Font Switching Optimizations

Adobe Developer Support

Technical Note #5048

31 March 1992

Adobe Systems Incorporated
Adobe Developer Technologies
345 Park Avenue
San Jose, CA 95110
http://partners.adobe.com/

PN LPS5048
Copyright © 1990–1992 by Adobe Systems Incorporated. All rights reserved.

No part of this publication may be reproduced, stored in a retrieval system, or transmitted, in any form or by any means, electronic, mechanical, photocopying, recording, or otherwise, without the prior written consent of the publisher. Any software referred to herein is furnished under license and may only be used or copied in accordance with the terms of such license.

PostScript, the PostScript logo, Display PostScript, and the Adobe logo are trademarks of Adobe Systems Incorporated which may be registered in certain jurisdictions. Apple, AppleTalk, and LaserWriter are registered trademarks of Apple Computer, Inc. Other brand or product names are the trademarks or registered trademarks of their respective holders.

*This publication and the information herein is furnished AS IS, is subject to change without notice, and should not be construed as a commitment by Adobe Systems Incorporated. Adobe Systems Incorporated assumes no responsibility or liability for any errors or inaccuracies, makes no warranty of any kind (express, implied or statutory) with respect to this publication, and expressly disclaims any and all warranties of merchantability, fitness for particular purposes and noninfringement of third party rights.*
Contents

Font Switching Optimizations 5

1 Introduction 5

2 Font Switching Techniques and Issues 5
   Using a Procedure 6
   Inverted Coordinate Systems 7
   Re-Encoding Fonts 8
   Using selectfont 11
   More on Font Dictionary Caching 13

3 Summary 15

Appendix A: Code Used For Timing Studies 17

A.1 Code to Compute Timings in Table 1 17
A.2 Code to Compute Timings in Table 2 20
A.3 Code to Compute Timings in Table 3 24

Appendix B: Changes Since Earlier Versions 27

Index 29
Font Switching Optimizations

1 Introduction

Supporting multiple fonts is one of the basic functions of a PostScript™ printer driver. Therefore, being able to switch efficiently from one font to another becomes very important when writing an efficient printer driver. This technical note addresses the efficiency issues related to switching fonts, and presents techniques for optimizing this operation.

Setting a font might include several prerequisite operations, such as downloading or re-encoding the font. Although these topics are covered in more detail in other technical notes, the order in which they happen is important, and will be noted later. For the purpose of this document, we will assume that the font has been downloaded or is already available in the printer.

2 Font Switching Techniques and Issues

The most basic method of switching fonts (in-line code to set a font) is shown in the following code example.

```
/Times-Roman findfont 12 scalefont setfont
```

This method of selecting a font works for both Level 1 and Level 2 printers. It is the first way that a new PostScript language programmer learns. It is the most straightforward way to set a font, but not necessarily the most efficient. In the following sections we will improve upon this simple piece of code.
2.1 Using a Procedure

Setting a font in the above manner requires the PostScript interpreter to execute three name lookups: \texttt{findfont}, \texttt{scalefont}, and \texttt{setfont}. It also requires 42 bytes, including the font name /Times-Roman and the point size 12. Efficiency can be improved somewhat by defining a procedure to replace some of the in-line code.

```
/sf { % scale name
   findfont exch scalefont setfont }
def
```

Setting a font then becomes

```
12 /Times-Roman sf
```

The advantages of defining such a procedure is that transmission time is reduced because fewer characters are being sent. The byte count to switch to the font becomes 17 instead of 42. If a page contains 50 or 60 font changes (not an uncommon situation), this can be a significant savings, especially over a slow communications channel such as a serial connection at 9600 baud.

In addition, the code will execute faster if a bound procedure is defined, because the interpreter does not have to look up the names \texttt{findfont}, \texttt{scalefont}, and \texttt{setfont} each time the procedure is executed. (The name lookup is done once for each operator during the \texttt{bind}.) Again, defining a procedure such as \texttt{sf} will work on both Level 1 and Level 2 devices. The following is a bound PostScript language procedure to set a font.

```
/sf { % scale name
   findfont exch scalefont setfont }
bind def
```

A timing study shown in the Table 1 compares the performance differences between the three bodies of code previously discussed.

