White Paper: Legacy Gaiji Solutions & SING

Version: July 16, 2008

Dr. Ken Lunde
Senior Computer Scientist, CJKV Type Development, Adobe Systems Incorporated
lunde@adobe.com

There have been a myriad of solutions to the so-called *gaiji problem*, collectively referred to as *legacy gaiji solutions* in this document. Given the complex and broad nature of the gaiji problem, it should not be a surprise that many solutions have been developed and deployed over the years. Most importantly, no one can deny that many of these legacy gaiji solutions work, but with obvious limitations or drawbacks. The first two pages of this White Paper shall focus on these legacy gaiji solutions. The last two pages shall focus on SING, an acronym for *Smart INdependent Glyphlets*, as a *gaiji solution* without the drawbacks of legacy gaiji solutions.

The Gaiji Problem

The gaiji problem can be easily described as the absence of desired glyphs. Because fonts represent a critical or key component of writing using digital media, and because fonts are finite collections of glyphs, which are based on one or more character set standards, it is natural and expected that there will be times when a desired glyph is not present in a font. Western writing systems are, for the most part, limited and closed. Their letters combine in a sequence to form higher-level linguistic entities, such as words. East Asian writing systems include thousands or tens of thousands of ideographs, which represent an open-ended writing system. It is a trivial process to coin a new ideograph, or to create a new variant of an existing ideograph.

The gaiji problem also applies to Western writing systems, such as for logos or new symbols. When a new currency symbol is established, there is always a frantic scramble to add its glyph to existing fonts, and to also make sure that it becomes part of Unicode in a timely manner. In fact, in the past we provided "euro symbol" gaiji fonts that supplemented legacy fonts.

What Is A Gaiji?

The prototypical *gaiji* is an ideograph, and is either a stand-alone character, or a variant form of an ideograph that is already encoded. But, a gaiji can be *any* glyph, as long as the user cannot enter it. A gaiji can be a generic symbol, a corporate logo, or a new currency symbol. It may or may not be in Unicode. It may be in a known glyph set. It may be in a font, but simply not accessible due to IME limitations or other reasons. What is key is that the desired glyph is either not available in any installed font, or is desired in the style of the currently selected font, but is not available in that style.

Legacy Gaiji Solutions

Legacy gaiji solutions are almost always font-based, but other techniques have been implemented by those without the ability to develop fonts, such as inline graphics, and even hand-written after the document has been printed, using a textual element or graphic as a place-holder or indicator. Because nearly all gaiji are intended to be typeset as though they were characters, font-based solutions are almost always desired over solutions that involve inline graphics.

Legacy gaiji solutions can and do work *most* of the time. And, for some users, these legacy gaiji solutions might be ideal, because they are able to achieve the desired end result. The more closed a system or environment is, the better these legacy gaiji solutions tend to function. Today's trend, however, is toward open systems. Legacy gaiji solutions fall short when *portability* and *document interchange*, necessitated by open systems or environments, become part of the workflow.

What Motivates Gaiji Solutions?

What drives the vast majority of legacy gaiji solutions is simply being aware that if one encodes a glyph, and as long as the code point of the glyph is known, it can be entered into a document. Thus, there is a very strong motivation to encode gaiji. The encoding requirement is why legacy gaiji solutions work most of the time, and for some users in some environments, all of the time.

How Are Gaiji Encoded?

Early gaiji fonts were encoded in a single-byte array, meaning less than 256 code points were used. Multiple single-byte gaiji fonts were once used if the number of gaiji exceeded the limits of a single-byte array. Now, Shift-JIS encoding's 1,880 user-defined code points, or Unicode's PUA (*Private Use Area*) are generally used for encoding gaiji. Unicode supports 6,400 PUA code points in the BMP (*Basic Multilingual Plane*), and an additional 131,068 in Planes 15 and 16. *Adobe Type Composer* (ATC) allowed users to effectively add multiple single-byte gaiji fonts to existing fonts, but only on Mac OS, only for specific font formats, and only for Shift-JIS encoding. Microsoft's EUDC (*End User Defined Character*) mechanism that is specific to Windows® makes use of PUA code points.

