Building CMap Files for CID-Keyed Fonts

Adobe Developer Support

Technical Note #5099

14 October 1998

Adobe Systems Incorporated

Corporate Headquarters
345 Park Avenue
San Jose, CA 95110
(408) 536-6000 Main Number
(408) 536-9000 Developer Support
Fax: (408) 536-6883

European Engineering Support Group
Adobe Systems Benelux B.V.
P.O. Box 22750
1100 DG Amsterdam
The Netherlands
+31-20-6511 355
Fax: +31-20-6511 313

Adobe Systems Eastern Region
24 New England
Executive Park
Burlington, MA 01803
(617) 273-2120
Fax: (617) 273-2336

Adobe Systems Co., Ltd.
Yebisu Garden Place Tower
4-20-3 Ebisu, Shibuya-ku
Tokyo 150
Japan
+81-3-5423-8169
Fax: +81-3-5423-8204

PN LPS5099

Contents

Building CMap Files for CID-Keyed Fonts

This document is a tutorial for developers (and perhaps end users) who need to create or modify CMap files designed for CJK character collections for CID-keyed fonts, whether or not they were developed by Adobe®. Some familiarity with CID-keyed font technology is assumed. Real utilities, written in Perl, that simplify CMap file development are also provided in this document as an appendix. The principles and tools presented here can be applied to the development of CMap files for any character collection.

Because this document is designed to be a tutorial, it provides practical advice and helpful tips for creating and modifying CMap files. A more complete description and treatment of CMap files, to include the full syntactic specification, can be found in Adobe Tech Note #5014, "Adobe CMap and CIDFont Files Specification." A complete description of the various CJK character collections and their associated CMap files can be found in Adobe Technical Note #5094, "Adobe CJKV Character Collections and CMaps for CID-Keyed Fonts." Both documents are available from the Adobe Developers Association.

The software provided as part of this document, written in Perl, comes with no warranty, written or expressed. Adobe Systems assumes no liability for data lost as a result of executing them. Also note that a Perl interpreter is required to run these tools (Perl is freely available for a variety of platforms).

1 Introduction

If you are unfamiliar with the structure and syntax of a CMap file, this section provides a short example. CMap file syntax is simply a specialized set of PostScript® language commands. For information on PostScript language syntax, see the *PostScript Language Reference Manual, Second Edition* (Addison-Wesley, 1990).

The following is a minimal—but working—Unicode™ (UCS-2 encoding) CMap file named */UniJIS-UCS2-H*. It maps a single Unicode character, U+4E00 (0x4E00), to its respective CID in the Adobe-Japan1-2 character collection, namely 1200.

```
%!PS-Adobe-3.0 Resource-CMap
%%DocumentNeededResources: ProcSet (CIDInit)
%%IncludeResource: ProcSet (CIDInit)
%%BeginResource: CMap (UniJIS-UCS2-H)
%%Title: (UniJIS-UCS2-H Adobe Japan1 2)
%%Version: 1.000
%%EndComments

/CIDInit /ProcSet findresource begin

12 dict begin

begincmap

/CIDSystemInfo 3 dict dup begin
  /Registry (Adobe) def
  /Ordering (Japan1) def
  /Supplement 2 def
end def

/CMapName /UniJIS-UCS2-H def

/CMapVersion 1.000 def
/CMapType 1 def

/XUID [1 10 25356] def

/WMode 0 def

1 begincodespacerange
   <0000> <D7FF>
   <E000> <FFFF>
endcodespacerange

1 beginnotdefrange
<0000> <001f> 1
endnotdefrange

1 begincidrange
<4e00> <4e00> 1200
endcidrange
endcmap
CMapName currentdict /CMap defineresource pop
end
end

%%EndResource
%%EOF
```

Much of a CMap file's structure is static, that is, it does not change across different CMap files.

The sections that follow step you through each portion of a CMap file, using the above minimal CMap file as an example. Special attention will be given to those portions that establish code space ranges and perform the actual character code to CID mappings.

2 Header Information

The CMap header is syntactically just a series of PostScript comments, but their content is important (and parsed by some software).

```
%!PS-Adobe-3.0 Resource-CMap
%%DocumentNeededResources: ProcSet (CIDInit)
%%IncludeResource: ProcSet (CIDInit)
%%BeginResource: CMap (UniJIS-UCS2-H)
%%Title: (UniJIS-UCS2-H Adobe Japan1 2)
%%Version: 1.000
%%EndComments
```

The first three lines are static. The first is the typical comment line ('percent bang') that indicates a PostScript file. The next two lines indicate that the CID procedure set (ProcSet) resource is required, and is in a file called CIDInit.

```
%!PS-Adobe-3.0 Resource-CMap
%%DocumentNeededResources: ProcSet (CIDInit)
%%IncludeResource: ProcSet (CIDInit)
```

The next line indicates the CMap file name in parentheses:

```
%%BeginResource: CMap (UniJIS-UCS2-H)
```

The following line indicates the CMap file name and the /Registry, /Ordering, and /Supplement of the character collection on which this CMap file is based, again in parentheses:

```
%%Title: (UniJIS-UCS2-H Adobe Japan1 2)
```

The following line indicates the version number of the CMap file, indicated by a real number:

```
%%Version: 1.000
```

Note the lack of parentheses (only PostScript-language strings are delimited by parentheses).

CMap files developed by Adobe Systems also include a copyright notice in the header as follows:

```
%%Copyright: ----------------------------------------------------------
%%Copyright: Copyright 1990-1998 Adobe Systems Incorporated.
%%Copyright: All Rights Reserved.
%%Copyright:
%%Copyright: Patents Pending
%%Copyright:
%%Copyright: NOTICE: All information contained herein is the property
%%Copyright: of Adobe Systems Incorporated.
%%Copyright:
%%Copyright: Permission is granted for redistribution of this file
%%Copyright: provided this copyright notice is maintained intact and
%%Copyright: that the contents of this file are not altered in any
%%Copyright: way from its original form.
%%Copyright:
```

```
%%Copyright: PostScript and Display PostScript are trademarks of
%%Copyright: Adobe Systems Incorporated which may be registered in
%%Copyright: certain jurisdictions.
%%Copyright: ----------------------------------------------------------
```

Whether to include similar copyright information is at the discretion of the creator of the CMap file.

The final line of the header simply indicates the end of the header.

```
%%EndComments
```

Vertical-use CMap files use their horizontal counterpart for the majority (and sometimes all) of its mappings, and thus have extra entries in the header to indicate this external dependency. The following is what the header of the */UniJIS-UCS2-V* CMap file would look like:

```
%!PS-Adobe-3.0 Resource-CMap
%%DocumentNeededResources: ProcSet (CIDInit)
%%DocumentNeededResources: CMap (UniJIS-UCS2-H)
%%IncludeResource: ProcSet (CIDInit)
%%IncludeResource: CMap (UniJIS-UCS2-H)
%%BeginResource: CMap (UniJIS-UCS2-V)
%%Title: (UniJIS-UCS2-V Adobe Japan1 2)
%%Version: 1.000
%%EndComments
```

Note the two additional lines (the third and fifth lines from the above segment):

```
%%DocumentNeededResources: CMap (UniJIS-UCS2-H)
%%IncludeResource: CMap (UniJIS-UCS2-H)
```

These lines indicate that this CMap file depends on the existence and contents of the horizontal-use CMap file, namely */UniJIS-UCS2-H*.