<table>
<thead>
<tr>
<th>Technique</th>
<th>9600 baud</th>
<th>AppleTalk®</th>
</tr>
</thead>
<tbody>
<tr>
<td>In-line code</td>
<td>4.541</td>
<td>0.373</td>
</tr>
<tr>
<td>PostScript Language procedure</td>
<td>2.037</td>
<td>0.308</td>
</tr>
<tr>
<td>Bound procedure</td>
<td>2.036</td>
<td>0.302</td>
</tr>
</tbody>
</table>

(in seconds) Tests performed on an Apple® LaserWriter® NTX.
2.2 Inverted Coordinate Systems

Depending on the type of font effects (shearing, skewing, and so on) your application allows, and depending on the coordinate system in which your driver works, you might need to do more to the font than simply scale it before setting. For instance, the origin of the coordinate system for many computer displays begins in the upper left corner with the numeric values of the y-axis increasing toward the bottom of the display. The origin in the default state of the PostScript interpreter is at the lower left corner of the page, with the numeric values of the y-axis increasing toward the top of the page.

If your application works in a different coordinate system, you might want to modify the PostScript interpreter’s coordinate system. The following line of code can be used to “flip” the PostScript language coordinate system, so that the origin is at the upper left corner of a letter size page.

```
0 792 translate 1 -1 scale
```

*Note* The above code to “flip” the coordinate system is inherently device dependent. It is assuming U.S. letter size paper, which is 11” in height.

Such a coordinate system transformation usually occurs at the beginning of each page, so that all following graphics commands refer to the flipped coordinate system. After flipping the coordinate system, however, each font used will be rendered upside-down.

**Figure 1** Text shown at (72, 72) in default and flipped coordinate systems

![Default and Flipped Coordinate Systems](image)
The `makefont` operator must be used, instead of `scalefont`, to undo this effect by explicitly setting the size and orientation of the font. (The `scalefont` operator is a special case of `makefont` that sets the width and height values of the font matrix to the same value.) The following example shows how a previous procedure can be modified to perform a `makefont` instead of a `scalefont` operation. This results in having to pass an entire matrix to the procedure instead of single number representing the desired point size.

**Example 1: Using `makefont` to invert a font**

```
/sfm { % mtrx fontname
    findfont exch makefont setfont
} bind def
```

Setting a font then becomes:

```
[12 0 0 -12 0 0] /Times-Roman sfm
```

Depending on the speed of the communication line, using the `makefont` operator in this manner might be less efficient than using `scalefont`, since there is more data to be transmitted. But using `makefont` allows for easy transformations of text, and if you need to scale the font so it is oriented correctly in a flipped coordinate system then `makefont` must be used.

Additionally, the `makefont` operator actually executes more quickly than the `scalefont` operator, although the difference in execution speed is minimal, and might be outweighed by the cost of the extra data, depending on the transmission channel. A more efficient use of `makefont` is to flip the font matrix once with the matrix `[1 0 0 -1 0 0]` and using the inverted font as the basis for scaling subsequent fonts as needed. Section 2.3 discusses inverting the font matrix at the same time as re-encoding a font.

### 2.3 Re-Encoding Fonts

Many drivers do some form of re-encoding of fonts to gain access to a different set of characters. Although re-encoding is an operation that takes time, by naming the re-encoded font appropriately, you can reduce transmission overhead and thereby increase the speed of the driver.

Re-encoding a font involves making a copy of the top-level font dictionary, replacing the encoding vector in the copied font with a new one, and executing `definefont`, thus registering the new font in `FontDirectory`. 
Example 2: Re-encoding every font

```
/RX { % /NewFontName /FontName RX -
  findfont dup length dict begin
  { % copy font dictionary into new dict
    1 index /FID ne {def} {pop pop} ifelse
  } forall
  /Encoding MyArr def
  /FontName 1 index def
  currentdict definefont
  end pop
} bind def

/fm {findfont exch makefont setfont} bdef

/RE-Times-Roman /Times-Roman RX [10 0 0 -10 0 0] /RE-Times-Roman fm
```

In this example, the variable **NewFontName** is the new name of the font that will be used for subsequent calls to the font setting procedure. **FontName** is the name of the base font being re-encoded.