Another approach to encoding gaiji involves a technique referred to as *code point poaching* in which an existing—and typically inappropriate—code point is used to encode gaiji. To some extent, single-byte gaiji fonts use this technique. Code point poaching can result in unpredictable behavior due to the properties that are associated with code points, which can affect line breaking, spacing, and other line-layout behaviors. PUA code points have no intrinsic properties, so their behavior tends to be more predictable.

However, PUA code points work only if no one is using the same code point for a different purpose. Consider two different gaiji, made by two different users or companies, that are encoded at the same PUA code point. This is a recipe for trouble, especially if these two gaiji are expected to be used in the same font, system, or environment.

How Are Gaiji Entered Into Documents?

Because gaiji are encoded, what ultimately is entered into a document is a code point, and the selected font hopefully encodes the gaiji at the same code point. Input by code point, or through the use of a palette or panel, is common. For larger and somewhat more standardized collections of gaiji, such as those offered by Biblos Font, additional dictionary entries for one or more commonly used IMEs are often included, which effectively enable user input through the use of single-character or compound readings.

Gaiji Font Implementations

In the past, before OpenType® and Unicode became widely used, typical gaiji fonts were implemented as one or more single-byte Type 1 fonts. These Type 1 gaiji fonts would be used standalone, or combined and added to the user-defined region of a larger font using ATC. OpenType effectively enabled gaiji font creators to break away from the single-byte limitations, and Unicode increased the number of available code points at which the gaiji can be encoded.

Why Do Legacy Gaiji Solutions Fail?

Portability is the primary reason why legacy gaiji solutions fail. Their code points, whether PUA or poached, are not portable. As long as the system or environment is closed, portability is obviously not an issue. If document interchange is important, even within a closed system or environment, all the components that comprise the legacy gaiji solution must be included with the document. Legacy gaiji solutions thus depend on one or more special fonts to be installed.

In summary, we can state that the gaiji problem issue that needs to be solved is thus portability, which affects the glyphs themselves, and the requirement to encode them.

SING—Smart INdependent Glyphlets

Adobe Systems exhaustively explored and carefully studied the shortcomings of *legacy gaiji solutions*, and SING, an acronym for *Smart INdependent Glyphlets*, was conceived as a new *gaiji solution*.

Put simply, SING was specifically designed from the beginning to be a gaiji solution without the shortcomings and deficiencies that plague legacy gaiji solutions. Where legacy gaiji solutions fail the all-important *portability* test, meaning that *document interchange* is a challenge and hurdle, SING passes this test with flying colors. SING also does not require that gaiji be encoded in order to be minimally or fully functional.

SING represents a gaiji solution that is implemented through the creation, distribution, and use of small font-like objects called *SING glyphlets*, along with libraries and applications to manage and use them.

What Is A SING Glyphlet?

A SING glyphlet is effectively a small OpenType font that lacks the key tables, such as 'name' and 'OS/2', that would otherwise allow it to become selectable in application font menus.

A SING glyphlet includes one meaningful glyph at GID+1. Additional functional glyphs, such as a vertical variant, or other contextual variants, can be included at GID+2 or greater. Because the typical number of glyphs in a SING glyphlet is one, it is therefore small and lightweight.

The small and lightweight nature of a SING glyphlet means that it can travel easily and quickly. A SING glyphlet is intended to be sticky to the document in which it is used, by being embedded. A SING glyphlet is thus portable. This characteristic of SING clearly solves the portability issue that is inherent in legacy gaiji solutions. Due to its architecture, gaiji that are implemented as SING glyphlets are easily recognized and distinguished from other glyphs. When a standard font resource is used as a gaiji font, distinguishing gaiji from non-gaiji becomes a much greater challenge, and necessitates heuristics. Heuristics have a strong tendency to fail.

The glyph that is represented by the SING glyphlet does not need to be encoded. If there is an appropriate Unicode code point for its primary glyph, it can be used, but this is not a requirement. The metadata that is included in the 'META' table serves to identify and classify the glyph so that it can behave as though it were a standard character. Legacy gaiji solutions require that their glyphs be encoded, because they cannot be embedded in the document in which they are used.

