SING Glyphlet Production: Tips, Tricks & Techniques

Technical Note #5148

August 2, 2006

Contents

Contents

1 SING Glyphlet Production

1.1 Introduction

This tutorial is intended for font developers who plan to build SING glyphlets, and is specifically designed to provide tips, tricks, and techniques that go beyond the standard documentation and specifications, and in effect, offer a more easily understood interpretation of them. Note that this document is not a substitute for the standard documentation and specifications, but rather complements them.

While this initial version is written in English, a Japanese translation is planned and forthcoming. Until the Japanese translation is available, please note that this English version was specifically written to be clear, concise, and easy to understand.

If you find any errors or omissions in this Adobe Tech Note, please contact its author, Dr. Ken Lunde, at the following email address, in English or in Japanese:

```
lunde@adobe.com
```

1.1.1 Conventions

This document assumes that the Adobe-supplied *cvt2sing* tool is used for building glyphlets. It is available for a variety of platforms, such as Windows and Mac OS X.

Unicode values are indicated through the use of the standard "U+" prefix notation, and does not imply a particular encoding. For example, U+0020 represents an ASCII "space," U+4E00 represents the CJK Unified Ideograph that means *one*, and U+20000 represents the first character in Plane 2.

1.2 Glyphlet Types & Categories

Glyphlets, in terms of the glyphs that they carry throughout the workflow, can be categorized in different ways, yet they are ways that are regular and predictable. Each category has different methods for constructing the metadata, in terms of its content.

1.2.1 Glyphlets Based on Known Glyph Collection

Under some circumstances, a type foundry may wish to make the glyphs for some of their designs available as individual glyphlets, rather than issuing a complete font. In this case, the glyphlets can be from an established glyph collection, such as Adobe-Japan1-5 or Adobe-Japan1-6. Information stored in the 'META' table makes the relationship to a known glyph collection explicit.

1.2.2 Glyphlets Beyond Known Glyph Collections

For glyphs that are not found in known glyph collections—meaning that they are variants of glyphs in known glyph collections—or for glyphs of parent characters not covered by known glyph collections, glyphlets can be built to make these accessible by users through existing (installed) fonts or as separate glyphs. The metadata that is recorded in the 'META' table becomes critical in allowing users to enter the glyphs into their documents. The more rich the metadata, the greater the ease in which users can enter the glyphs.

1.2.3 Generic Glyphlets

Generic glyphs, whether they are considered variants of glyphs in known glyph collections, or beyond the scope of those glyph collections, benefit from an adequate amount of metadata to allow users to enter them into their documents. Corporate logos are but a single example of a generic glyphlet.

1.3 Tools

A variety of tools can be used to create and verify glyphlets. The tools supplied by Adobe are generally cross-platform, and will run on Mac OS X and Windows.

Always be sure that you are using the latest version of these tools, to ensure that the best possible glyphlets are being built. Also ensure that you are using the latest version of the specifications.

1.3.1 Glyphlet Production

The *cvt2sing* tool is the general-purpose glyphlet production tool. There are two modes available. One is single-glyphlet mode, meaning that a single execution builds a single glyphlet, specifying parameters on the command line. The other is batch mode, in which a single file serves as input to *cvt2sing*, each line of which contains the parameters for building a single glyphlet.

The following command lines build two glyphlets, using CID 23056 and 23057 of `R.cff` as the source font, and using the `ADBE_KozMin-R_C23056.xml` and `ADBE_KozMin-R_C23057.xml` files as metadata, respectively:

```
% cvt2sing -g /23056 -m XML/ADBE_KozMin-R_C23056.xml -dd ./GAI R.cff
% cvt2sing -g /23057 -m XML/ADBE_KozMin-R_C23057.xml -dd ./GAI R.cff
```

The following command line builds the same glyphlets, but uses batch mode:

```
% cvt2sing -s R.txt
```

The following lines represent the contents of the `R.txt` file:

```
-g /23056 -m XML/ADBE_KozMin-R_C23056.xml -dd ./GAI R.cff
-g /23057 -m XML/ADBE_KozMin-R_C23057.xml -dd ./GAI R.cff
```

1.3.2 Glyphlet Verification

Besides using the glyphlets within a workflow, or with applications that are SING-savvy, two Adobe-supplied tools are useful for inspecting the contents of glyphlets.

The *OTFProof* tool allows any 'sfnt' table to be viewed, usually in human-readable form, and often in more than one way.

The *tx* tool can be used to inspect the 'CFF' tables of glyphlets, and can also be used to subset the source fonts, which is a technique for increasing the speed at which *cvt2sing* runs. This technique is described elsewhere in this document.

1.4 Source Fonts

There are several factors to consider when preparing the source fonts, specifically the fonts from which *cvt2sing* extracts the glyphs for building glyphlet files.

Note that if a CIDFont file or a CID-keyed OpenType font is used as the source font for glyphlets, whatever glyph collection designation (/Registry, /Ordering, and /Supplement, such as "Adobe-Japan1-6") that the source font advertises will be flattened to become "Adobe-Identity-0" in the glyphlet's 'CFF' table. Adobe-Identity-0 is a language- and script-independent glyph collection designation. If a name-keyed font is used as the source font, no glyph collection designation will be advertised in the resulting glyphlet's 'CFF' table.

1.4.1 Handling and Specifying GID=0

GID=0 in a glyphlet file is extracted from the source font. Its purpose is the same as that of any other font, specifically to be used as the "undefined" (aka, ".notdef") glyph, and is required. The behavior of *cvt2sing* is as follows:

- If the source font is named-keyed, its ".notdef" glyph is extracted and used for the glyphlet's GID=0.