Executing this long piece of code for every font change is unnecessary. To eliminate the unnecessary invocations, the host must keep track of the fonts that have been re-encoded, and only place in the job stream the code needed to set the font. Example 3 shows how the script is shortened using the procedures defined in Example 2.

Example 3: Re-encoding once per font

```
/RE-Times-Roman /Times-Roman RX
/RE-Times-Bold /Times-Bold RX

[10 0 0 -10 0 0] /RE-Times-Roman fm
[12 0 0 -12 0 0] /RE-Times-Bold fm
```

Choosing a short name for the re-encoded font reduces the transmission time for specifying font changes. For instance, instead of naming the re-encoded version of New Century Schoolbook as NewCenturySchoolbook-CodePage792 (or RE-Times-Roman), use a lookup table on the host side to map the name to F1. This is a savings of 30 characters. For a page with 50 font changes, that means 1500 bytes, or about 1.25 additional seconds at 9600 baud, not including interpretation time.

Example 4: Using short names

```
/F1 /Times-Roman RX
/F2 /Times-Bold RX

[10 0 0 -10 0 0] /F1 fm
[12 0 0 -12 0 0] /F2 fm
```

If you are using **makefont** to create a flipped font, as discussed earlier, you can save time by doing the **makefont** in the re-encoding procedure. The **makefont** should occur after the **definefont**. Character caching will not
occur across jobs if \texttt{makefont} is used to change the font matrix before executing \texttt{definefont}. This is because of how the PostScript interpreter decides whether a character is present in the font cache. If the font matrix for a particular character is different from the base font’s matrix, the interpreter will not recognize characters that might already be in the cache as characters that belong to the same font.

Example 5 is a re-encoding procedure that also saves a 1-point flipped font dictionary into a short key that is defined into the current dictionary.

\textbf{Example 5: Re-encoding and inverting the font in the same procedure}

\begin{verbatim}
/MyArr [...encoding array definition here...] def
/FlipMtx [1 0 0 -1 0 0] def
/RE { % /ShortName /FontName
    findfont dup length dict begin
    { % copy font dictionary into new dict
    1 index /FID ne
    {def} {pop pop} ifelse
    } forall
    /Encoding MyArr def
    currentdict definefont
    end
    FlipMtx makefont def
} bind def

/F1 /Times-Roman RE
\end{verbatim}

By using the re-encoding procedure to invert the font’s matrix, the number of calls to the \texttt{makefont} operator can be reduced. Since the matrices passed to \texttt{makefont} are larger than a simple scale value, the total amount of data transmitted is reduced.

When using the procedure in Example 5, it is important to note that F1 is defined in the current dictionary, and not in \texttt{FontDirectory}. Therefore, it is not necessary to execute a \texttt{findfont}. Placing F1 in-line leaves the flipped font dictionary on the operand stack. This is known as \textit{font dictionary caching}, and results in reduced lookup time.

\textbf{Table 2} Comparing code to switch and re-encode fonts with various methods

<table>
<thead>
<tr>
<th>Technique</th>
<th>9600 baud</th>
<th>AppleTalk</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ex. 2: Re-encode every font</td>
<td>7.080</td>
<td>4.268</td>
</tr>
<tr>
<td>Ex. 3: Re-encode once (long names)</td>
<td>3.874</td>
<td>0.500</td>
</tr>
<tr>
<td>Ex. 4: Re-encode once (short names)</td>
<td>2.601</td>
<td>0.492</td>
</tr>
<tr>
<td>Ex. 5: Cached dictionaries</td>
<td>1.125</td>
<td>0.368</td>
</tr>
</tbody>
</table>

See Appendix A for a complete code listing used in these timing studies.
2.4 Using selectfont

When the Display PostScript™ system was designed, it was recognized that a more efficient method for switching fonts was needed at the language level. Therefore, the `selectfont` operator, which combines the functions of `findfont`, `scalefont`, and `setfont`, was introduced. A call to `selectfont` is of the following form:

```
/Times-Roman 12 selectfont
```

The `selectfont` operator is standard in Level 2 printers, as well as Display PostScript systems. Since some drivers must perform more than just scaling operations to the font matrix, it is also possible to pass a matrix to `selectfont`.