3 The CMap Body

The first three lines of the CMap file body are static for all CMap files. The purpose of the first line is to check for the existence of the CID procedure set (ProcSet) as a resource named *CIDInit*:

```
/CIDInit /ProcSet findresource begin
```

If the result returned from the interpreter is true, then the device is enable for CID-keyed fonts. This does not apply to ATM implementations.

The following line creates a new dictionary with 12 entries, the purpose of which is to store the dictionary entries that follow:

```
12 dict begin
```

Storing items in a PostScript dictionary makes them accessible at a later time.

The following line indicates the start of the CMap dictionary:

```
begincmap
```

The following line, which uses the **usecmap** operator, appears in vertical-use CMap files, and indicates that all the mappings from the horizontal-use CMap file are inherited.

```
/UniJIS-UCS2-H usecmap
```

Our example CMap file did not include this line because it is for horizontal use. The */UniJIS-UCS2-V* CMap file, its vertical-use counterpart, includes this line, though.

3.1 Compatibility Information

The following three-item dictionary, named */CIDSystemInfo*, establishes compatibility information for the CMap file, namely the */Registry*, */Ordering*, and */Supplement* information (these three pieces of information can also be referred to as a single string that is a concatenation using hyphens, such as 'Adobe-Japan1-2'). The same dictionary is also present in CIDFont files. These entries must agree with the character collection with which the CMap file is to function (a matching */Supplement* entry is not required for CIDFont and CMap file compatibility).

```
/CIDSystemInfo 3 dict dup begin
  /Registry (Adobe) def
  /Ordering (Japan1) def
  /Supplement 2 def
end def
```

Note how the */Supplement* value is not enclosed in parentheses. This is because it is an integer value, not a string. The example CMap is thus compatible with the Adobe-Japan1-2 character collection, but can be used with CIDFonts that specify a */Supplement* value other than 2.

The following line, namely the dictionary entry */CMapName*, establishes the name of the CMap file:

```
/CMapName /UniJIS-UCS2-H def
```

In this case, the name of the CMap file is */UniJIS-UCS2-H*.

The following line, namely the dictionary entry */CMapVersion*, indicates the version number of the CMap file, expressed as a real number. This should agree with that found in the header section (the '*%%Version:*' line).

```
/CMapVersion 1.000 def
```

The following line indicates the type of CMap file:

```
/CMapType 1 def
```

This opens up the possibility of different CMap file formats in the future. For now, treat this line as the standard way of specifying the CMap type.

3.2 /UIDOffset and /XUID Values

Next, there are the */UIDOffset* and */XUID* dictionary entries. These values are used to cache rendered bitmaps in VM (virtual memory, namely RAM) or on disk for faster imaging during subsequent rendering requests.

The */UIDOffset* value is used to calculate a range of UniqueIDs, and is for backwards compatibility with PostScript Level 1 devices (thus, not always required). This entry was omitted from the example CMap, */UniJIS-UCS2-H*. The following is the */UIDOffset* entry for the */H* CMap from the Adobe-Japan1-2 character collection:

```
/UIDOffset 280 def
```

This simply means that UniqueIDs for the */H* CMap file are consumed starting from an offset of 280 UniqueIDs from a base value set in the CIDFont file itself (the */UIDBase* dictionary entry). The */H* CMap consumes 94 UniqueIDs, but 100 are assigned (meaning that the */UIDOffset* value for the next CMap is 380). If the */UIDBase* of the CIDFont is 2200000, then the */H* CMap will begin assigning UniqueIDs at value 2200280.

The */XUID* value is used in PostScript LanguageLevel 2 and LanguageLevel 3 caching mechanisms, and is implemented as a three-element array. The first element (a number assigned by the Adobe Developers Association) represents the vendor who developed the CMap file, and in this case it is 1 (for Adobe Systems). The second and all subsequent elements in the array are numbers that the vendor assigns to ensure uniqueness, and to signify any meaning they wish. In this example, the second element designates the file type, with a value of 10 (Adobe chose the convention that 10 is used for CMap files, and 11 for CIDFont files). The third element is an integer that is unique across CMap files. The following */XUID* entry comes from the example CMap file:

```
/XUID [1 10 25356] def
```

More detailed information about how to calculate UniqueIDs for the */UIDOffset* dictionary entry can be found in Section 6 of this document, **Calculating UniqueIDs and /UIDOffset Values**.

3.3 Writing Modes

There are two writing modes in PostScript, 0 and 1. Writing mode 0 is horizontal, and writing mode 1 is vertical. These should agree with the last character in the CMap file name, namely 0 with '-H', and 1 with '-V'.

```
/WMode 0 def
```

The example CMap file, */UniJIS-UCS2-H*, uses the horizontal writing mode. The corresponding vertical-use CMap file, */UniJIS-UCS2-V*, will use the following line instead:

```
/WMode 1 def
```

The origin of writing mode 0 begins at coordinate 0,0, and the pen direction is left to right. The origin of writing mode 1 begins at coordinate 0,0.

Note 1 *The 0,0 coordinate is at a vector of 500,880 from the origin of writing mode 0 (at the top center of a full-width 1000-unit character). The writing direction is top to bottom.*

3.4 Code Space Range

The lines of CMap code described in this section establish the code space range for the supported encoding.

Note 2 *It is extremely important to understand that the theoretical code space range should be used here, not just the code space range that contains defined characters. This is a typical mistake often made by developers of CMap files! But, some encodings that have disjoint encoding blocks, such as Shift-JIS (Japanese), Big Five (Traditional Chinese), Unified Hangul Code (Korean), or Johab (Korean), are best expressed as a contiguous block.*

You will learn that a rich mixture of encodings can be specified by a CMap file. One- to four-byte representations can be expressed. A current limitation of CID compatibility mode (not available on PostScript devices Version 2014 and earlier) is two-byte representations, but native mode (available on PostScript devices Version 2015 or later) can handle up to four-byte representations. Windows® - and Macintosh® -based PCs typically use a mixture of one- and two-byte encodings. Unix systems use a mixture of one- through four-byte representations. Unicode (UCS-2) is strictly two-byte.

Code space ranges are expressed as hexadecimal codes that represent the beginning and ending code point of each contiguous encoding block. The uppercase hexadecimal digits A through F (as opposed to the lowercase hexadecimal digits a through f) are used—as a convention rather than a rule—to distinguish the code space range definition from character code to CID mappings, but this is not a requirement. The following is an example code space range definition:

```
1 begincodespacerange
  <0000> <FFFF>
endcodespacerange
```

In this case, the code space range is strictly two-byte, and is 0x0000 through 0xFFFF (65,536 code points). This is UCS-2 encoding for ISO 10646-1:1993.