- If the source font is CID-keyed, the glyph of CID=0 is extracted and used for the glyphlet's GID=0.

It doesn't matter whether GID=0 is printing or non-printing, but it simply must be present in the glyphlet's 'CFF' table. Note that GID=0 or the ".notdef" is a required glyph in fonts, including a glyphlet's 'CFF' table, and in the context of glyphlets is not expected to be displayed or otherwise used.

1.4.2 Font Size Versus Speed

The speed at which *cvt2sing* can build glyphlet files is related to the number of glyphs in the source font.

As an example, if you were to build a set of kanji-only glyphlets that represent the delta of Adobe-Japan1-4 and Adobe-Japan1-6—meaning the 5,525 glyphs in the two CID ranges 16779–20316 (Adobe-Japan1-5) and 21071–23057 (Adobe-Japan1-6)—using a complete Adobe-Japan1-6 source font (CIDs 0 through 23057), it takes approximately 15 seconds to build each glyphlet. But, if you were to include in the source font only those glyphs to be included in the glyphlets—CIDs 16779–20316 and 21071–23057, along with the mandatory CID=0 for the ".notdef" glyph—approximately three to four glyphlets can be built per second. This nearly fifty-fold increase in speed is significant, and demonstrates the usefulness of this technique to increase production speed.

Note that the *tx* tool can be used to easily generate a subset CFF font containing only those glyphs that are necessary for building glyphlets. The following command line creates a CFF font (with a filename of `5525-R.cff`) that contains only CIDs 0, 16779–20316, and 21071–23057:

```
% tx -g 0,16779-20316,21071-23057 -cff KozMinProVI-Regular.otf > 5525-R.cff
```

1.5 XML Issues

Writing or generating syntactically-correct XML files is crucial in developing industrial-strength glyphlets. The following sections detail some aspects that require more care than usual, in the context of building glyphlets.

1.5.1 UTF-8 Encoding

It is recommended that UTF-8 be specified as the encoding for the XML files used for building glyphlets. The following line, the very first line of the XML file, declares this:

```
<?xml version="1.0" encoding="utf-8"?>
```

UTF-8 encoding is a multiple-byte representation of Unicode that is compatible with ASCII (any valid ASCII value—0x00 through 0x7F—is also a valid UTF-8 value, representing the same character), and uses a mixed one- through four-byte encoding. All BMP characters outside of ASCII are represented using two or three bytes. All non-BMP characters are represented using four bytes.

1.5.2 UTF-8 Encoding Versus Unicode Character XML References

The use of UTF-8 encoding makes possible two pathways for specifying non-ASCII characters, as follows:

- Directly encoded as UTF-8 (0xE4B880 for U+4E00, 0xF0A08080 for U+20000, and so on)

- Unicode Character XML References (`一` for U+4E00, `𠀀` for U+20000, and so on)

Note that most Japanese characters are represented with three bytes using UTF-8 encoding, and a smaller number are handled using two or four bytes.

Throughout this document, all non-ASCII characters in sample XML code are represented using Unicode Character XML References.

The use of Unicode Character XML References to represent non-ASCII characters in XML files ensures that the file is "seven-bit clean," and thus far less likely to become damaged or corrupt, through transmission, or when manipulated by clients that are not capable of fully supporting UTF-8 encoding.

NOTE: When generating Unicode Character XML References, always be sure to include the trailing semicolon. Some metadata strings use a semicolon as a field separator, so in some cases the correct syntx will require two contiguous semicolons, such as in the following META.ID=1 (UNIUnifiedBaseChars) string: `一;a` (made up of three components: `一` plus `;` plus `a`).

1.5.3 Unicode Character XML Reference Issues

When using Unicode Character XML References in XML files, and if you intend to have an ASCII "space" (U+0020) character adjacent to it, meaning directly before or directly after it, you need to specify an explicit "space" character using the appropriate Unicode Character XML Reference, as follows:

` `

The following is an example of an incorrect copyright string:

`Copyright © 2005 Adobe Systems Incorporated. All Rights Reserved.`

The "space" before and after the Unicode Character XML Reference, for the copyright symbol, will not appear in the glyphlet's corresponding 'META' table entry. A correctly-formed copyright string is as follows:

`Copyright © 2005 Adobe Systems Incorporated. All Rights Reserved.`

1.5.4 Minimal XML File

The following is a minimal XML file, which can serve as the basis for creating your own XML files:

```
<?xml version="1.0" encoding="utf-8"?>
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
xmlns:SING="http://ns.adobe.com/SING/1.0/">
<![CDATA[ <unicodeVersion> 30100 </unicodeVersion> <nMetaRecs> 1 </nMetaRecs> ]]>
<rdf:Description about="GID@1">

</rdf:Description>
</rdf:RDF>
```

A blank line serves to indicate where the metadata entries are placed.

1.5.5 Learning Through Example

Adobe Systems provides to developers a large number of example XML files, along with the glyphlets that are built from them. These files, and others, should be thoroughly and carefully explored to better understand the recommendations and conventions set forth in this document.

1.6 META Table—General Issues

The 'META' table is the heart of a glyphlet, and provides the information that allows it to be effectively used within a workflow. It is critical that its contents be made carefully.

Generating metadata content by hand is error-prone due to its complexity. We strongly recommend that a database be used to manage the metadata, and used to generate the XML files that serve as input to the *cvt2sing* tool.