The benefit of using the `selectfont` operator might seem obvious. Instead of executing a PostScript language procedure that itself executes several operators, using `selectfont` invokes only one operator, which is written in compiled code, so it executes very quickly.

However, another benefit is not obvious: `selectfont` maintains a cache of recently scaled and set font dictionaries. This means if you are switching between a small number of fonts in your page description (for example, a roman and an italic face), using `selectfont` eliminates the need to re-scale each font dictionary, because `selectfont` can get a dictionary out of its font dictionary cache.

Additionally, it is more efficient to shorten font names (especially easy if they are going to be re-encoded) and also shorten the operator name.

Example 6: Calling `selectfont` efficiently

```
/RE { % /NewFontName [NewEncodingArray] /FontName RE -
    findfont dup length dict begin
    { % copy font dictionary into new dict
        1 index /FID ne
        [def] {pop pop} ifelse
    } forall
    /Encoding exch def
    /FontName 1 index def
    currentdict definefont pop
end
} bind def

/s1 /selectfont load def
/MyVec StandardEncoding 256 array copy def

/F1 MyVec /Times-Roman RE
/F1 10 s1
```

Since `selectfont` is not available on Level 1 printers, it must be emulated if your driver is to work with both Level 1 and Level 2 printers. The following code verifies whether `selectfont` is available, and if not, defines an emulation...
for it. The resulting code will produce the desired results on either a Level 1 or Level 2 printer, although the same performance benefits are not possible on a Level 1 printer.

**Example 7: Conditional emulation of the `selectfont` operator**

```
/s /selectfont where
  { pop /selectfont load }
  { exch findfont exch dup type /arraytype eq
    {makefont}{scalefont} ifelse setfont } bind
  } ifelse
def
```

If your driver never uses the `makefont` case of `selectfont`, you might want to define only a partial emulation of `selectfont` to improve the emulation performance. Note, however, that incomplete emulations of operators should never be defined with the original operator’s name. This will help to avoid collisions with other drivers (or even your own) that check for the existence of an operator. The assumption is that if the operator exists, it is the real implementation or a full emulation of that operator. The emulation used for the timing studies appears in Example 8.

**Example 8: Short (and incomplete) `selectfont` emulation**

```
/SN { %scale /Font SN
  findfont exch scalefont setfont
} bind def

/F1 MyVec /Courier RE
10 /F1 SN
```

2.5 More on Font Dictionary Caching

Example 2 showed how to save the re-encoded font dictionary into a key that is defined in the current dictionary. In the example, the font dictionary saved is a one-point font. Most text setting applications will want to scale the font to various sizes before displaying any text.

One method is to have a procedure that takes a one-point font dictionary and the desired size, and scale and set the font. The sf procedure has been re-written to do this. A typical line from the resulting page description to set a font is very short, as shown.

```
/sf % fontdict scale
{ scalefont setfont } bind def
F1 24 sf
```

In addition, if certain sizes of particular fonts are used frequently, you might want to also save the scaled font dictionary into a key, so it can be set without scaling it every time. This technique requires two procedures.

In the following code sample, sfc takes a new key name, the one-point font dictionary and the scale factor. It scales the font dictionary by the requested amount and saves that off with the new key. A second procedure, sfnt, is a new, shorter name for the operator setfont. It would be called repeatedly to set an already scaled font dictionary to be the current font.

```
/sfc % newkey fontdict scale
{ scalefont def } bind def
/sfnt /setfont load def
/F2 F1 24 sfc
F2 sfnt
```

This technique requires more data to be maintained on the host because instead of having a handful of one-point fonts of which to keep track, there are possibly many more scaled versions of those fonts.