The integer immediately before the **begincodespacerange** operator must indicate and agree with the total number of encoding range lines. Furthermore, all encoding ranges are required to be in increasing byte-value order.

Other code space ranges include the following examples. These can be used as study material as they may be useful for developing other CMap files.

The first example is for two-byte 7-bit ISO-2022 (applicable to JIS X 0208:1997, JIS X 0212-1990, GB 2312-80, KS X 1001:1992, and all planes of CNS 11643-1992):

```
1 begincodespacerange
  <2121> <7E7E>
endcodespacerange
```

This provides the usual 8,836 code points in a 94×94 matrix.

The next example is for Shift-JIS (JIS X 0201-1997 and JIS X 0208:1997):

```
4 begincodespacerange
  <00> <80>           % JIS X 0201-1997 Roman
  <8140> <9FFC>       % The first two-byte block
  <A0> <DF>           % Half-width katakana
  <E040> <FCFC>       % The second two-byte block
endcodespacerange
```

Note how the user-defined range (0xF040 through 0xFCFC) is included as part of the second two-byte block. While the second-byte range of Shift-JIS is disjoint (0x40 through 0xFC, but not 0x7F), it is expressed here as a contiguous range to simplify the character code to CID mappings that are assigned later.

Another example is for EUC-JP (JIS X 0201-1997, JIS X 0208:1997, and JIS X 0212-1990):

```
4 begincodespacerange
  <00> <80>           % EUC code set 0 (JIS X 0201-1997 Roman)
  <8EA0> <8EDF>       % EUC code set 2 (half-width katakana)
  <8FA1A1> <8FFEFE>   % EUC code set 3 (JIS X 0212-1990)
  <A1A1> <FEFE>       % EUC code set 1 (JIS X 0208:1997)
endcodespacerange
```

Note the use of one-, two-, and three-byte character codes.

The next example is for EUC-CN (GB 1988-89 and GB 2312-80):

```
2 begincodespacerange
  <00> <80>           % EUC code set 0 (GB 1988-89 Roman)
  <A1A1> <FEFE>       % EUC code set 1 (GB 2312-80)
endcodespacerange
```

The above also applies to EUC-KR encoding, (KS X 1001:1992 and KS X

1003:1993).

The next example is for EUC-TW (ASCII and the complete CNS 11643-1992):

```
18 begincodespacerange
   <00> <80>                % EUC code set 0 (ASCII)
   <8EA1A1A1> <8EA1FEFE>    % EUC code set 2 (Plane 1)
   <8EA2A1A1> <8EA2FEFE>    % EUC code set 2 (Plane 2)
   <8EA3A1A1> <8EA3FEFE>    % EUC code set 2 (Plane 3)
   <8EA4A1A1> <8EA4FEFE>    % EUC code set 2 (Plane 4)
   <8EA5A1A1> <8EA5FEFE>    % EUC code set 2 (Plane 5)
   <8EA6A1A1> <8EA6FEFE>    % EUC code set 2 (Plane 6)
   <8EA7A1A1> <8EA7FEFE>    % EUC code set 2 (Plane 7)
   <8EA8A1A1> <8EA8FEFE>    % EUC code set 2 (Plane 8)
   <8EA9A1A1> <8EA9FEFE>    % EUC code set 2 (Plane 9)
   <8EAAA1A1> <8EAAFEFE>    % EUC code set 2 (Plane 10)
   <8EABA1A1> <8EABFEFE>    % EUC code set 2 (Plane 11)
   <8EACA1A1> <8EACFEFE>    % EUC code set 2 (Plane 12)
   <8EADA1A1> <8EADFEFE>    % EUC code set 2 (Plane 13)
   <8EAEA1A1> <8EAEFEFE>    % EUC code set 2 (Plane 14)
   <8EAFA1A1> <8EAFFEFE>    % EUC code set 2 (Plane 15)
   <8EB0A1A1> <8EB0FEFE>    % EUC code set 2 (Plane 16)
   <A1A1> <FEFE>            % EUC code set 1 (Plane 1)
endcodespacerange
```

Note how CNS 11643-1992 Plane 1 is (redundantly) encoded in both EUC-TW code sets 1 and 2, and how a mixture of one-, two-, and four-byte character codes is used. But some may wonder why it is not possible to express the second through seventeenth lines as a single line as follows:

```
<8EA1A1A1> <8EB0FEFE>      % EUC code set 2 (CNS 11643-1992)
```

It is a valid question and concern. The reason is because of UniqueID assignment (explained in Section 6), which is done on the basis of an encoding row. The least significant byte can have up to 256 values. A single UniqueID can handle up to 256 code points. This means that the number of UniqueIDs required for each code space range line is determined by the number of unique values in the second byte from the right. In the above case it is 0xA1 through 0xFE, which requires 94 UniqueIDs.

The next example is for IBM® DBCS-Host encoding (applicable for Chinese, Japanese, and Korean):

```
2 begincodespacerange
   <4040> <4040>            % Full-width space character
   <4141> <FEFE>            % Two-byte characters
endcodespacerange
```

Note how the full-width space character, a single code point, is handled.

The next example is for Big Five encoding (Traditional Chinese, and equivalent to CNS 11643-1992 Planes 1 and 2):

```
2 begincodespacerange
   <00> <80>                % ASCII
```

```
  <A140> <FEFE>            % Two-byte characters
endcodespacerange
```

While Big Five encoding has two disjoint two-byte encoding blocks, namely 0xA140 through 0xFE7E and 0xA1A1 through 0xFEFE, the encoding block specified here joins them by including the intervening encoding block, namely 0xA17F through 0xFEA0. This is done to simplify the character code to CID mappings that are described later.

The next example is for Johab encoding, which includes all 11,172 hangul (KS X 1001:1992 plus KS X 1003:1993):

```
4 begincodespacerange
  <00> <80>                % KS X 1003: 1993 Roman
  <8441> <D3FE>            % Hangul
  <D831> <DEFE>            % Symbols
  <E031> <F9FE>            % Hanja
endcodespacerange
```

The next example is for Unified Hangul Code (UHC) encoding, supported in Microsoft® Windows 95 Korean. It represents the same character set as Johab encoding, but its encoding scheme is different:

```
2 begincodespacerange
  <00> <80>                % KS X 1003: 1993 Roman
  <8141> <FEFE>            % Two-byte characters
endcodespacerange
```

The final example shows that one-byte CMaps can also be defined:

```
1 begincodespacerange
  <00> <FF>                % ASCII or EBCDIC
endcodespacerange
```

This code space range is applicable for both ASCII- or EBCDIC-style encodings.

Note that the code space range section is not found in vertical-use CMap files. Vertical-use CMap files instead inherit the code space range from the corresponding horizontal-use CMap file, indicated by the **usecmap** operator.

Note also that it is possible to set the largest possible code space range, such as the following for 7-bit ISO-2022 (normally 0x2121 through 0x7E7E):

```
1 begincodespacerange
  <0000> <FFFF>
endcodespacerange
```

But this method wastes a significant amount of UniqueID space (over 150 UniqueIDs), and permits many characters outside the valid 7-bit ISO-2022 range. Joining non-contiguous encoding blocks that share the same first-byte values, such as we saw with Shift-JIS and Big Five encodings, does not waste any UniqueIDs.