1.6.1 Non-BMP Code Points

Non-BMP code points, such as those in Plane 2, should be specified using either UTF-8 encoding, which means four bytes per character, or as Unicode Character XML References. The use of a High and Low Surrogate, such as the following for U+20000, is not permitted:

```
&#xD840;&#xDC00;
```

The correct Unicode Character XML Reference for U+20000 is as follows:

```
&#x20000;
```


1.6.2 PUA Code Points

PUA (Private Use Area) code points shall not be used in glyphlets to the extent that it would allow PUA code points to persist within the workflow. There are three PUA regions within Unicode, as follows:

● U+E000 through U+F8FF (Private Use Area)

- U+F0000 through U+FFFFD (Supplementary Private Use Area-A)

- U+100000 through U+10FFFD (Supplementary Private Use Area-B)

Earlier versions of *cvt2sing* allow PUA code points to be used in fields that would result in PUA usage within the workflow. The current version of *cvt2sing* will issue a fatal error, and will fail to build the glyphlet file (or files) if PUA code points are used in the following fields:

- META.ID=1 (UNIUnifiedBaseChars)

- META.ID=602 (UNIDerivedByOpenType)

- META.ID=603 (UNIOpenTypeYields)

1.6.3 Setting Permissions

We recommend that embedding permissions for glyphlets be set such that glyphlets can become an effective part of the entire workflow. The SING.permissions field is analogous to the FSType and OS/2.type fields as used in CIDFonts and OpenType/TrueType fonts, respectively.

The following XML line sets the SING.permissions field to 0, which is the recommended value:

```
<SING:Permissions> 0 </SING:Permissions>
```

Note that if SING.permissions is set to 0, the embedded glyphlet can be freely copied from one document to another. Setting this field to 8 prevents this.

1.6.4 Glyphlet Naming Conventions

We recommend that a glyphlet name be made up of three or four portions, separated by an underscore (U+005F). Note that the total length of the glyphlet name, excluding the file extension, cannot exceed 27 characters.

The first portion is the Developer ID, such as 'ADBE' for Adobe Systems. If your company does not have a Developer ID, please contact Adobe Developer Support to have one registered or assigned.

The second portion is a descriptive name for the glyphlet, made up of twelve (12) or less characters. For non-generic glyphlets, this is typically set to something that resembles the font or fonts for which it is designed. For Kozuka Mincho Regular glyphlets, Adobe Systems sets this to 'KozMin-R'.

The third portion is set to one of the following:

- 'C' plus a zero-padded five-digit decimal CID value—if the glyphlet is specifically designed to be identical to a CID within a known glyph collection, such as Adobe-Japan1-6.

- "U" plus a zero-padded five-digit hexadecimal Unicode value.

In the latter case, a fourth field, a zero-padded two-digit variant identifier, is suggested.

Some examples are as follows:

```
ADBE_KozMin-R_C23057
ADBE_Pi-Circle400_U025C9_00
ADBE_Pi-Circle400_U025C9_01
```

Note that you may choose any convention, but the use of the Developer ID at the beginning is very strongly recommended so as to avoid naming conflicts with other developers.

The glyphlet name becomes manifested in the following four ways within a glyphlet file:

- As the file name itself
- META.ID=65000 (UniqueName)
- SING.uniqueName
- CFF.FontName

1.6.5 Localized Menu Names

When setting the localized menu names, as META.ID=25 (LocalizedBaseFontName), in addition to setting non-English menu names, be sure to also include English-language menu names. For example, this is beneficial when a Chinese-language glyphlet is used on a Japanese system. English-language menu names provide a common ground.

1.6.6 OpenType Only

When setting the BaseFontName and LocalizedBaseFontName fields, the only fonts that should be listed are those that correspond to OpenType fonts. Only OpenType fonts can be augmented with glyphlets. And, only OpenType fonts provide the necessary foundation for advanced typographic functionality required for glyphlets, such as glyph substitution.

More details about how these fields should be set, and how they relate to one another, are found elsewhere in this document.

1.6.7 Vertical Variants

When the primary glyph of the glyphlet (GID=0 of its CFF table) has a vertical variant, the use of the "-V" option in the following command line assigns the vertical variant to GID=2:

```
% cvt2sing -g /16236 -V /16333 -m ADBE_KozMin-R_C16236.xml R.cff
```

The resulting glyphlet will have the glyphs for CIDs 16236 and 16333 of the source font assigned to GIDs 1 (primary; horizontal) and 2 (vertical), respectively.

1.6.8 References to CIDs

When referencing CIDs in META table fields, specifically in META.IDs 20 (CIDBaseChars), 604 (CIDDerivedByOpenType), and 605 (CIDOpenTypeYields), it critical to note that the meaning or usage of the glyphlet must match that of the CID specified, or must be a variant thereof. Breaking this rule will result in workflow failure.

Because this point is absolutely critical for complete workflow success, it shall be repeated throughout this document at the appropriate places.

1.7 META Table—Specific Fields

The following sections detail the contents of specific 'META' table fields. Note that not all fields are covered, specifically those that are likely to be of greater use. For more information about a particular 'META' table field, or to discover additional ones not covered here, you are encouraged to read the META table specification.

1.7.1 MojikumiX4051 (ID=0)

For Japanese glyphlets, meaning those that have META.ID=3 set to `ja-JP`, the JIS X 4051 Mojikumi class shall be specified. A complete list of the integer values, from 1 to 20, along with their meanings, are provided in the 'META' table specification.

CJK Unified Ideographs, for example, are in class 12, and this is specified as follows:

```
<SING:MojikumiX4051> 12 </SING:MojikumiX4051>
```

For supporting glyph classes outside the context of JIS X 4051, such as in specific implementations, user-defined or vendor-specific metadata entries can be used. A section of this document describes such usage, with examples.