These two techniques can be combined, saving the scaled font dictionaries that are most frequently used and scaling the one-point size font dictionaries when a different, less-used font size is requested.

Maintaining a cache of font dictionaries on the host is a programming technique that is very efficient on both Level 1 and Level 2 printers. Even though Level 2 has the selectfont operator that performs font dictionary caching in the interpreter, to achieve the same operation on Level 1 printers, the application must do the work on the host, as previously discussed.
If your application is already doing this work for Level 1 printers, it should use the same technique for Level 2 printers. This actually improves execution time of the resulting page description even on Level 2 printers by removing the burden of caching font dictionaries from the PostScript interpreter.

An example of the output of a driver using cached scaled font dictionaries appears in Example 9.

**Example 9: Cached scaled font dictionaries**

```
/SF { % /FontIndex FontSize /RE_Fontname
  findfont exch
  scalefont dup def
  setfont
} bind def

/sf % FontIndex
/setfont load def

/F1 12 /Times-Roman SF
/F2 10 /Times-Bold SF
F1 sf
F2 sf
```

Table 3 compares an emulation of the `selectfont` operator versus using the built-in `selectfont` operator versus using code on the host to keep track of scaled font dictionaries.

**Table 3** Comparing `selectfont` with host-based caching

<table>
<thead>
<tr>
<th>Technique</th>
<th>9600 baud</th>
<th>AppleTalk</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ex. 8: <code>selectfont</code> emulation</td>
<td>1.229</td>
<td>0.469</td>
</tr>
<tr>
<td>Ex. 6: <code>selectfont</code> operator</td>
<td>1.208</td>
<td>0.173</td>
</tr>
<tr>
<td>Ex. 9: Cached scaled font dictionaries</td>
<td>0.810</td>
<td>0.138</td>
</tr>
</tbody>
</table>
3 Summary

The fastest method for font switching will always depend on a number of factors, such as the communication channel bandwidth, the language level of the interpreter, and the length of the names chosen for the re-encoded font dictionaries.

For both Level 1 and Level 2 printers, it is important to reduce the amount of data transmitted to the printer. The easiest and most effective way to do this is to create prolog procedures that reduce the amount of in-line code. These procedures should have one- or two-character names and should use `bind` for maximum efficiency.

Another important method to reduce data transmission is renaming of re-encoded or cached font dictionaries to short names (2 to 4 characters). By using cached scaled font dictionaries, data transmission can be reduced even further, because the point size of the font need not be sent each time.

For Level 2 printers, the `selectfont` operator is a good alternative to using cached scaled font dictionaries. Because caching font dictionaries requires more overhead on the application side, using `selectfont` or a `selectfont` emulation (for Level 1 printers) can make sense in low-memory host environments. Additionally, this technique requires the least amount of programming effort.
Appendix A: Code Used For Timing Studies

A.1 Code to Compute Timings in Table 1

This appendix contains code listings of the jobs that were used to compute the timing results throughout the document. The following code compares using in-line code with calls to \texttt{findfont} ... \texttt{scalefont} ... \texttt{setfont} versus using a procedure for the same function versus using a \textit{bound} procedure. Some of the script has been removed to save space in this document.

%! %Title: fig1.ps
%! %Creator: developer support group
%! %Date: 3-oct-90

errordict begin
  /syntaxerror {pstack stop} def
end

/bdef {bind def} bind def

/realtile where {pop}{/realtile /usertime load def} ifelse
/timeon{/oldtime realtime def}bdef
/timeoff{realtime oldtime sub 1000 div ==}bdef