3.5 .Notdef Range

If a special mapping is needed for certain unused characters or character ranges, such as the ASCII control character range 0x00 through 0x1F, a specific CID can be assigned for this purpose. By default, unused code points are assigned CID 0. This is not always desired because the width of CID 0 is typically 1000 units (one em). The following is an example:

```
1 beginnotdefrange
<0000> <001f> 1
endnotdefrange
```

In this case, the Unicode (UCS-2) control character range, U+0000 through U+001F, is assigned CID 1, which is a proportional-width space (in the Adobe-Japan1-2 character collection). A half-width space can be assigned for Shift-JIS or EUC-JP encoding as follows (again, this CID applies to the Adobe-Japan1-2 character collection):

```
1 beginnotdefrange
<00> <1f> 231
endnotdefrange
```

If a CMap uses the one-byte ASCII set, it may be desirable to specify either a half- or proportional-width space, depending on the characteristics of the CMap file. For example, the Macintosh CMap file for Japanese, */90pv-RKSJ-H*, uses a proportional-width ASCII set, and thus specify CID 1 (proportional-width space) for the control character range as follows:

```
1 beginnotdefrange
<00> <1f> 1
endnotdefrange
```

CMap files that use a half-width ASCII (or equivalent) set would use a half-width space for this section (CID 231 for the Adobe-Japan1-2 character collection).

3.6 Character Code to CID Mapping Basics

This section represents the most critical part of the CMap file because it establishes all of the character code to CID mappings. The following minimal mapping handles a single character code:

```
1 begincidrange
<4e00> <4e00> 1200
endcidrange
```

The character code 0x4E00 simply maps to CID 1200. The number that appears before the **begincidrange** operator indicates the number of mapping lines. The **endcidrange** operator terminates a block of mappings.

If a CMap file contains more than 100 mapping entries, they must be

segmented into 100-line blocks as follows:

```
100 begincidrange
<2121> <217e>  633
<2221> <222e>  727
<223a> <2241>  741
<224a> <2250>  749
```

<...92 mappings omitted...>

```
<6021> <607e> 5594
<6121> <617e> 5688
<6221> <627e> 5782
<6321> <637e> 5876
endcidrange

18 begincidrange
<6421> <647e> 5970
<6521> <657e> 6064
<6621> <667e> 6158
<6721> <677e> 6252
```

<...10 mappings omitted...>

```
<7221> <727e> 7286
<7321> <737e> 7380
<7421> <7424> 7474
<7425> <7426> 8284
endcidrange
```

In the above example, the 118 mapping lines used in the */H* CMap file for the Adobe-Japan1-2 character collection are separated into a block of 100 and a block of 18.

The three entries of each mapping line comprise the begin character code (in hexadecimal), the end character code (in hexadecimal), and the begin CID (in decimal). If the line maps a single character code, both the begin and end character codes are identical.

The following mapping handles the 94 characters that make up the first row of kanji from the */H* CMap from the Adobe-Japan1-2 character collection—this is 7-bit ISO-2022 encoding:

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

This is equivalent to writing:

```
<3021> <3021> 1125
<3022> <3022> 1126
<3023> <3023> 1127
<3024> <3024> 1128
```

<...86 mappings omitted...>

```
<307b> <307b> 1215
```

```
<307c> <307c> 1216
<307d> <307d> 1217
<307e> <307e> 1218
```

but is much more efficient. In principle, if both character code *and* CID ranges are contiguous, they can be collapsed into a single mapping line. This results in a shorter (smaller) CMap file, and increased efficiency.

The corresponding 94 character codes in the JIS C 6226-1978 CMap for the Adobe-Japan1-2 character collection, */78-H*, does not have completely contiguous CIDs (although the character codes are contiguous). The following lines map the same set of 94 character codes as shown above, namely 0x3021 through 0x307E:

```
<3021> <3021> 1125
<3022> <3022> 7633 % JIS C 6226-1978 kanji
<3023> <3028> 1127
<3029> <3029> 8266 % JIS C 6226-1978 kanji
<302a> <3031> 1134
<3032> <3032> 7961 % JIS C 6226-1978 kanji
<3033> <3033> 7330 % JIS C 6226-1978 kanji
<3034> <303a> 1144
<303b> <303b> 7634 % JIS C 6226-1978 kanji
<303c> <306d> 1152
<306e> <306e> 7635 % JIS C 6226-1978 kanji
<306f> <3072> 1203
<3073> <3073> 7636 % JIS C 6226-1978 kanji
<3074> <307b> 1208
<307c> <307c> 7637 % JIS C 6226-1978 kanji
<307d> <307e> 1217
```

Note how non-contiguity of CIDs results in significantly more mapping lines (16 lines for the */78-H* CMap file, compared to only 1 line for the */H* CMap file).

The order of mapping lines must also be in increasing byte-value order, meaning that the following order is not acceptable, and results in an invalid CMap file:

```
<3121> <317e> 1219
<3021> <307e> 1125
```

It must instead be as follows:

```
<3021> <307e> 1125
<3121> <317e> 1219
```

While the character codes must be in increasing byte-value order, there is no such stipulation for the order of CIDs.

In some cases, a CMap file will exist, but contain no character code to CID mapping lines. The vertical-use CMap files for the Adobe-Japan2-0 character collection, */Hojo-V* and */Hojo-EUC-V*, are such examples. Their sole purpose is to specify writing mode 1 (vertical), but no character code to CID mappings

are necessary (all the character code to CID mappings are inherited from their corresponding horizontal-use CMap files (see Section 3.3).

3.7 Building Character Code to CID Mappings

The previous section described the basics of character code to CID mappings. This section contains practical advice about how to build up these mappings from scratch using other tables or sources. Proven tools for simplifying CMap development are also presented. These tools can be used as-is, or customized for other purposes.

First, it is very useful to have a control file that maps each CID to a common name (or encoding). The following example is for the complete set of 6,355 kanji in JIS X 0208:1997 (based on the Adobe-Japan1-2 character collection):

```
1125 3021
1126 3022
1127 3023
1128 3024

<...6,347 contiguous CIDs and character codes omitted...>

7476 7423
7477 7424
8284 7425
8285 7426
```

The first column represents the CID (in decimal), and the second column represents the corresponding Japanese 7-bit ISO-2022 code (in hexadecimal). Such a single CID to character code mapping alone is not very useful in building new CMap files (because a 7-bit ISO-2022 CMap file, */H*, already exists), but together with other mapping tables, can be used as a powerful development aid.