1.7.2 UNIUnifiedBaseChars (ID=1)

If the glyphlet is directly mapped from a Unicode code point, or can be considered a variant thereof, this field should be included, and specified accordingly. Note that this field can appear more than once. Consider the following entries, which correspond to Adobe-Japan1-5 CID=16781:

```
<SING:UNIUnifiedBaseChars> &#x4F6A;;a </SING:UNIUnifiedBaseChars>
<SING:UNIUnifiedBaseChars> &#x5F8A;;v </SING:UNIUnifiedBaseChars>
```

the first line uses the "a" specifier, which serves to indicate that the glyphlet represents the "actual" glyph for the specified code point. That is, U+4F6A maps to CID=16781. This mapping is true of any Adobe-Japan1-5 or higher font.

The second line uses the "v" specifier, which serves to indicate that the glyphlet is a variant of the specified code point, and is thus not mapped directly.

For glyphlets whose glyphs are not considered a variant of an existing character in Unicode, the following shall be used:

```
<SING:UNIUnifiedBaseChars> &#xFFFD;;v </SING:UNIUnifiedBaseChars>
```

Note the intentional use of the "v" specifier to indicate that the glyphlet is a variant.

Furthermore, if U+FFFD is used, META.ID=500 (UNICODEproperty_GeneralCategory) shall also be set, and in the case of kanji, the appropriate value is Lo (Letter, other), as follows:

```
<SING:UNICODEproperty rdf:parseType='Resource'>
    <SING:Key> GeneralCategory </SING:Key>
    <SING:Value> Lo </SING:Value>
</SING:UNICODEproperty>
```

1.7.3 BaseFontName (ID=2)

If a glyphlet is intended to be generic, to be used with any typeface within its language and script, regardless of style or weight, this field must be set to "Generic" as exemplified below:

```
<SING:BaseFontName> Generic </SING:BaseFontName>
```

When including specific font names, bear in mind that only OpenType fonts be specified, and that this field is set to match the CIDFontName. As an example, consider Adobe Systems' glyphlets that cover Adobe-Japan1-5 and Adobe-Japan1-6. This field should be set as follows:

```
<SING:BaseFontName> KozMinStd-Regular </SING:BaseFontName>
<SING:BaseFontName> KozMinPro-Regular </SING:BaseFontName>
```

It is important to note that "KozMin-Regular" is not specified, because it does not correspond to an OpenType font. Also note that "KozMinProVI-Regular" is not specified, because it includes all Adobe-Japan1-5 and Adobe-Japan1-6 glyphs. If the META.ID=21 (IsUpdatedGlyph) field were to be set to a non-zero value, specifying "KozMinProVI-Regular" would become appropriate.

1.7.4 Language (ID=3)

The language of the glyphlet is indicated by using the notation specified in RFC 1766, and thus consists of a two-letter lowercase language code plus a two-letter uppercase country code, separated by a hyphen, as follows:

- Japanese `ja-JP`
- Korean `ko-KR`
- Simplified Chinese `zh-CN`
- Traditional Chinese `zh-TW`

The following is an example that specifies that the language of the glyphlet is Japanese:

```
<SING:Language> ja-JP </SING:Language>
```

If a glyphlet supports more than one language, multiple instances may be specified.

1.7.5 WritingScript (ID=8)

The intended writing script of the glyph can be made explicit through the use of a four-letter writing script code as defined in the OpenType specification. The appropriate writing script code for CJK Unified Ideographs is `hani`, and is expressed as follows in the XML file:

```
<SING:WritingScript> hani </SING:WritingScript>
```

The use of `DFLT` indicates that the glyph is independent of any writing script:

```
<SING:WritingScript> DFLT </SING:WritingScript>
```

Note that this field does not need to be included for glyphs with unambiguous usage, in terms of layout needs. For example, it is not necessary for glyphs that represent CJK Unified Ideographs because they are used left-to-right in horizontal writing mode, and top-to-bottom in vertical writing mode.

1.7.6 StrokeCount (ID=10)

If the glyphlet represents a CJK Unified Ideograph, a single instance of this field can be specified. Note that only one instance shall be specified. The following is an example that indicates that the total number of strokes of the glyph is 23:

```
<SING:StrokeCount> 23 </SING:StrokeCount>
```

Note that this integer value shall be the same as the sum of the integer values in corresponding pairs of META.IDs 11 and 12, linked by the same indexing radical.

1.7.7 IndexingRadical (ID=11)

If the glyphlet represents a CJK Unified Ideograph, one or more indexing radicals, within the range U+2F00 through U+2FD5 can be specified, followed by the number of strokes in the instance of the indexing radical. The following is used for Adobe-Japan1-5 CID=16824:

```
<SING:IndexingRadical> &#x2F8B;;4 </SING:IndexingRadical>
<SING:IndexingRadical> &#x2F21;;3 </SING:IndexingRadical>
<SING:IndexingRadical> &#x2F22;;3 </SING:IndexingRadical>
```

1.7.8 RemStrokeCount (ID=12)

After factoring out the strokes in the indexing radical (META.ID=11), the number of remaining strokes can be specified, followed by the indexing radical. Like META.ID=11, multiple instances are allowed. The following is used for Adobe-Japan1-5 CID=16824:

```
<SING:RemStrokeCount> 19;&#x2F8B; </SING:RemStrokeCount>
<SING:RemStrokeCount> 20;&#x2F21; </SING:RemStrokeCount>
<SING:RemStrokeCount> 20;&#x2F22; </SING:RemStrokeCount>
```

Note that the use of the indexing radical, such as U+2F8B in the first line, allows the information in META.IDs 11 and 12 to be correctly correlated.

