had grown to nearly 2400, organized into 104 groups.

SMuFL makes use of the Private Use Area within Unicode’s Basic Multilingual Plane (code points from U+E000–U+F8FF). The Unicode standard includes three distinct Private Use Areas, which are not assigned characters by the Unicode Consortium so that they may be used by third parties to define their own characters without conflicting with Unicode Consortium assignments.

SMuFL is a superset of the Unicode Musical Symbols range, and it is recommended that common characters are included both at code points in the Private Use Area as defined in SMuFL and in the Unicode Musical Symbols range.

The groups of characters within SMuFL are based on the groupings defined by Perry Roland in the Unicode Musical Symbols range, but with finer granularity. There are currently 108 groups, proceeding roughly in order from least to most idiomatic, i.e. specific to particular instruments, types of music, or historical periods. The grouping has no significance other than acting as an attempt to provide an overview of the included characters.

Groups are assigned code points in multiples of 16. Room for future expansion has, where possible, been left in each group, so code points are not contiguous. The code point of each character in SMuFL 1.0 is intended to be immutable, and likewise every character has a canonical name, also intended to be immutable. Since the release of SMuFL 1.0, a few additional characters have already been identified that should be added to groups that were already fully populated, and, in common with the approach taken by the Unicode Consortium, new supplemental groups have been added at the end of the list of existing groups to accommodate these additions.

4.2 Inclusion criteria

No formal criteria have been developed for whether or not a given character is suitable for inclusion in SMuFL. Members of the community make proposals for changes and additions to the repertoire of characters, giving rise to public discussion, and once consensus is reached, those changes are made in the next suitable revision.

In general a character is accepted if it is already in widespread use: although composers and scholars invent new symbols all the time, such a symbol can only be included in SMuFL if there is broad community support.

4.3 Recommended and optional glyphs

One of the aims of SMuFL is to make it as simple as possible for developers both of fonts and of scoring software to implement support for a wide range of musical symbols. Although modern font technologies such as OpenType enable a great deal of sophistication in automatic substitution features, applications that wish to use SMuFL-compliant fonts are not obliged to support advanced OpenType features.

The basic requirements for the use of SMuFL-compliant fonts are the ability to access characters by their Unicode code point, to measure glyphs, and to scale them (e.g. by drawing the font at different point sizes). If applications are able to access OpenType features such as stylistic sets and ligatures, then additional functionality may be enabled.

However, all glyphs that can be accessed via OpenType features are also accessible via an explicit code point. For example, a stylistic alternate for the sharp accidental designed to have a clearer appearance when reproduced at a small size can be accessed as a stylistic alternate for the character **accidentalSharp**, but also by way of its explicit code point, which will be in the range U+F400–U+F8FF.

Because optional glyphs for ligatures, stylistic alternates, etc. are not required, and different font developers may choose to provide different sets (e.g. different sets of glyphs whose designs are optimized for drawing at different optical sizes), SMuFL does not make any specific recommendations for how these glyphs should be assigned explicit code points, except that they must be within the range U+F400–U+F8FF, which is reserved for this purpose and for any other private use required by font or application developers.

In summary, recommended glyphs are encoded from U+E000, with a nominal upper limit of U+F3FF (a total of 5120 possible glyphs), while optional glyphs (ligatures, stylistic alternates, etc.) are encoded from U+F400, with a nominal upper limit of U+F8FF (a total of 1280 possible glyphs).

In order for a font to be considered SMuFL-compliant, it should implement as many of the recommended glyphs as are appropriate for the intended use of the font, at the specified code points. Fonts need not implement every recommended glyph, and need not implement any optional glyphs, in order to be considered SMuFL-compliant.

4.4 SMuFL metadata

To aid software developers in implementing SMuFL-compliant fonts, three support files in JSON format [1] are available.

glyphnames.json maps code points to canonical glyph names, which by convention use lower camel case, a convenient format for most programming languages. The file is keyed using the glyph names, with the SMuFL code point provided as the value for the **codepoint** key, and the Unicode Musical Symbols range code point (if applicable) provided as the value for the **alternateCodepoint** key. The **description** key contains the glyph's description.

classes.json groups glyphs together into classes, so that software developers can handle similar glyphs (e.g. noteheads, clefs, flags, etc.) in a similar fashion. Glyphs are listed within their classes using the names specified in **glyphnames.json**. Not all glyphs are contained within classes, and the same glyph can appear in multiple classes.

ranges.json provides information about the way glyphs are presented in discrete groups in this specification. This file uses a unique identifier for each group as the primary key, and within each structure the **description** specifies the human-readable range name, **glyphs** is an array listing the canonical names of the glyphs contained within the group, and the **range_start** and **range_end** key/value pairs specify the first and last code point allocated to this range respectively.

