CID-keyed fonts

CID-keyed fonts represent the basic underlying technology used for building today's (and tomorrow's) PostScript CJKV fonts. CID stands for *Character ID*, and it is a character and glyph access type for PostScript. Technically speaking, a CID-keyed font is a Type 0 (composite) font with FMapType 9. A CID-keyed font can also be CIDFontType 0, 1, or 2, depending on the type of character descriptions that are included. CIDFontType 0 is by far the most common CID-keyed font.

There are two components that constitute a valid a CID-keyed font: a CIDFont resource that contains the glyph descriptions (outlines), along with other data necessary to properly render them, such as hinting information; and one or more *Character Map* (CMap) resources that are used to establish character-code to CID mappings. The CIDs are ultimately used to index into the CIDFont resource for retrieving glyph descriptions.

A valid CID-keyed font instance consists of a CIDFont plus CMap concatenated using one or two hyphens.* Table 6-11 lists some CIDFonts and CMaps, along with the corresponding valid CID-keyed fonts.

Table 6-11. CIDFont and CMap resources versus CID-keyed fonts

CIDFont resource	CMap resource	CID-keyed font
Munhwa-Regular	KSC-H	Munhwa-Regular--KSC-H
HeiseiMin-W3	H	HeiseiMin-W3-H
MSung-Light	CNS-EUC-V	MSung-Light--CNS-EUC-V

When is it appropriate to use one versus two hyphens as glue? For CID-keyed fonts that were once available as OCF fonts, such as HeiseiMin-W3-H (that is, the "HeiseiMin-W3" CIDFont resource name plus "-" plus the "H" CMap resource name), a single hyphen is recommended for compatibility, because the OCF font machinery does not understand two hyphens, and will fail to execute, or "find," the font. Otherwise, two consecutive hyphens are recommended.

I intentionally did not label CID-keyed fonts as a legacy font format. This is because the CIDFont and CMap resources for CID-keyed fonts still serve as fundamental source material for building OpenType CJKV fonts, which is a topic that is fully covered later in this chapter. Although end-user use of CID-keyed fonts is effectively deprecated and no longer encouraged, font developers are still given the option to develop CIDFont and CMap resources in order to more easily build OpenType fonts.

The best way to learn how to use CID-keyed fonts in a conforming manner is to read *CID Font Tutorial* (Adobe Systems Technical Note #5643).

* Two hyphens are recommended because it explicitly tells PostScript and other clients what portions represent the CIDFont and CMap resources. For some CID-capable PostScript clone implementations, two hyphens are required.

The CIDFont resource

The CIDFont resource, which is a container for the character descriptions, assigns a unique character ID (CID) for every glyph. This CID is independent of any normal encoding and is simply an enumeration beginning at zero. Table 6-12 provides an example of a few CIDs, along with a graphic representation of the characters that are associated with them (examples taken from the Adobe-Japan1-6 character collection).

Table 6-12. Adobe-Japan1-6 CIDs and their graphic representations—samples

CID	Glyph
0	*.notdef*
1	(proportional-width space)
…	(thousands of CIDs omitted)
7474	堯
7475	槇
7476	遙
7477	瑤
…	(hundreds of CIDs omitted)
8284	凜
8285	熙
…	(thousands of CIDs omitted)
23057	鑾

The CIDFont resource is constructed from two basic parts, each of which is outlined below, along with a brief description:

A header
: Provides global font attributes, such as the font name, number of CIDs, private dictionaries, and so on.

A binary portion
: Contains offsets to subroutines, the subroutines themselves, offsets to charstrings, and the charstrings themselves.

While the header of a CIDFont resource is simply PostScript-like ASCII text, the remainder is binary data.

The CMap resource

The information that associates encoded values with CIDs is defined in the CMap resources. In most cases, or in a perfect world, an encoding range is associated with a CID

range. The following are example lines taken from a standard Adobe-Japan1-6 CMap resource, named simply "H," which specifies ISO-2022-JP encoding:

```
18 begincidrange
… (16 CID ranges omitted)
<7421> <7424> 7474
<7425> <7426> 8284
endcidrange
```

The convention is to use hexadecimal values enclosed in arrow brackets for character codes, and to use decimal values, not enclosed, for CIDs. Carefully note how the encoding ranges are associated with a range of CIDs. The following describes how it works: the encoding range is specified by two encoded values, one for each end of the encoding range—sometimes, noncontiguous encoding or CID ranges make one-to-one mappings necessary. In the preceding case, the two ranges are <74 21> through <74 24> and <74 25> through <74 26>. The CID associated with each range indicates the starting point from which encoded positions are associated with CIDs. For example, the two ranges just listed result in the associations between encoded values and CIDs, as described in Table 6-13.