1.7.9 DesignCategory (ID=15)

The 'META' table specification provides a list of the integer values for this field, ranging from 0 to 5. Serif-style designs, such as Mincho, use the integer value 1, as follows:

```
<SING:DesignCategory> 1 </SING:DesignCategory>
```

For symbols and logos, that are not specific to any typeface style, or are generic, the integer value 0 shall be used, as follows:

```
<SING:DesignCategory> 0 </SING:DesignCategory>
```

1.7.10 DesignWeight (ID=16)

The relative weight of the glyphlet is recorded in this field, and should match the `OS/2.weightClass` value in its parent font. The following is used for Kozuka Mincho Regular glyphlets:

```
<SING:DesignWeight> 400 </SING:DesignWeight>
```

For glyphs that are weight-independent, such as generic ones, a zero (0) shall be used, as follows:

```
<SING:DesignWeight> 0 </SING:DesignWeight>
```

1.7.11 DesignWidth (ID=17)

The design width of the glyphlet is recorded in this field, and should match the `OS/2.widthClass` value in its parent font. For typical Japanese glyphs that use a square design space (aka, em-square), the value should be set to 5, as exemplified below:

```
<SING:DesignWidth> 5 </SING:DesignWidth>
```

For glyphs that are width-independent, such as generic ones, a zero (0) shall be used, as follows:

```
<SING:DesignWidth> 0 </SING:DesignWidth>
```

Please consult the META table specification for more complete descriptions of the non-zero values of this field.

1.7.12 DesignAngle (ID=18)

For typical glyphs that have a square or otherwise rectangular design space, a value of this field shall be zero (0), as exemplified below:

```
<SING:DesignAngle> 0 </SING:DesignAngle>
```

Non-zero values, ranging from 1 to 4, indicate the direction of obliquing, in a positive or negative direction, and along the X- (horizontal) or Y-axis (vertical).

Please consult the META table specification for more complete descriptions of the non-zero values of this field.

1.7.13 CIDBaseChars (ID=20)

If the glyph of the glyphlet is identical to or a variant of a glyph in a known glyph collection, such as Adobe-Japan1-6, this field is used to indicate such a relationship. Consider a glyphlet whose glyph is identical to that of Adobe-Japan1-5 CID=16781:

```
<SING:CIDBaseChars> Adobe-Japan1;16781;a </SING:CIDBaseChars>
<SING:CIDBaseChars> Adobe-Japan1;4790;v </SING:CIDBaseChars>
```

This field, when present, is made up of three components. The first component is the /Registry and /Ordering strings of the glyph collection separated by a hyphen, and in the case of the example above, is Adobe-Japan1. The second component is the CID of the glyph collection. The third component uses an "a" or "v" specifier to indicate whether the CID represents the "actual" or "variant" glyph, respectively. These three fields are separated by a semicolon (U+003B).

NOTE: When referencing CIDs in META table fields, specifically in META.IDs 20 (CIDBaseChars), 604 (CIDDerivedByOpenType), and 605 (CIDOpenTypeYields), it critical to note that the meaning or usage of the glyphlet must match that of the CID specified, or must be a variant thereof. Breaking this rule will result in workflow failure.

1.7.14 IsUpdatedGlyph (ID=21)

If a glyphlet is issued to serve as a replacement for or correction of an existing glyph in a font, this field shall be set to a non-zero value. Normal glyphlets shall have this field set to 0 (zero), as follows:

```
<SING:IsUpdatedGlyph> 0 </SING:IsUpdatedGlyph>
```

There are three non-zero integer values that are possible, depending on whether the replacement is done at the CID or Unicode level, or both. The three non-zero values are as follows:

● 1—The replacement is done at the Unicode level, using the ";a" (actual) value or values in META.ID=1 (UNIUnifiedBaseChars).

- 2—The replacement is done at the CID level, using the ";a" (actual) value or values in META.ID=20 (CIDBaseChars).

- 3—The replacement is done at the Unicode and CID levels, as indicated for values 1 and 2.

1.7.15 VendorGlyphURI (ID=22)

This field allows the developer to assign to the glyph a unique identifier, either to reference an existing standard such as a glyph collection specification, or to uniquely identify the glyph in terms of shape and font attributes. The following XML code identifies the glyph as being the same as Adobe-Japan1-6 CID=23057:

```
<SING:VendorGlyphURI>
      http://glyphs.adobe.com/collection/Adobe-Japan1/cid23057
</SING:VendorGlyphURI>
```

1.7.16 LocalizedInfo (ID=24)

This field is used to encapsulate information that would be found in name.ID strings 0, 3, 7, 8, 9, 10, 11, 12, 13, 14, or 19 of complete OpenType fonts. This field is made up of four parts, described as follows:

- The encodingID for the string—only 1 (BMP-only Unicode) and 10 (full Unicode) are valid values, and unless your string contains non-BMP characters, 1 is recommended.

- The languageID for the string, expressed in hexadecimal—for example, 404 indicates Traditional Chinese (Taiwan), 409 indicates American English, 411 indicates Japanese, 412 indicates Korean, and 804 indicates Simplified Chinese (Mainland China).

- The name.ID to which the string corresponds.

- The string itself—only Unicode strings are allowed.