4.5 Font-specific metadata

It is further recommended that SMuFL-compliant fonts also contain font-specific metadata JSON files. The metadata file allows the designer to provide information that cannot easily (or in some cases at all) be encoded within or retrieved from the font software itself, including recommendations for how to draw the elements of music notation not provided directly by the font itself (such as staff lines, barlines, hairpins, etc.) in a manner complementary to the design of the font, and important glyph-specific metrics, such as the precise coordinates at which a stem should connect to a notehead.

Glyph names may be supplied either using their Unicode code point or their canonical glyph name (as defined in the **glyphnames.json** file). Measurements are specified in staff spaces, using floating point numbers to any desired level of precision.

The only mandatory values are the font's name and version number. All other key/value pairs are optional.

The **engravingDefaults** structure contains key/value pairs defining recommended defaults for line widths etc.

The **glyphsWithAnchors** structure contains a structure for each glyph for which metadata is supplied, with the canonical glyph name or its Unicode code point as the key, and is discussed in more detail below.

The **glyphsWithAlternates** structure contains a list of the glyphs in the font for which stylistic alternates are provided, together with their name and code point. Applications that cannot access advanced font features like OpenType stylistic alternates can instead determine the presence of an alternate for a given glyph, and its code point, using this data.

The **glyphBBoxes** structure contains information about the actual bounding box for each glyph. The glyph bounding box is defined as the smallest rectangle that encloses every part of the glyph's path, and is described as a pair of coordinates for the bottom-left (or southwest) and top-right (or northeast) corners of the rectangle, expressed staff spaces to any required degree of precision, relative to the glyph origin.

The **ligatures** structure contains a list of ligatures defined in the font. Applications that cannot access advanced font features like OpenType ligatures can instead determine the presence of a ligature that joins together a number of recommended glyphs, and its code point, using this data.

The **sets** structure contains a list of stylistic sets defined in the font. Applications that cannot access advanced font features like OpenType stylistic sets can instead determine the presence of sets in a font, the purpose of each set, and the name and code point of each glyph in each set, using this data.

4.5.1 Example of how font-specific metadata is used

Figure 1 shows how font-specific metadata may be used in conjunction with the conventions of glyph registration

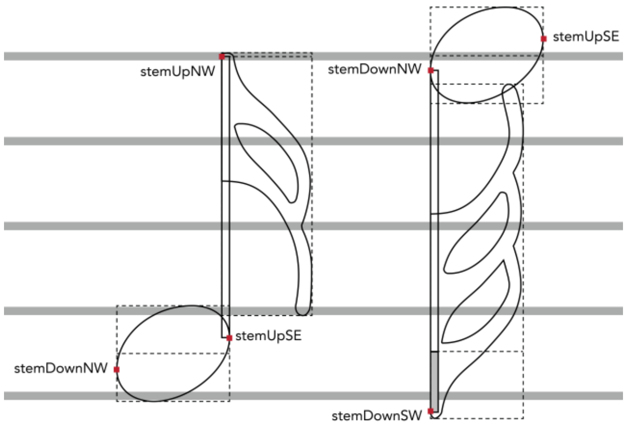

Figure 1 : Diagram illustrating how points defined in font-specific metadata can be used by scoring software.

to construct two notes: an up-stem 16th note (semiquaver), and a down-stem 32nd (demisemiquaver).

• The horizontal grey lines denote staff lines, for scale.

• The dashed boxes show glyph bounding boxes, with the left-hand side of the box corresponding to x=0, while the horizontal lines bisecting the blue boxes show the origin for each glyph, i.e. y=0.

• The shaded red boxes show the locations of the glyph attachment points, as specified in the font metadata JSON file.

• The shaded area on the down-stem note shows the amount by which a stem of standard length (i.e. the unfilled portion of the stem) should be extended in order to ensure good on-screen appearance at all zoom levels.

Note that the **stemUpSE** attachment point corresponds to the bottom right-hand (or south-east) corner of the stem, while **stemDownNW** corresponds to the top left-hand (or north-west) corner of the stem. Likewise, for correct alignment, the flag glyphs must always be aligned precisely to the left-hand side of the stem, with the glyph origin positioned vertically at the end of the normal stem length.