/sf {
  findfont exch scalefont setfont
} def

/sb {
  findfont exch scalefont setfont
} bind def

/Times-Roman findfont 10 scalefont setfont % eliminate "1st find" overhead
/Times-Bold findfont 12 scalefont setfont
(The time for find/scale/set in-line is: ) print
timeon
/Times-Roman findfont 10 scalefont setfont
/Times-Bold findfont 12 scalefont setfont
/Times-Roman findfont 10 scalefont setfont
/Times-Bold findfont 12 scalefont setfont
/Times-Roman findfont 10 scalefont setfont
/Times-Bold findfont 12 scalefont setfont
/Times-Roman findfont 10 scalefont setfont
/Times-Bold findfont 12 scalefont setfont
/Times-Roman findfont 10 scalefont setfont
/Times-Bold findfont 12 scalefont setfont
/Times-Roman findfont 10 scalefont setfont
/Times-Bold findfont 12 scalefont setfont
/Times-Roman findfont 10 scalefont setfont
/Times-Bold findfont 12 scalefont setfont
/Times-Roman findfont 10 scalefont setfont
/Times-Bold findfont 12 scalefont setfont
/Times-Roman findfont 10 scalefont setfont
/Times-Bold findfont 12 scalefont setfont
/Times-Roman findfont 10 scalefont setfont
/Times-Bold findfont 12 scalefont setfont
timeoff

(The time for the sf (unbound) procedure is: ) print
timeon
10 /Times-Roman sf
12 /Times-Bold sf
10 /Times-Roman sf
12 /Times-Bold sf
10 /Times-Roman sf
12 /Times-Bold sf
10 /Times-Roman sf
12 /Times-Bold sf
10 /Times-Roman sf
12 /Times-Bold sf
10 /Times-Roman sf
12 /Times-Bold sf
10 /Times-Roman sf
12 /Times-Bold sf
10 /Times-Roman sf
12 /Times-Bold sf
10 /Times-Roman sf
12 /Times-Bold sf
10 /Times-Roman sf
12 /Times-Bold sf
10 /Times-Roman sf
12 /Times-Bold sf
timeoff
(The time for the s2 (bound) procedure is: ) print
timeon
  10 /Times-Roman sb
  12 /Times-Bold sb
  10 /Times-Roman sb
  12 /Times-Bold sb
  10 /Times-Roman sb
  12 /Times-Bold sb
  10 /Times-Roman sb
  12 /Times-Bold sb
  10 /Times-Roman sb
  12 /Times-Bold sb
..80 similar lines omitted for brevity..
  10 /Times-Roman sb
  12 /Times-Bold sb
  10 /Times-Roman sb
  12 /Times-Bold sb
  10 /Times-Roman sb
  12 /Times-Bold sb
  10 /Times-Roman sb
  12 /Times-Bold sb
  10 /Times-Roman sb
  12 /Times-Bold sb
timeoff
A.2 Code to Compute Timings in Table 2

In the following code, some of the script has been removed to save space in this document.

%! 
%%Title: make vs cache
%%Creator: developer support group
%%Date: 21-sep-90

% this file tests re-encode and makefont in-line (each time)
% vs re-encoded non-flipped fonts with long names
% vs re-encoded non-flipped fonts with short names
% vs re-encoded and flipped fonts with short names

/bdef {bind def} bind def
/realt ime where {pop}{/realt ime /usertime load def} ifelse
/timeon{/oldtime realtime def}bdef
/timeoff{realtime oldtime sub 1000 div}bdef

/RE { % /CacheName /NewFontName /FontName RE -
findfont dup length dict begin
{ % copy font dictionary into new dict
1 index /FID ne {def} {pop pop} ifelse
} forall
(Encoding MyArr def
/FontName 1 index def
currentdict definefont
end
FlipMtx makefont def
} bind def

/RX { % /NewFontName /FontName RX -
findfont dup length dict begin
{ % copy font dictionary into new dict
1 index /FID ne {def} {pop pop} ifelse
} forall
/Encoding MyArr def
/FontName 1 index def
currentdict definefont
end pop
} bind def

/ss {scalefont setfont} bdef
/ms {makefont setfont} bdef
/fm {findfont exch makefont setfont} bdef

/MyArr StandardEncoding 256 array copy def
/FlipMtx [1 0 0 -1 0 0] def

%%%EndProlog
%%%BeginSetup
errordict begin
    /typecheck {pstack stop} def
    /VError {pstack stop} def
end

0 792 translate 1 -1 scale
% eliminate "first find" effects:
/Times-Roman findfont 10 scalefont setfont
/Times-Bold findfont 12 scalefont setfont
%%EndSetup