The following exemplifies appropriate XML entries that correspond to name.IDs 0, 7, 8, 9, 11, and 14:

```
<SING:LocalizedInfo>
     1;409;0;Copyright&#x0020;&#x00A9;&#x0020;2004-2005 Adobe Systems
Incorporated. All Rights Reserved.
</SING:LocalizedInfo>
<SING:LocalizedInfo>
     1;409;7;Kozuka Mincho is either a registered trademark or trademark of
Adobe Systems Incorporated in the United States and/or other countries.
</SING:LocalizedInfo>
<SING:LocalizedInfo>
     1;409;8;Adobe Systems Incorporated
</SING:LocalizedInfo>
```

```
<SING:LocalizedInfo>
    1;409;9;Masahiko Kozuka&#x0020;&#x5c0f;&#x585a;&#x660c;&#x5f66;
</SING:LocalizedInfo>
<SING:LocalizedInfo>
    1;411;9;Masahiko Kozuka&#x0020;&#x5c0f;&#x585a;&#x660c;&#x5f66;
</SING:LocalizedInfo>
<SING:LocalizedInfo>
    1;409;11;http://www.adobe.co.jp/products/type/
</SING:LocalizedInfo>
<SING:LocalizedInfo>
    1;409;14;http://www.adobe.com/type/legal.html
</SING:LocalizedInfo>
```

Note the explicit use of ` ` to indicate a space (U+0020).

More information about name.ID fields and languageIDs can be found in the 'name' table description of the OpenType specification:

http://www.microsoft.com/typography/otspec/name.htm

1.7.17 LocalizedBaseFontName (ID=25)

First and foremost, if META.ID=2 (BaseFontName) is set to "Generic," this field shall not be specified.

For every META.ID=2 (BaseFontName) entry that exists, at least one META.ID=25 (LocalizedBaseFontName) entry must exist. For non-Western glyphlets, such as those intended for Japanese use, English names are strongly recommended, in case the glyphlets are used in a non-Japanese environment.

The following correspond to the META.ID=2 (BaseFontName) entries for KozMinStd-Regular:

```
<SING:LocalizedBaseFontName>
    KozMinStd-Regular;1;409;Kozuka Mincho Std;R
</SING:LocalizedBaseFontName>
<SING:LocalizedBaseFontName>
    KozMinStd-Regular;1;411;&#x5C0F;&#x585A;&#x660E;&#x671D;&#x0020;Std;R
</SING:LocalizedBaseFontName>
```

And, the following correspond to the META.ID=2 (BaseFontName) entries for KozMinStd-Regular:

```
<SING:LocalizedBaseFontName>
    KozMinPro-Regular;1;409;Kozuka Mincho Pro;R
</SING:LocalizedBaseFontName>
<SING:LocalizedBaseFontName>
    KozMinPro-Regular;1;411;&#x5C0F;&#x585A;&#x660E;&#x671D;&#x0020;Pro;R
</SING:LocalizedBaseFontName>
```

Note the explicit use of ` ` to indicate a space (U+0020).

1.7.18 ReadingString: GlyphName (ID=100)

The name of the glyph, in the form of a descriptive or informative string that details its source or properties, can be encapsulated in this field. Note that multiple instances of this field are allowed, in case the glyph has multiple names.

Consider Adobe-Japan1-5 CID=16779, which is from the JIS X 0212-1990 and JIS X 0213:2004 character set standards. Appropriate XML code to indicate this is provided below:

```
<SING:ReadingString rdf:parseType='Resource'>
    <SING:Key> GlyphName </SING:Key>
    <SING:Value> JIS X 0212-1990 16-72 </SING:Value>
</SING:ReadingString>
<SING:ReadingString rdf:parseType='Resource'>
    <SING:Key> GlyphName </SING:Key>
    <SING:Value> JIS X 0213:2004 1-14-14 </SING:Value>
</SING:ReadingString>
```

Because this field is not mandatory, and simply informative, if there is not a convenient name for the glyph, this field can be safely omitted.

1.7.19 ReadingString: JapaneseYomi (ID=101)

The Japanese reading string, expressed with hiragana and katakana (U+3041 through U+30FE), as appropriate for "On" and "Kun" readings, along with "+" (U+002B) used as the okurigana separator, is expressed in this field. Multiple readings necessitate multiple instances of this field.

Consider Adobe-Japan1-5 CID=20073, whose readings can be expressed using the XML code provided below:

```
<SING:ReadingString rdf:parseType='Resource'>
    <SING:Key> JapaneseYomi </SING:Key>
    <SING:Value> &#x30E8; </SING:Value>
</SING:ReadingString>
<SING:ReadingString rdf:parseType='Resource'>
    <SING:Key> JapaneseYomi </SING:Key>
    <SING:Value> &#x3042;&#x305F;+&#x3048;&#x308B; </SING:Value>
</SING:ReadingString>
```

1.7.20 ReadingString: JapaneseTranslit (ID=102)

Although this field can be included in glyphlets, META.ID=101 (ReadingString— JapaneseYomi) is more useful because it less ambiguously indicates the reading, and can thus more directly interact with Input Methods. In other words, META.ID=101 yields a more powerful and more useful glyphlet than one that uses META.ID=102 instead.

Consider Adobe-Japan1-5 CID=20073, whose transliterated readings can be expressed using the XML code provided below:

```
<SING:ReadingString rdf:parseType='Resource'>
    <SING:Key> JapaneseTranslit </SING:Key>
    <SING:Value> yo </SING:Value>
</SING:ReadingString>
<SING:ReadingString rdf:parseType='Resource'>
    <SING:Key> JapaneseTranslit </SING:Key>
    <SING:Value> ata+eru </SING:Value>
</SING:ReadingString>
```

This field exhibits some degree of ambiguity, because there are multiple transliteration methods that can be used, such as the Hepburn, Kunrei, Nippon, and Waapuro systems. The use of META.ID=101 exhibits little or no ambiguity, and is thus recommended for Japanese readings.