4.6 Glyph registration and metrics recommendations

In addition to providing a standard approach to how musical symbols should be assigned to Unicode code points, SMuFL also aims to provide two sets of guidelines for the metrics and glyph registration, addressing the two most common use cases for fonts that contain musical symbols, i.e. use within dedicated scoring applications, and use within text-based applications (such as a word processors, desktop publishers, web pages, etc.).

Since it is helpful for scoring applications that all symbols in a font be scaled relative to each other as if drawn on a staff of a particular size, and conversely it is helpful for musical symbols to be drawn in-line with text to be scaled relative to the letterforms with which the musical symbols are paired, in general a single font cannot address these two use cases: the required metrics and relative scaling of glyphs are incompatible.

Therefore, it is recommended that font developers make clear whether a given font is intended for use by scoring applications or by text-based applications by appending "Text" to the name of the font intended for text-based applications; for example, "Bravura" is intended for use by scoring applications, and "Bravura Text" is intended for use by text-based applications (or indeed for mixing musical symbols with free text within a scoring application).

The complete guidelines for key font metrics and glyph registration are too detailed to reproduce here, so they can be read in full in the SMuFL specification.[7] Those guidelines that apply to the font as a whole, rather than specific groups of glyphs, are reproduced below.

[7] See `http://www.smufl.org/download`

4.6.1 Guidelines for fonts for scoring applications

Dividing the em in four provides an analogue for a five-line staff: if a font uses 1000 upm (design units per em), as is conventional for a PostScript font, one staff space is equal to 250 design units; if a font uses 2048 upm, as is conventional for a TrueType font, one staff space is equal to 512 design units.

The origin (bottom left corner of the em square, i.e. x = 0 and y = 0 in font design space) therefore represents the middle of the bottom staff line of a nominal five-line staff, and y = 1 em represents the middle of the top staff line of that same five-line staff.

All glyphs should be drawn at a scale consistent with the key measurement that one staff space = 0.25 em.

Unless otherwise stated, all glyphs shall be horizontally registered so that their leftmost point coincides with x = 0.

Unless otherwise stated, all glyphs shall have zero-width side bearings, i.e. no blank space to the left or right of the glyph.

4.6.2 Guidelines for fonts for text-based applications

Upper case letters in a text font do not typically occupy the whole height of the em square: instead, they typically occupy around 75–80% of the height of the em square, with the key metrics for ascender and caps height both falling within this range. In order for the line spacing of a font containing music characters to be equivalent to that of a text font, its key metrics must match, i.e. the ascender, caps height and descender must be very similar. Glyphs with unusually large ascenders and descenders (such as notes of short duration with multiple flags) should not be scaled individually in order to fit within the ascender height, as they will not then fit with the other glyphs at the same point size; however, the behavior of glyphs that extend beyond the font's ascender and descender metrics is highly variable between different applications.

Leading on from the premise that a SMuFL-compliant font for text-based applications should use metrics compatible with regular text fonts, specific guidelines are as follows:

Dividing 80% of the height of the em in four provides an analogue for a five-line staff. If a font uses 1000 upm (design units per em), as is conventional for a PostScript font, the height of a five-line staff is 800 design units, or 0.8em; therefore, one staff space height is 200 design units, or 0.2 em. If a font uses 2048 upm, as is conventional for a TrueType font, the height of a five-line staff is 1640 design units, and one staff space is 410 design units.

The origin (bottom left corner of the em square, i.e. x = 0 and y = 0 in font design space) therefore represents the middle of the bottom staff line of a nominal five-line staff, and y = 0.8 em represents the middle of the top staff line of that same five-line staff.

Unless otherwise stated, all glyphs should be drawn at a scale consistent with the key measurement that one staff space = 0.2 em.

Unless otherwise stated, all glyphs shall be horizontally registered so that their leftmost point coincides with x = 0.

Unless otherwise stated, all glyphs shall have zero-width side bearings, i.e. no blank space to the left or right of the glyph.

Staff line and leger line glyphs should have an advance width of zero, so that other glyphs can be drawn on top of them easily.

5. REFERENCE FONT

To demonstrate all of the key concepts of SMuFL, a reference font has been developed. The font family is called Bravura, and consists of two fonts: Bravura, which is intended for use in scoring applications; and Bravura Text, which is intended for use in text-based applications.