An example of this power can be shown when trying to build a UCS-2 or DBCS-Host CMap file. The Unicode Consortium has developed and provides (at URL *ftp://ftp.unicode.org/Public/MAPPINGS/*) a number of useful mapping tables. One such table maps Japanese 7-bit ISO-2022 to Unicode (UCS-2 encoding). IBM can also provide similar mappings tables for their DBCS-Host encoding. Such mapping tables provide the following type of information (Row 0x45 of Japanese DBCS-Host encoding, which has 190 characters, being the example here):

```
4541 306c
4542 4673
4543 3b30
4544 3b4d

<...182 character code mappings omitted...>

45fb 314a
```

```
45fc 346f
45fd 364c
45fe 423f
```

The first column is the DBCS-Host code (in hexadecimal) and the second column is the Japanese 7-bit ISO-2022 code (also in hexadecimal). The DBCS-Host encoding range is 0x4541 through 0x45FE. It is important that such mapping tables are sorted in increasing ASCII order.

Now, with some powerful text processing tools, such as Awk or Perl, one can take tables with thousands of such mappings, and create character code to CID mappings from them. Associative arrays serve as an ideal data structure for loading and retrieving such mapping information. Perl is an ideal programming language for manipulating CMap files. It is not only free, and available for numerous platforms, but is extremely powerful. Its primary strength lies in its regular expression and parsing capabilities.

The following short Perl program is designed to create the raw (uncollapsed, namely one character code to CID mapping per line) character code to CID mappings:

```
#!/usr/local/bin/perl

open(CID,"<cid.tbl") || die "Error opening cid.tbl file.\n";

$count = 0;

while (defined ($line = <CID>) ) {
  chop($line);
  ($cid,$code) = split(/\s+/,$line);
  $code =~ tr/A-F/a-f/;
  $table{$code} = $cid;
  $count++;
}
close(CID);

print STDERR "Processed $count CID mappings...\n";
$count = 0;

while (defined ($line = <STDIN>) ) {
  chop($line);
$line =~ tr/A-F/a-f/;
  ($new,$code) = split(/\s+/,$line);
  cid = $table{$code};
  print STDOUT "<$new> <$new> $cid\n";
  $count++;
}

print STDERR "Created $count raw character code to CID mappings.\n";
```

The CID-indexed file (each line indicates the CID plus the equivalent Japanese 7-bit ISO-2022 code) is first read in by explicitly opening the file named "cid.tbl". The other mapping table (each line indicates the target encoding plus

the equivalent 7-bit ISO-2022 code) is read in as standard input (STDIN). The result, which conforms to CMap file syntax, is then written to standard output (STDOUT).

If the character code mapping table has multiple fields (that is, more than two), the following line of code:

```
($new,$code) = split(/\s+/,$line);
```

can be replaced with something like:

```
($new,$code) = (split(/\s+/,$line))[0,4];
```

to extract only the two desired fields. The first field, array index 0, is assigned to the variable $new, and the fifth field, array index 4, is assigned to the variable $code. Perl arrays, like in C, begin with index 0. Other digits can be used to extract specific fields.

When using the above two mappings tables with the Perl utility, one can create the following 190 lines of CMap code:

```
<4541> <4541> 1200
<4542> <4542> 3275
<4543> <4543> 2174
<4544> <4544> 2203
```

<...182 mappings omitted...>

```
<45fb> <45fb> 1260
<45fc> <45fc> 1579
<45fd> <45fd> 1732
<45fe> <45fe> 2847
```

Note how the character codes are in increasing byte-value order, and that neither the character codes nor the CIDs are contiguous, resulting in one character code mapping per line. The order of the kanji characters in the Adobe-Japan1-2 character collection favors native Japanese encodings, such as 7-bit ISO-2022, Shift-JIS, and EUC-JP. Encodings such as UCS-2 or DBCS-Host do not follow the same ordering (that is, the sequence in which characters appear), thus usually require many more mapping lines.

When building complex CMap files with many non-contiguous CIDs and character codes, it is easiest to initially use single character code to CID mappings (like the output of the above Perl program), then to later collapse those that are contiguous. A sample Perl program to collapse a CMap file is shown in Appendix A.

When processing these 190 DBCS-Host mapping lines with this second utility, they are reduced by only one to 189. The following two (input) mapping lines have both contiguous CIDs and character codes:

```
<45fa> <45fa> 1259
```

```
<45fb> <45fb> 1260
```

These two lines are subsequently reduced to the following single character code to CID mapping:

```
<45fa> <45fb> 1259
```

This slight reduction in mapping lines is typical when dealing with DBCS-Host and UCS-2 encodings. When building the entire Japanese DBCS-Host CMap file based on the Adobe-Japan1-2 character collection, the approximately 7,200 character code to CID mappings are reduced to just under 5,000 mapping lines.

3.8 CMap Trailer

The following static lines terminate the CMap file by closing the dictionary and defining the CMap file as a resource:

```
endcmap
CMapName currentdict /CMap defineresource pop
end
end

%%EndResource
%%EOF
```

4 The Identity CMap

The Identity CMap is a special type of CMap file whose purpose is to allow access to all CIDs of a character collection through the CID values themselves. A normal CMap file allows access to a subset of the character collection by using conventional character codes. The name of an Identity CMap is the concatenation of the */Registry*, */Ordering*, and */Supplement* values, using a single hyphen as a joining element. So, the Identity CMap for the Adobe-Japan1-2 character collection is named as follows:

```
/Adobe-Japan1-2
```

That is, the same name as the character collection itself. This naming convention makes its content explicit.

One use for an Identity CMap is for CIDFont proofing purposes (this is why they were originally developed and how they are used at Adobe Systems). It is best to proof a CIDFont at the CID level rather than by character code. The CMap files, at least the ones developed by Adobe Systems, are fully tested. They do not change from CIDFont to CIDFont—they are common across all CIDFont files of a given character collection.

Adobe Systems creates an Identity CMap for every character collection and for every supplement thereof. We are currently exploring other uses for Identity

CMap files.

4.1 Unique Entries and Values

An Identity CMap does have one special dictionary entry not found in other CMap files, namely a */CIDCount* value. An example is as follows (taken from the Identity CMap for the Adobe-Japan1-2 character collection):

```
/CIDCount 8720 def
```

This entry explicitly indicates the number of CIDs in a given character collection.

The following are lines extracted from the Identity CMap for the Adobe-Japan1-2 character collection (two from the CMap header, and one from the CMap body):

```
%%BeginResource: CMap (Identity)
%%Title: (Identity Adobe Japan1 2)
/CMapName /Adobe-Japan1-2 def
```

Note how the string 'Identity' is used for the CMap name in the header, but not in the body.

4.2 Identity CMap Code Space Range

The code space range for Identity CMaps begins with 0x0000 and has an upper limit of 0xFFFF (this handles character collections with up to 65,536 CIDs). But, instead of using the following code space range:

```
1 begincodespacerange
  <0000> <FFFF>
endcodespacerange
```

The last occupied row is used as the upper limit instead, as follows (for the Adobe-Japan1-2 character collection, which has 8,720 CIDs):

```
1 begincodespacerange
  <0000> <22FF>
endcodespacerange
```

The hexadecimal equivalent of 8720 (0 through 8719) is 0x220F, and the upper limit of 0x22FF as expressed above covers this. Exactly what upper limit is set depends on the number of CIDs in the character collection. The reason 0x0000 through 0xFFFF is not used is because the size of the code space range can affect memory usage and UniqueID assignment.