To exemplify the degrees of ambiguity, consider one of the "Kun" readings of Adobe-Japan1-0 CID=2887, expressed as follows using hiragana, along with four transliteration systems:

- Hiragana おお
- Hepburn ō
- Kunrei ô
- Nippon ô
- Waapuro oo

Note how there is no ambiguity when the reading is expressed using hiragana, but that there are three different transliteration instances, one of which is shared by two transliteration systems.

1.7.21 Additional ReadingString Fields (IDs 103 through 114)

META.IDs 103 through 114 provide additional ReadingString fields for supporting Chinese, Korean, and Vietnamese readings, along with transliteration systems thereof. Please consult the META table specification for more details.

1.7.22 AltIndexingRadical Fields (IDs 200 through 202)

If the glyph represents a CJK Unified Ideograph, and if META.ID=11 (IndexingRadical) cannot sufficiently express the indexing radical, one of the AltIndexingRadical fields can be used to indicate the indexing radical, expressed as an integer.

1.7.23 AltLookup Fields (IDs 250 through 257)

There are several fields that allow common Chinese character dictionary indexes to be recorded for a glyphlet. If a dictionary provides an unambiguous indexing scheme that corresponds to a glyph, such as an enumeration expressed as an integer, it shall be used directly. For example, the appropriate META.ID=253 (AltLookup_DaiKanwa) value for Adobe-Japan1-6 CID=22947 is exemplified below:

```
<SING:AltLookup rdf:parseType='Resource'>
    <SING:Key> DaiKanwa </SING:Key>
    <SING:Value> 45636 </SING:Value>
</SING:AltLookup>
```

If this dictionary were to lack an unambiguous indexing scheme, the following would represent the scheme that is set forth in the META table specification:

```
<SING:AltLookup rdf:parseType='Resource'>
    <SING:Key> DaiKanwa </SING:Key>
    <SING:Value> 12p658.5a </SING:Value>
</SING:AltLookup>
```

This scheme is made up of four or five parts, described as follows:

● The volume number, if any, expressed as an integer—the example indicates Volume 12.

● The page number, expressed as an integer, prefixed with a lowercase "p" (U+0070)—the example indicates Page 658.

● A "period" (U+002E) to act as a separator.

● The relative position of the dictionary entry, expressed as an integer, and starting from 1—the example indicates 5.

● Whether the dictionary entry is the actual glyph, or a variant thereof, expressed as "a" or "v," respectively—the example indicates "a."

1.7.24 UNICODEproperty: GeneralCategory (ID=500)

If META.ID=1 (UNIUnifiedBaseChars) is set to U+FFFD, and if the glyphlet does not represent a bi-directional, combining class, math, mirrored, or hangul character, this field shall be set to an appropriate two-letter Unicode General Category identifier. For CJK Unified Ideographs, the appropriate value is Lo which stands for "Letter, other." The following is an example:

```
<SING:UNICODEproperty rdf:parseType='Resource'>
    <SING:Key> GeneralCategory </SING:Key>
    <SING:Value> Lo </SING:Value>
</SING:UNICODEproperty>
```

1.7.25 Additional UNICODEproperty Fields (IDs 501 through 505)

META.IDs 501 through 505 provide additional UNICODEproperty fields, for handling other types of Unicode-specific data to be encapsulated in a glyphlet. Please consult the 'META' table specification for more information these additional UNICODEproperty fields.

1.7.26 UNIDerivedByOpenType (ID=602)

Please read and understand the important notes below before proceeding:

NOTE: This and the following three sections shall use Adobe-Japan1-5 CID=20295 for the examples. Seeing how these four fields are related and interact, given the same source glyph, is important for understanding how their contents are to be constructed.

NOTE: Glyphlets can directly represent only two OpenType GSUB feature layout rule types: one-to-one and *n*-to-one (aka, ligature) substitution. Note that *n*-from-one layout rules can be simulated by including multiple one-to-one substitution instances, as exemplified in this and the following three sections.

If the glyph of the glyphlet can result when a Unicode value and an OpenType GSUB feature are applied, an entry for this field should be included. Note that multiple instances can be included, for multiple GSUB features, and for multiple instances of the same GSUB feature if multiple Unicode values are expected to result in the glyph of the glyphlet.

The following are examples for Adobe-Japan1-5 CID=20295, based on the Adobe-Japan1-6 GSUB feature set:

```
<SING:UNIDerivedByOpenType> &#x877F;;aalt </SING:UNIDerivedByOpenType>
<SING:UNIDerivedByOpenType> &#x8805;;aalt </SING:UNIDerivedByOpenType>
<SING:UNIDerivedByOpenType> &#x8805;;jp04 </SING:UNIDerivedByOpenType>
<SING:UNIDerivedByOpenType> &#x8805;;nlck </SING:UNIDerivedByOpenType>
```

In order to add discretionary ligature support to glyphlets, through the use of the 'dlig' GSUB feature, simply specify a string of two or more Unicode values. The following is an example for a discretionary ligature ('dlig' GSUB feature) that uses a string of six katakana characters ( ) for expressing "glyphlet" in Japanese:

```
<SING:UNIDerivedByOpenType>
    &#x30B0;&#x30EA;&#x30D5;&#x30EC;&#x30C3;&#x30C8;;dlig
</SING:UNIDerivedByOpenType>
```

This field is identical to META.ID=604 (CIDDerivedByOpenType), but uses a Unicode value (or values) instead of a glyph collection designation plus CID (or CIDs).

1.7.27 UNIOpenTypeYields (ID=603)

If the glyph of the glyphlet can result in a Unicode value when an OpenType GSUB feature is applied, an entry for this field should be included. Note that multiple instances can be

included, for multiple GSUB features, and for multiple instances of the same GSUB feature if multiple Unicode values are expected.