The word *Bravura* comes from the Italian word for "cleverness", and also, of course, has a meaning in music, referring to a virtuosic passage or performance; both of these associations are quite apt for the font. From an aesthetic perspective, Bravura is somewhat bolder than most other music fonts, with few sharp corners on any of the glyphs, mimicking the appearance of traditionally-printed music, where ink fills in slightly around the edges of symbols, and the metal punches used in plate engraving lose their sharp edges after many uses. A short musical example set in Bravura is shown below (Figure 2).

Steinberg has released the Bravura fonts under the SIL Open Font License [2]. Bravura is free to download, and can be used for any purpose, including bundling it with other software, embedding it in documents, or even using it as the basis for a new font. The only limitations placed on its use are that: it cannot be sold on its own; any derivative font cannot be called "Bravura" or contain "Bravura" in its name; and any derivative font must be released under the same permissive license as Bravura itself.

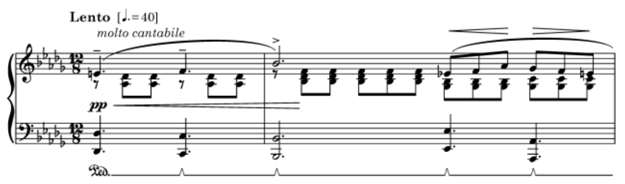

Figure 2. Example of the Bravura font.

6. IMPLEMENTATION CASE STUDY: THE NOVEMBER FONT

Unlike designers of text fonts, music font designers have historically had great freedom, which has been both a blessing and a curse. Before SMuFL, while there was some common sense about what the kernel of music symbols should be (clefs, noteheads, accidentals, etc.), the actual position of characters in the font, their naming (though there was generally none provided), and the addition of rarer symbols beyond the basic set was left up to the designer's imagination and to some specific requirements of the target music notation software.

Things are changing for the font designer with SMuFL as its main goal is to address the issues of symbol position, naming and repertoire in a universal way. SMuFL is a great source of inspiration for the designer – surely one of its benefits – but it also imposes new constraints and requirements, and leads to a more demanding design workflow.

6.1 The November Font – Summary

The November music font was designed in 1999 specifically for the software Finale, and its repertoire of 330 characters, spread over two font files, ranging through historical periods spanning the Renaissance to the 20th century *avant garde*, was considered large at that time. Before SMuFL, the extension of November's repertoire had often been considered, but it would have most likely led to the multiplication of font files, as had occurred with, for example, Opus or Maestro, which the designer was reluctant to do, and consequently only small updates had been made over the years.

6.2 Moving to SMuFL

The emergence of SMuFL back in 2013 was a great opportunity for November to make a bigger jump: one single font file with a greatly extended range of characters, wrapped in OpenType, and complying with a new standard.

By switching to SMuFL, the font designer, who generally is a single individual, must be ready to face the temptation of adding more and more symbols, making the development process potentially much longer.[8] And not only must the designer deal with thousands of vectors and points, but also to some extent he or she must turn into a programmer. Python scripting, for instance, can be a great ally for generating the required metadata automatically; this was used extensively for the November 2.0 project.[9] For SMuFL-scaled font projects, it is impractical to create those metadata manually, and,

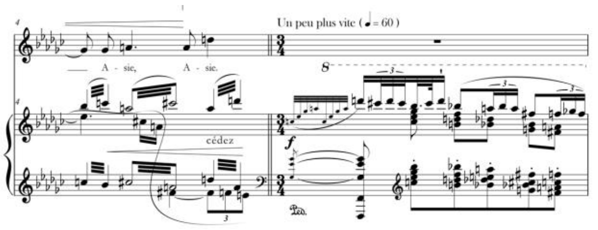

Figure 3. Example of the November 2.0 font.

to make the design workflow even better, one can invent sophisticated tools, for instance to compare the font being crafted with the reference font, Bravura. All of these considerations change the font development workflow deeply.

November 2.0, released in February 2015, now has over 1100 characters, with about 80% of them coming from the SMuFL specifications, and is the first commercially-released font to comply with SMuFL. A short musical example set in November 2.0 is shown below (Figure 3).

6.3 Compatibility with existing scoring software

Unlike the font Bravura, which for now has largely served as a reference font for SMuFL, commercial SMuFL-compliant music fonts are intended to be used in existing music notation programs.