Because an Identity CMap's character codes and CIDs are always contiguous, it is simply a matter of mapping 256-character-code chunks (second-byte values ranging from 0x00 through 0xFF) to CIDs, as follows:

```
35 begincidrange
<0000> <00ff> 0
<0100> <01ff> 256
<0200> <02ff> 512
<0300> <03ff> 768
```

<...27 mappings omitted...>

```
<1f00> <1fff> 7936
<2000> <20ff> 8192
<2100> <21ff> 8448
<2200> <220f> 8704
endcidrange
```

4.3 Using Identity CMaps

The following example makes use of Identity CMaps as arguments to the **findfont** or **selectfont** (PostScript Level 2 only) procedures:

```
/HeiseiMin-W3--Adobe-Japan1-2 findfont 12 scalefont setfont
/HeiseiMin-W3--Adobe-Japan1-2 12 selectfont
```

This usage and behavior is no different from that of conventional CMap files.

Subsequent calls using the **show** operator take the following form, for example, if CIDs 1500 and 1501 are requested for rendering:

```
<05dc 05dd> show
```

Values inside of < > are interpreted as two-digit hexadecimal values — spaces are ignored. PostScript looping structures can then be used to easily render all characters in a CIDFont.

4.4 Example Identity CMap

The following is a complete Identity CMap taken from the Adobe-Japan1-2 character collection. It may be used as a template for creating other Identity CMaps.

```
%!PS-Adobe-3.0 Resource-CMap
%%DocumentNeededResources: ProcSet (CIDInit)
%%IncludeResource: ProcSet (CIDInit)
%%BeginResource: CMap (Identity)
%%Title: (Identity Adobe Japan1 2)
%%Version: 1.000
%%Copyright: -----------------------------------------------------------
%%Copyright: Copyright 1990-1997 Adobe Systems Incorporated.
%%Copyright: All Rights Reserved.
%%Copyright:
%%Copyright: Patents Pending
%%Copyright:
%%Copyright: NOTICE: All information contained herein is the property
%%Copyright: of Adobe Systems Incorporated.
```

```
%%Copyright:
%%Copyright: Permission is granted for redistribution of this file
%%Copyright: provided this copyright notice is maintained intact and
%%Copyright: that the contents of this file are not altered in any
%%Copyright: way from its original form.
%%Copyright:
%%Copyright: PostScript and Display PostScript are trademarks of
%%Copyright: Adobe Systems Incorporated which may be registered in
%%Copyright: certain jurisdictions.
%%Copyright: ----------------------------------------------------------
%%EndComments

/CIDInit /ProcSet findresource begin

12 dict begin

begincmap

/CIDSystemInfo 3 dict dup begin
  /Registry (Adobe) def
  /Ordering (Japan1) def
  /Supplement 2 def
end def

/CMapName /Adobe-Japan1-2 def
/CMapVersion 1.000 def
/CMapType 1 def

/UIDOffset 480 def
/XUID [1 10 25358] def

/WMode 0 def

/CIDCount 8720 def

1 begincodespacerange
  <0000> <22FF>
endcodespacerange

35 begincidrange
<0000> <00ff>    0
<0100> <01ff>  256
<0200> <02ff>  512
<0300> <03ff>  768
<0400> <04ff> 1024
<0500> <05ff> 1280
<0600> <06ff> 1536
<0700> <07ff> 1792
<0800> <08ff> 2048
<0900> <09ff> 2304
<0a00> <0aff> 2560
<0b00> <0bff> 2816
<0c00> <0cff> 3072
<0d00> <0dff> 3328
<0e00> <0eff> 3584
<0f00> <0fff> 3840
<1000> <10ff> 4096
<1100> <11ff> 4352
```

```
<1200> <12ff> 4608
<1300> <13ff> 4864
<1400> <14ff> 5120
<1500> <15ff> 5376
<1600> <16ff> 5632
<1700> <17ff> 5888
<1800> <18ff> 6144
<1900> <19ff> 6400
<1a00> <1aff> 6656
<1b00> <1bff> 6912
<1c00> <1cff> 7168
<1d00> <1dff> 7424
<1e00> <1eff> 7680
<1f00> <1fff> 7936
<2000> <20ff> 8192
<2100> <21ff> 8448
<2200> <220f> 8704
endcidrange
endcmap
CMapName currentdict /CMap defineresource pop
end
end

%%EndResource
%%EOF
```

5 CMap Naming Conventions

CMap file names established by Adobe Systems are composed of up to three parts, each separated by a hyphen. One indicates character set, another indicates encoding, and the last indicates writing direction. Although CMap files can use arbitrary names, these conventions make it easier to identify characteristics of the character set and encoding they support.

The following are character set field values based on existing or prototype Adobe CMap files:

Character Set	Description
78	JIS C 6226-1978
83pv	Macintosh KanjiTalk® 6 JIS X 0208-1983 extensions
90ms	Microsoft Windows 3.1J JIS X 0208:1997 extensions
90pv	Macintosh KanjiTalk7 JIS X 0208:1997 extensions
90sk	IBM JIS X 0208:1997 extensions
Add	Fujitsu® JIS X 0208:1997 extensions
CNS	CNS 11643-1992
CNS1	CNS 11643-1992 Plane 1

Character Set	Description
CNS2	CNS 11643-1992 Plane 2
CNS3	CNS 11643-1992 Plane 3
ETen	ETen Big Five extensions
Ext	NEC® JIS C 6226-1978 extensions
GB	GB 2312-80
GBT	GB/T 12345-90
GBK	GBK
KSC	KS X 1001:1992
KSCms	Microsoft® KS X 1001:1992 extensions
NWP	NEC Word Processor (obsolete)
UniJIS	Adobe-Japan1 subset of ISO 10646-1:1993
UniHojo	Adobe-Japan2 subset of ISO 10646-1:1993
UniGB	Adobe-GB1 subset of ISO 10646-1:1993
UniKS	Adobe-Korea1 subset of ISO 10646-1:1993
UniCNS	Adobe-CNS1 subset of ISO 10646-1:1993

In the case of Adobe-Japan1-2, if the character set is generic JIS X 0208:1997 (or JIS X 0208-1983), the character set value is omitted.

The following are encoding field values based on existing and prototype Adobe CMap files:

Encoding	Description
B5	Big Five encoding
EUC	Extended UNIX Code
Host	IBM DBCS-Host encoding
Johab	Johab encoding
RKSJ	Shift-JIS encoding (same as IBM Japanese DBCS-PC)
UCS2	ISO 10646-1:1993 UCS-2 encoding
UHC	Unified Hangul Code (Microsoft Windows 95 Korean)
UTF8	ISO 10646-1:1993 UTF-8 encoding

If the encoding is 7-bit ISO-2022, the encoding value is omitted. Also, the actual encoding ranges used for EUC and RKSJ may differ depending on the

character collection or character set value.