How Does SING Work?

The current implementation of SING, as deployed in CCJK versions of Adobe® InDesign® CS2 and CS3 software, makes use of two libraries. The *SING Library*, used by *Adobe SING Glyphlet Manager* (ASGM), organizes SING glyphlets, and makes them available to SING-*savvy* applications. The *Tin Library* is used by SING-savvy applications to augment installed fonts with the SING glyphlets that refer to them in their META.IDs 2 (*BaseFontName*) and 25 (*LocalizedBaseFontName*) fields.

After the Tin Library augments installed fonts with the appropriate SING glyphlets, the glyphs that correspond to the SING glyphlets behave as though they are among the standard glyphs in the installed fonts. They appear in the Glyph Panel, and can interact with other glyphs.

How Are SING Glyphlets Created?

SING glyphlets can be created, by developers and end users, through the use of at least four known tools, two of which were developed by Adobe Systems.

Adobe Systems developed and currently maintains the *cvt2sing* command-line tool, which is a very robust and industrial-strength SING glyphlet compiler, and is included in the *Glyphlet*

Development Kit (GDK).* Metadata and other information are specified in an XML file, which serves as input to *cvt2sing*, along with a font resource that contains the desired glyph data.

Adobe Illustrator® CS2 and later, purchased as part of the Adobe Creative Suite® with Adobe InDesign, includes a plug-in called *Glyphlet Creation Tool* (GCT) that is specifically designed to build SING glyphlets, with the obvious added feature to design the glyphs themselves using the strong drawing capabilities in Illustrator, along with access to the glyph outlines of installed fonts.

FontLab's *SigMaker*, Version 3.0 and later, includes options and features for building metadata-rich SING glyphlets, and runs on Mac OS X and Windows.†

Musashi System's *SINGEdit* represents another program for building SING glyphlets.‡ Although the program itself runs only on Windows, the SING glyphlets that it builds are cross-platform.

Why Does SING Better Serve Customer Needs Than Legacy Gaiji Solutions?

As stated in the first half of this White Paper, the more closed a system or environment is, the better that legacy gaiji solutions tend to function. SING is specifically designed to work with open systems and environments, and is expected to function the same way no matter how closed or open a system or environment is. No legacy gaiji solution can make such a claim.

Removing the requirement to encode gaiji clearly distinguishes SING from legacy gaiji solutions that require their glyphs to be encoded. In fact, SING disallows PUA code points to be used in the 'META' table of SING glyphlets.

The most profound advantages of SING, when compared to legacy gaiji solutions, are its portability and document interchange properties, along with the self-contained nature of SING glyphlets. In other words, SING glyphlets are intended to travel with the documents in which they are used, through all stages of the documents' workflow. SING glyphlets are thus sticky to a document. The fact that SING glyphlets are small, lightweight, and self-contained makes portability and document interchange possible.

How Do We Make SING Successful?

First and foremost, relative to the Adobe InDesign CS2/CS3 implementations, SING requires performance enhancements, in terms of the speed of font augmentation, which subsequently affects its ability to handle a larger number of glyphlets. This performance enhancement work is already underway.

PDF Version 1.3 and greater, is SING-*compatible* in that SING glyphlets properly embed as glyphs, and thus display and print properly, but they lose their all-important metadata.

Lastly, SING needs to be supported by third-party applications and at the OS level. Clearly, in order for this to happen, we must first make SING functional and successful in more of our own applications. Our own efforts with regard to the deployment of SING sends a clear and strong message to third-party application and OS developers, in terms of our commitment to the technology, and the extent to which it is supported in our own products.

We are very fortunate that we develop a broad range of applications that cover the document workflow, meaning that even without third-party application, IME, or OS support, we have the ability to deliver a fully-functional gaiji solution to our customers through SING. A business case for SING necessarily spans the entire globe, not merely Japan. The global publishing market cannot be conquered until the gaiji problem has been solved.

* http://www.adobe.com/devnet/opentype/gdk/topic.html

† http://www.fontlab.com/font-utility/sigmaker/

‡ http://musashi.or.tv/singedit.htm