At the present time, no currently available notation software officially directly supports SMuFL, though such support is likely forthcoming in the future. In the short- to medium-term, therefore, a SMuFL-compliant font like November 2.0 must still be packaged specifically for each notation program. The SMuFL metadata, for instance, is currently not consumed at all by any of the major existing applications (including Finale, Sibelius, and LilyPond), and idiosyncratic component files[10] must be supplied along with the font in order to ensure a smooth user experience.

But in a positive way, the claim of SMuFL-compliance for a popular music font like November can potentially help serve as an impetus for the developers of music notation software to support SMuFL more quickly.

7. SUPPORT FOR SMUFL

SMuFL 1.0 was released in June 2014. The standard remains under active development, and it is hoped that an increasing number of software developers and font designers will adopt it for their products. Below is a

[8] Somehow the designer could not resist this temptation with November 2.0 in any case!

[9] November 2.0 was made with the open source program FontForge (`http://fontforge.github.io/`), which has a powerful Python interface.

[10] Finale's Font Annotations and Libraries, Sibelius's House Styles, LilyPond's snippets…

summary of the projects that have been publicly announced that are making use of SMuFL.

7.1 Software with SMuFL support

Steinberg's forthcoming scoring application will support SMuFL-compliant fonts.

The open source scoring application MuseScore supports SMuFL-compliant fonts in version 2.0, which is currently in beta testing.[11]

The web browser-based interactive sheet music and guitar tablature software Soundslice uses SMuFL and Bravura for its music notation display.[12]

The open source Music Encoding Initiative (MEI) rendering software, Verovio, also uses SMuFL for its music notation display.[13]

The commercial scoring application Finale, from MakeMusic Inc., will support SMuFL in a future version[14]. MakeMusic's MusicXML import/export plug-in for Finale, Dolet, supports SMuFL as of version 6.5.[15]

The commercial digital audio workstation application Logic Pro X, from Apple Inc., supports SMuFL and is compatible with Bravura from version 10.1.[16]

7.2 Fonts with SMuFL support

In addition to the reference font Bravura, other SMuFL-compliant music fonts are beginning to be available.

MuseScore 2.0 includes SMuFL-compliant versions of Emmentaler and Gootville, based respectively on the Emmentaler and Gonville fonts designed for use with LilyPond.

Verovio includes a SMuFL-compliant font called Leipzig.

Robert Piéchaud has designed an updated version of his November font family that is SMuFL-compliant[17].

8. FUTURE DIRECTIONS

Although SMuFL has reached version 1.0 and contains an enormous range of characters, it remains under active development, and further minor revisions are expected for the indefinite future as new characters are identified, proposed, and accepted for inclusion, and as the need for new or improved metadata is identified.

It is also expected that MusicXML, a widely-used format for the interchange of music notation data between software of various kinds, will develop closer ties to SMuFL in its next major revision, version 4.0, which may necessitate some changes to SMuFL.

9. CONCLUSIONS

In this paper, a new standard for the layout of musical symbols into digital fonts has been outlined. The new standard, called the Standard Music Font Layout (SMuFL) is appropriate for modern technologies such as Unicode and OpenType. Through community-driven development, the standard has reached version 1.0 and includes nearly 2400 characters, categorized into 104 groups, and is poised for future expansion as necessary. A reference font family, Bravura, has been developed to promote the adoption of the new standard. Both SMuFL and Bravura are available under permissive free software licenses, and are already being adopted by software developers and font designers.

Acknowledgements

SMuFL is developed in the open by a community of music software developers, academics, font designers, and other interested parties. Too many people to list here have contributed to the development of the standard to date, and their contributions have been of great value to the project.[18]

10. REFERENCES

[1] ECMA-404, *The JSON Data Interchange Format, 1st edition*, 2013.

[2] N. Spalinger and V. Gaultney, *SIL Open Font License (OFL)*, 2007.

[11] See `http://musescore.org/en/node/30866`

[12] See `http://www.soundslice.com`

[13] See `https://rism-ch.github.io/verovio/smufl.xhtml?font=Leipzig`

[14] See `http://www.sibeliusblog.com/news/finale-2014d-and-beyond-a-discussion-with-makemusic/`

[15] See `http://www.musicxml.com/dolet-6-5-finale-plugin-now-available/`

[16] See `http://support.apple.com/en-us/HT203718`

[17] See `http://www.klemm-music.de/notation/november/index.php`

[18] A full list of contributors is printed in the SMuFL specification, which can be found at `http://www.smufl.org/download`