The writing mode must be either H or V, and should agree with the */WMode* dictionary entry in the CMap file itself. A '-V' CMap should contain only those character code to CID mappings that are specific to the vertical writing mode.

The CMap file names established by Adobe Systems are unique in that they contain no */Registry* string because they are intended to be used by all font developers who wish to use them with their CIDFonts. Developers other than Adobe Systems are required to register a short */Registry* string to be prefixed to all CMap file names, regardless of the character collection on which they are based. Registry strings can be registered through the Adobe Developers Association. The purpose of this policy is to avoid duplication of CMap file names.

6 Calculating UniqueIDs and /UIDOffset Values

There are two stages in calculating UniqueIDs for the */UIDOffset* dictionary entry. The first is to determine how many UniqueIDs are required for each CMap file. The next is to add up the UniqueID values for all CMap files of a character collection—this determines how many UniqueIDs to assign to each font based on the character collection.

Each CMap file is assigned an offset value from a base value of zero. The */UIDOffset* for the first CMap file of a character collection is zero (0).

Adobe CMap files adhere to the following figures when determining UniqueID assignment:

Encoding	Assigned	Consumed
One-byte	2	1
7-bit ISO-2022	100	94
Big Five	100	95 or 96, depending on CMap
EUC-JP[a]	100	96
EUC-JP[b]	100	94
EUC-JP[c]	200	190
EUC-CN	100	95 or 96, depending on CMap
EUC-KR	100	95 or 96, depending on CMap
EUC-TW[d]	300	283
EUC-TW[e]	1599	1600
DBCS-Host	200	190

Encoding	Assigned	Consumed
Johab	120	115
RKSJ	70	62 or 63, depending on CMap
UHC	130	127
Vertical	10	varies, depending on CMap
GBK	130	127

a. EUC code sets 0, 1, and 2 (for Adobe-Japan1-1).
b. EUC code set 3 only (for Adobe-Japan2-0).
c. EUC code sets 0 through 3 (for Adobe-Japan1-2 plus Adobe-Japan2-0).
d. EUC code sets 0, 1, and 2 (for Adobe-CNS1-0, namely CNS 11643-1992 Planes 1 and 2).
e. EUC code sets 0, 1, and 2 (CNS 11643-1992 Planes 1 through 16).

Note how a small buffer of UniqueIDs is used to ensure that no UniqueID collisions take place, which could result in font caching and subsequent rendering problems.

If the desired encoding is not found in the above listing, you can calculate the number of required UniqueIDs yourself using some basic guidelines:

- The code space range specification for the CMap file (see Section 3.4) is used for the calculation.

- A single UniqueID can handle up to 256 code points, but is limited to the code space determined by the least significant (or final) byte.

- The number of UniqueIDs required for each code space range line is determined by the number of unique values in the second byte from the right.

Now look at the following code space range specification as an example (Big Five encoding from Adobe-CNS1-1):

```
2 begincodespacerange
  <00> <80>                % ASCII
  <A140> <FEFE>            % Two-byte characters
endcodespacerange
```

The first code space range line specifies one-byte encoding, thus requires (and consumes) one UniqueID (129 characters). Each UniqueID in the second code space range line will be used to express 191 characters (0x40 through 0xFE), and 94 UniqueIDs are required for the second code space range line (0xA1 through 0xFE). This means that a total of 95 UniqueIDs are required for this code space range specification, but this figure is then rounded to 100.

The following list provides a complete collection of */UIDOffset* values for all

published and planned CJK character collections for CID-keyed fonts (also included are the UniqueID ranges in parentheses and the number of assigned UniqueIDs):

Adobe-Japan1-0: 869 UniqueIDs		
83pv-RKSJ-H	0 (0–69)	70
Ext-RKSJ-H	70 (70–139)	70
Add-RKSJ-H	140 (140–209)	70
RKSJ-H	210 (210–279)	70
H	280 (280–379)	100
Ext-H	380 (380–479)	100
Adobe-Japan1-0	480 (480–579)	100
Add-H	580 (580–679)	100
EUC-H	680 (680–779)	100
Add-RKSJ-V	780 (780–789)	10
Add-V	790 (790–799)	10
EUC-V	800 (800–809)	10
Ext-RKSJ-V	810 (810–819)	10
Ext-V	820 (820–829)	10
unused	830 (830–839)	10
RKSJ-V	840 (840–849)	10
V	850 (850–859)	10
Hankaku	860 (860–861)	2
Hiragana	862 (862–863)	2
Katakana	864 (864–865)	2
Roman	866 (866–867)	2
WP-Symbol	868 (868)	1

Adobe-Japan1-1: 1065 UniqueIDs		
90pv-RKSJ-H	870 (870–939)	70
90pv-RKSJ-V	940 (940–949)	10
Adobe-Japan1-1	*1070 (1030–1064)*	*35*

Adobe-Japan1-2: 1100 UniqueIDs		
90ms-RKSJ-H	950 (950–1019)	70
90ms-RKSJ-V	1020 (1020–1029)	10
Adobe-Japan1-2	*1065 (1065–1099)*	*35*

Adobe-Japan2-0: 270 UniqueIDs		
Hojo-EUC-H	0 (0–99)	100
Hojo-H	100 (100–199)	100
Hojo-EUC-V	200 (200–209)	10
Hojo-V	210 (210–219)	10
Adobe-Japan2-0	220 (220–269)	50

Adobe-GB1-0: 380 UniqueIDs		
GBpc-EUC-H	0 (0–99)	100
GB-EUC-H	100 (100–199)	100
GB-H	200 (200–299)	100
GBpc-EUC-V	300 (300–309)	10
GB-EUC-V	310 (310–319)	10
GB-V	320 (320–329)	10
Adobe-GB1-0	330 (330–379)	50

Adobe-GB1-1: 750 UniqueIDs		
GBTpc-EUC-H	380 (380–479)	100
GBT-EUC-H	480 (480–579)	100
GBT-H	580 (580–679)	100
GBTpc-EUC-V	680 (680–689)	10
GBT-EUC-V	690 (690–699)	10
GBT-V	700 (700–709)	10
Adobe-GB1-1	*710 (710–749)*	*40*

Adobe-GB1-2: 840 UniqueIDs		
Adobe-GB1-2	*750 (750–839)*	*90*

Adobe-Korea1-0: 330 UniqueIDs		
KSCpc-EUC-H	0 (0–99)	100
KSC-EUC-H	100 (100–199)	100
KSC-H	200 (200–299)	100
KSCpc-EUC-V	300 (300–309)	10
KSC-EUC-V	310 (310–319)	10
KSC-V	320 (320–329)	10

Adobe-Korea1-1: 700 UniqueIDs		
KSCms-UHC-H	430 (450–559)	130
KSC-Johab-H	560 (560–679)	120
KSCms-UHC-V	680 (590–689)	10
KSC-Johab-V	690 (580–699)	10
Adobe-Korea1-1	*330 (330–429)*	*100*

Adobe-CNS1-0: 1060 UniqueIDs		
B5pc-H	0 (0–99)	100
B5-H	100 (100–199)	100
ETen-B5-H	200 (200–299)	100
CNS-EUC-H	300 (300–699)	400
CNS1-H	700 (700–799)	100
CNS2-H	800 (800–899)	100
B5pc-V	900 (900–909)	10
B5-V	910 (910–919)	10
ETen-B5-V	920 (920–929)	10
CNS-EUC-V	930 (930–939)	10
CNS1-V	940 (940–949)	10
CNS2-V	950 (950–959)	10
Adobe-CNS1-0	960 (960–1059)	100

Adobe-CNS1-1: 1160 UniqueIDs		
Adobe-CNS1-1	*1060 (1060–1159)*	*100*

Note that several CMap files are not included in the above list because they do not include a */UIDOffset* dictionary entry.