The following are examples for Adobe-Japan1-5 CID=20295, based on the Adobe-Japan1-6 GSUB feature set:

```
<SING:UNIOpenTypeYields> aalt;&#x877F; </SING:UNIOpenTypeYields>
<SING:UNIOpenTypeYields> aalt;&#x8805; </SING:UNIOpenTypeYields>
```

This field is identical to META.ID=605 (CIDOpenTypeYields), but uses a Unicode value instead of a glyph collection designation plus CID.

1.7.28 CIDDerivedByOpenType (ID=604)

If the glyph of the glyphlet can result when a CID (of the specified glyph collection) and an OpenType GSUB feature are applied, an entry for this field should be included. Note that multiple instances can be included, for multiple GSUB features, and for multiple instances of the same GSUB feature if multiple CIDs (of the specified glyph collection) are expected to result in the glyph of the glyphlet.

The following are examples for Adobe-Japan1-5 CID=20295, based on the Adobe-Japan1-6 GSUB feature set:

```
<SING:CIDDerivedByOpenType> Adobe-Japan1;3358;aalt </SING:CIDDerivedByOpenType>
<SING:CIDDerivedByOpenType> Adobe-Japan1;6537;aalt </SING:CIDDerivedByOpenType>
<SING:CIDDerivedByOpenType> Adobe-Japan1;14212;aalt </SING:CIDDerivedByOpenType>
<SING:CIDDerivedByOpenType> Adobe-Japan1;6537;jp04 </SING:CIDDerivedByOpenType>
<SING:CIDDerivedByOpenType> Adobe-Japan1;6537;nlck </SING:CIDDerivedByOpenType>
```

In order to add discretionary ligature support to glyphlets, through the use of the 'dlig' GSUB feature, simply specify a list of two or more CIDs, using a space (U+0020) as a separator. The following is an example for a discretionary ligature ('dlig' GSUB feature) that uses a string of six katakana characters (   ) for expressing "glyphlet" in Japanese:

```
<SING:CIDDerivedByOpenType>
    Adobe-Japan1;940 998 977 1000 959 964;dlig
</SING:CIDDerivedByOpenType>
```

This field is identical to META.ID=602 (UNIDerivedByOpenType), but uses a glyph collection designation plus CID (or CIDs) instead of a Unicode value (or values).

NOTE: When referencing CIDs in META table fields, specifically in META.IDs 20 (CIDBaseChars), 604 (CIDDerivedByOpenType), and 605 (CIDOpenTypeYields), it critical to note that the meaning or usage of the glyphlet must match that of the CID specified, or must be a variant thereof. Breaking this rule will result in workflow failure.

1.7.29 **CIDOpenTypeYields (ID=605)**

If the glyph of the glyphlet can result in a CID (of the specified glyph collection) when an OpenType GSUB feature is applied, an entry for this field should be included. Note that multiple instances can be included, for multiple GSUB features, and for multiple instances of the same GSUB feature if multiple CIDs (of the specified glyph collection) are expected.

The following are examples for Adobe-Japan1-5 CID=20295, based on the Adobe-Japan1-6 GSUB feature set:

```
<SING:CIDOpenTypeYields> aalt;Adobe-Japan1;3358 </SING:CIDOpenTypeYields>
<SING:CIDOpenTypeYields> aalt;Adobe-Japan1;6537 </SING:CIDOpenTypeYields>
<SING:CIDOpenTypeYields> aalt;Adobe-Japan1;14212 </SING:CIDOpenTypeYields>
```

This field is identical to META.ID=603 (UNIOpenTypeYields), but uses a glyph collection designation plus CID instead of a Unicode value.

NOTE: When referencing CIDs in META table fields, specifically in META.IDs 20 (CIDBaseChars), 604 (CIDDerivedByOpenType), and 605 (CIDOpenTypeYields), it critical to note that the meaning or usage of the glyphlet must match that of the CID specified, or must be a variant thereof. Breaking this rule will result in workflow failure.

1.8 **BASE Table Overrides**

By default, the *cvt2sing* tool will set the 'BASE' table such that the design space extends from 0,-120 to 1000,880, specifically a 1000×1000 box set 120 units below the Latin baseline. It also sets an ICF (Ideographic Character Face) that is a 920×920 box, centered within the 1000×1000 design space. These values can be easily overridden in the XML file, and the following settings represent the default values:

```
<SING:BASEhoriz> -80 840 -120 0 </SING:BASEhoriz>
<SING:BASEvert> 40 960 0 120 </SING:BASEvert>
```

If one wanted to set these values for a design space that is set 200 units below the Latin baseline, the following values should be used:

```
<SING:BASEhoriz> -160 760 -200 0 </SING:BASEhoriz>
<SING:BASEvert> 40 960 0 200 </SING:BASEvert>
```

1.9 **User-defined & Vendor-specific Metadata**

META.IDs 20000 through 32767 have been established to provide a large number of "VendorSpecific" metadata fields, for encapsulating metadata that is not pre-defined as a field in the META table specification. For example, InDesign has a slightly richer set of glyph classes than those defined in JIS X 4051, and these can be easily specified in a vendor-specific field, as exemplified below:

```
<VendorSpecific rdf:parseType='Resource'>
    <Key> 20001 </Key>
    <Value>
    <ADOBE_InDesignMojikumiClassID> 33 </ADOBE_InDesignMojikumiClassID>
    </Value>
</VendorSpecific>
```

Note that the META.ID value, 20001 in this example, is between the `<Key>` and `</Key>` tags. Also note that the vendor-specific tags themselves, along with the value to be stored, 33 in this example, are between the `<Value>` and `</Value>` tags.