Table 6-13. Encoded values versus CIDs

Encoded value	CID
<7421>	7474
<7422>	7475
<7423>	7476
<7424>	7477
<7425>	8284
<7426>	8285

Note how only two lines of a CMap resource can be used to associate many code points with their corresponding glyphs, expressed as CIDs. In the case of a complete row of characters, such as the range <30 21> through <30 7E> (94 code points), the following single line can be used:

```
<3021> <307e> 1125
```

For this example, the encoding range <30 21> through <30 7E> is mapped to CIDs 1125 through 1218.

The ordering of the glyphs in character collections generally favors a specific character set, and thus adheres to its ordering. The glyphs of the Adobe-Japan1-6 character collection that correspond to the JIS X 0208:1997 character set are ordered accordingly. This is why it is possible to map large contiguous encoding ranges to correspondingly large contiguous ranges of CIDs. When these same six CIDs are mapped from their corresponding

Unicode code points, the following lines are the result, using the UniJIS2004-UTF32-H CMap resource as the example:

```
6 begincidchar
<000051DC> 8284
<0000582F> 7474
<000069C7> 7475
<00007199> 8285
<00007464> 7477
<00009059> 7476
endcidchar
```

There are two very important things to note in the preceding example CMap lines:

- The character codes are necessarily in ascending order, but because the Unicode order is significantly different than that set forth in the JIS X 0208:1997 character set, the CIDs to which they map are no longer in ascending order. There is absolutely nothing wrong with this.
- The character codes and CIDs are no longer in contiguous ranges, which necessitates the use of the `begincidchar` and `endcidchar` operators that are used to specify single mappings. The `begincidrange` and `endcidrange` operators are appropriate only when character codes and their corresponding CIDs are in contiguous ranges.

The following is an example CMap resource, named UniCNS-UTF32-H, that maps a single Unicode code point, specifically U+4E00 expressed in UTF-32BE encoding, to its corresponding glyph in the Adobe-CNS1-5 character collection, specifically CID+595:

```
%!PS-Adobe-3.0 Resource-CMap
%%DocumentNeededResources: ProcSet (CIDInit)
%%IncludeResource: ProcSet (CIDInit)
%%BeginResource: CMap (UniCNS-UTF32-H)
%%Title: (UniCNS-UTF32-H Adobe CNS1 5)
%%Version: 1.006
%%EndComments
/CIDInit /ProcSet findresource begin
12 dict begin

begincmap

/CIDSystemInfo 3 dict dup begin
  /Registry (Adobe) def
  /Ordering (CNS1) def
  /Supplement 5 def
end def

/CMapName /UniCNS-UTF32-H def
/CMapVersion 1.006 def
/CMapType 1 def

/XUID [1 10 25593] def

/WMode 0 def
```

```
1 begincodespacerange
<00000000> <0010FFFF>
endcodespacerange

1 beginnotdefrange
<00000000> <0000001f> 1
endnotdefrange

1 begincidchar
<00004e00> 595
endcidchar
endcmap
CMapName currentdict /CMap defineresource pop
end
end

%%EndResource
%%EOF
```

I have emboldened the sections that define the encoding space and the character code to CID mappings themselves. Of course, the actual UniCNS-UTF32-H CMap resource includes thousands of mappings. More detailed information about how to build your own CMap resources is available in *Building CMap Files for CID-Keyed Fonts* (Adobe Systems Technical Note #5099).

These sections provided merely a taste of what the CID-keyed fonts offer. This technology was specifically designed so that font developers could make CJKV fonts much smaller, and more easily and efficiently than ever before. CID-keyed fonts are also portable across platforms, at least in the past. For Mac OS, ATM version 3.5J or later supported CID-keyed fonts as did ATM version 3.8 (non-Japanese) or later. For Windows, but not Windows NT or later, ATM version 3.2J (but not version 4.0!) supported CID-keyed fonts. Contemporary versions of these OSes, meaning Mac OS X and Windows (2000, XP, and Vista), support OpenType fonts, which can be built from CID-keyed font sources. Perhaps more importantly, CIDFont and CMap resources represent critical raw materials for building OpenType fonts. Any effort in building CIDFont and CMap resources is not wasted. OpenType, covered later in this chapter, represents a truly cross-platform font format.