Appendix A
Perl Program for Creating CMap Files

Below is a short Perl program that collapses CMap mapping lines whose CIDs and character codes are contiguous (it assumes a complete CMap file as input).

This tool, by default (that is, when invoked with no command-line options), takes a CMap file as standard input (STDIN), and outputs a CMap file to standard output (STDOUT). In addition to collapsing CMap mapping lines with contiguous CIDs and character codes, and separating the mapping lines into chunks of 100, statistics, such as

- the number of total CMap lines;
- the number of collapsed CMap lines (if any);
- the number of character mappings; and
- the average character mappings per CMap line

are written to standard error (STDERR). This tool supports a '-e' switch which, when invoked on the command line, expands the character code to CID mappings into one mapping per line. In effect, it reverses this tool's default collapsing action. Other miscellaneous error detection is automatically performed for the code space range and character code to CID mapping sections.

```
#!/usr/local/bin/perl -w

$count = $tally = $expand = $expanded = $processed = $collapsed = $index = 0;
$chunk = "";
$header = 1;

while ($ARGV[0]) {
  if ($ARGV[0] eq "-e") {
    $expand = 1;
    print STDERR "Expanding CMap file...";
    shift;
  } else {
    print STDERR "Invalid option: $ARGV[0]! Skipping...\n";
    shift;
  }
```

```
}

if (!$expand) {
  print STDERR "Compressing CMap file...";
}

while (defined ($line = <STDIN>) ) {
  chop($line);
  if ($line =~ /usecmap/) {
    print STDOUT "$line\n";
    &GetHorizCodeSpace;
  } elsif ($line =~ /begincodespacerange/) {
    chop($line = <STDIN>);
    until ($line =~ /endcodespacerange/) {
      $line =~ tr/a-f/A-F/;
      if (($all,$start,$end) = $line =~ /^\s*(<([\dA-F]+)>\s+<([\dA-
F]+)>)\s*$/) {
        $enc_s[$index] = hex($start);
        $enc_e[$index] = hex($end);
        $enc_l[$index] = length($start);
        $index++;
        $chunk .= "  $all\n";
      }
      chop($line = <STDIN>);
    }
    print STDOUT "$index begincodespacerange\n${chunk}endcodespacerange\n";
    $chunk = "";
  } elsif ($line =~ /begincidrange/) {
    $header = 0;
    chop($line = <STDIN>);
    until ($line =~ /endcidrange/) {
      $line =~ tr/A-F/a-f/;
      if (($hs,$he,$c) = $line =~ /^\s*<([\da-f]+)>\s+<([\da-
f]+)>\s+(\d+)\s*$/) {
        $s = hex($hs); $e = hex($he);
        if (!$expand) {
          unless (!$processed) {
            if ((($s % 256) == (($oe % 256) + 1)) && ($s == ($oe + 1)) &&
            ($c == ($oc + ($oe - $os) + 1))) {
              $c = $oc; $s = $os;
              $collapsed++;
            } else {
              $i++;
              if ($i < 100) {
                $d = &getlength($os,$oe);
                $chunk .= sprintf("<%0${d}x> <%0${d}x> $oc\n",$os,$oe);
                $count += ($oe - $os) + 1;
              } else {
                $d = &getlength($os,$oe);
                $chunk .= sprintf("<%0${d}x> <%0${d}x> $oc\n",$os,$oe);
                $chunk .= "endcidrange\n\n";
                $count += ($oe - $os) + 1;
                print STDOUT "$i begincidrange\n$chunk";
                $tally += $i;
                $chunk = "";
                $i = 0;
              }
            }
```

```
          }
          $oc = $c; $oe = $e; $os = $s;
          $processed++;
          chop($line = <STDIN>);
        } else {
          $d = &getlength($s,$e);
          foreach $i ($s .. $e) {
            $chunk .= sprintf("<%0${d}x> <%0${d}x> $c\n",$i,$i);
            $c++;
          }
          $expanded += ($e - $s) + 1;
          $processed++;
          chop($line = <STDIN>);
        }
      } else {
        chop($line = <STDIN>);
      }
    }
  } elsif ($header) {
    print STDOUT "$line\n";
  }
}

if ($expand) {
  print STDOUT "$expanded begincidrange\n${chunk}endcidrange\nendcmap\n";
} else {
  $d = &getlength($os,$oe);
  $chunk .= sprintf("<%0${d}x> <%0${d}x> $oc\n",$os,$oe);
  $count += ($oe - $os) + 1;
  $i++;
  $tally += $i;
  print STDOUT "$i begincidrange\n${chunk}endcidrange\nendcmap\n";
}
print STDOUT "CMapName currentdict /CMap defineresource pop\nend\nend\n\n";
print STDOUT "\%\%EndResource\n\%\%EOF\n";

print STDERR "Done.\n";
if ($expand) {
  print STDERR "Expanded $processed CMap lines into $expanded lines.\n";
} else {
  print STDERR "Collapsed $collapsed of $processed CMap lines.\n";
  print STDERR "Total $count character mappings in $tally CMap lines.\n";
  printf STDERR "%3.2f average mappings per CMap line.\n",($count / $tally);
}

sub getlength {
  foreach $index (0 .. $#enc_s) {
    if (($_[0] >= $enc_s[$index]) && ($_[1] <= $enc_e[$index])) {
      return $enc_l[$index];
    }
  }
  printf STDERR "Codes 0x%X through 0x%X are out of range!\n",$_[0],$_[1];
  print STDERR "WARNING: Defaulting to eight-digit representation...\n";
  return 8;
}

sub GetHorizCodeSpace {
  ($hcmap) = $line =~ m#/(.+)\s+usecmap\s*#;
```

```
  open(HCMAP,"< $hcmap") || die "Cannot locate horizontal CMap file";
  while (defined ($line = <HCMAP>) ) {
    if ($line =~ /begincodespacerange/) {
      chop($line = <HCMAP>);
      until ($line =~ /endcodespacerange/) {
        if (($start,$end) = $line =~ /^\s*<([\da-f]+)>\s+<([\da-f]+)>\s*$/i) {
          $enc_s[$index] = hex($start);
          $enc_e[$index] = hex($end);
          $enc_l[$index] = length($start);
          $index++;
        }
        chop($line = <HCMAP>);
      }
    }
  }
  close(HCMAP);
}
```