The document entitled *Adobe CMap and CIDFont Files Specification* (Adobe Systems Technical Note #5014) describes CID-keyed fonts in more detail and is considered to be the engineering specification needed by font developers. A more gentle introduction is available in the document entitle *CID-Keyed Font Technology Overview* (Adobe Systems Technical Note #5092). Adobe Systems Technical Note #5213 (a PostScript version 2016 supplement) should also be of interest to developers because it more fully describes the various CIDFontTypes. If you are a font developer, I strongly encourage you to request a copy of the CID SDK (*CID-keyed Font Technology Software Development Kit*) from the Adobe Developers Association, which is delivered on a single CD-ROM. The CID SDK includes all the documentation and software necessary to implement CID-keyed font technology, including sample CIDFonts.

sfnt-wrapped CIDFonts—a legacy font format

A past twist to CID-keyed font technology was referred to as sfnt-wrapped CIDFonts. Font geeks and nerds alike are fully aware that TrueType fonts, at least those for Mac OS, resided within what is known as an 'sfnt' (*scalable font*) resource. But, weren't CID-keyed fonts the best thing since sliced bread? Well, sure they were, but….

CID-keyed font technology provides an extremely flexible mechanism for supporting large character sets and multiple encodings, but lacks host-based support such as user-selected and context-sensitive glyph substitution, alternate metrics (such as half-width symbols and punctuation, proportional kana, and even proportional ideographs), and easy vertical substitution. These types of advanced typographic features will be covered in greater detail when we get to Chapter 7.

An sfnt-wrapped CIDFont looks, smells, and behaves as though it were a TrueType—actually, AAT/QuickDraw GX—font, but instead of using TrueType outlines for its glyphs in the 'glyf' table, there is a CIDFont resource there instead in the 'CID' table that was established by Adobe Systems. Basically, the CIDFont file itself becomes one of the many tables within the 'sfnt' resource. Table 6-14 lists the additional 'sfnt' tables that are present in an sfnt-wrapped CIDFont.

Table 6-14. Additional 'sfnt' tables in sfnt-wrapped CIDFonts

Table Tag[a]	Description
ALMX	Alternate metrics
BBOX	Font bounding box
CID[b]	CIDFont resource
COFN	AAT font name for each component in a rearranged font
COSP	Code space for a rearranged font
FNAM	For AAT compatibility
HFMX	Horizontal font metrics
PNAM	Fully qualified PostScript font name
ROTA	Character rotation
SUBS	Subset definition
VFMX	Vertical font metrics
WDTH	Set widths

a. Note that all of these table tags are uppercase. This was intentional because only Apple is allowed to use all-lowercase table tags.

b. Note that this tag, like the others, consists of four characters: the three letters "CID" followed by a single space.

Note that the use of all-lowercase table tags is reserved by Apple, which explains why those table tags listed in Table 6-14 are entirely uppercase. The only exception to this

policy is the 'gasp' table that was defined by Microsoft, which should have included at least one uppercase letter. Gasp!

In addition, sfnt-wrapped CIDFonts typically include the 'bdat', 'bloc', 'cmap', 'feat', 'mort', 'name', 'post', and 'subs' tables, all of which are listed or described later in this chapter in the section entitled "AAT—formerly QuickDraw GX." The TrueType 'cmap' table in sfnt-wrapped CIDFonts functions in a way comparable to the CMap resource for CID-keyed fonts.

The first applications to recognize the advanced typographic features provided in sfnt-wrapped CIDFonts were Adobe Illustrator version 5.5J and Adobe PageMaker version 6.0J—both for Mac OS. Macromedia FreeHand 8.0J and higher also recognized these fonts' advanced typographic features. AAT applications could also recognize and use these fonts, as long as ATM version 3.9 or later was installed and enabled. More information on sfnt-wrapped CIDFonts can be found in *CID-Keyed sfnt Font File Format for the Macintosh* (Adobe Systems Technical Note #5180).

Now, it is clearly a better practice to build OpenType fonts. The good news is that the same CIDFont resource that is used to build an sfnt-wrapped CIDFont can be used to build an equivalent OpenType font. In fact, there are much better tools available for building OpenType fonts than for building sfnt-wrapped CIDFonts. This makes perfect sense con-sidering the obsolete and deprecated nature of sfnt-wrapped CIDFonts.