(The time for in-line encoded each time code is: ) print
timeon

/RE-Times-Roman /Times-Roman RX [10 0 0 -10 0 0] /RE-Times-Roman fm
/RE-Times-Bold /Times-Bold RX [12 0 0 -12 0 0] /RE-Times-Bold fm
/RE-Times-Roman /Times-Roman RX [10 0 0 -10 0 0] /RE-Times-Roman fm
/RE-Times-Bold /Times-Bold RX [12 0 0 -12 0 0] /RE-Times-Bold fm
/RE-Times-Roman /Times-Roman RX [10 0 0 -10 0 0] /RE-Times-Roman fm
/RE-Times-Bold /Times-Bold RX [12 0 0 -12 0 0] /RE-Times-Bold fm
/RE-Times-Roman /Times-Roman RX [10 0 0 -10 0 0] /RE-Times-Roman fm
/RE-Times-Bold /Times-Bold RX [12 0 0 -12 0 0] /RE-Times-Bold fm
/RE-Times-Roman /Times-Roman RX [10 0 0 -10 0 0] /RE-Times-Roman fm
/RE-Times-Bold /Times-Bold RX [12 0 0 -12 0 0] /RE-Times-Bold fm
/RE-Times-Roman /Times-Roman RX [10 0 0 -10 0 0] /RE-Times-Roman fm
/RE-Times-Bold /Times-Bold RX [12 0 0 -12 0 0] /RE-Times-Bold fm

..80 similar lines omitted for brevity..

/RE-Times-Roman /Times-Roman RX [10 0 0 -10 0 0] /RE-Times-Roman fm
/RE-Times-Bold /Times-Bold RX [12 0 0 -12 0 0] /RE-Times-Bold fm
/RE-Times-Roman /Times-Roman RX [10 0 0 -10 0 0] /RE-Times-Roman fm
/RE-Times-Bold /Times-Bold RX [12 0 0 -12 0 0] /RE-Times-Bold fm
/RE-Times-Roman /Times-Roman RX [10 0 0 -10 0 0] /RE-Times-Roman fm
/RE-Times-Bold /Times-Bold RX [12 0 0 -12 0 0] /RE-Times-Bold fm
/RE-Times-Roman /Times-Roman RX [10 0 0 -10 0 0] /RE-Times-Roman fm
/RE-Times-Bold /Times-Bold RX [12 0 0 -12 0 0] /RE-Times-Bold fm
/RE-Times-Roman /Times-Roman RX [10 0 0 -10 0 0] /RE-Times-Roman fm
/RE-Times-Bold /Times-Bold RX [12 0 0 -12 0 0] /RE-Times-Bold fm

(time for make inline and long names: ) print
timeon

/RE-Times-Roman /Times-Roman RX
/RE-Times-Bold /Times-Bold RX

[10 0 0 -10 0 0] /RE-Times-Roman fm
[12 0 0 -12 0 0] /RE-Times-Bold fm
[10 0 0 -10 0 0] /RE-Times-Roman fm
[12 0 0 -12 0 0] /RE-Times-Bold fm
[10 0 0 -10 0 0] /RE-Times-Roman fm
[12 0 0 -12 0 0] /RE-Times-Bold fm
[10 0 0 -10 0 0] /RE-Times-Roman fm
[12 0 0 -12 0 0] /RE-Times-Bold fm

..80 lines omitted for brevity..
(The time for make inline and short names: ) print
timeon
/F1 /Times-Roman RX
/F2 /Times-Bold RX

[10 0 0 -10 0 0] /F1 fm
[12 0 0 -12 0 0] /F2 fm
[10 0 0 -10 0 0] /F1 fm
[12 0 0 -12 0 0] /F2 fm
[10 0 0 -10 0 0] /F1 fm
[12 0 0 -12 0 0] /F2 fm
[10 0 0 -10 0 0] /F1 fm
[12 0 0 -12 0 0] /F2 fm
[10 0 0 -10 0 0] /F1 fm
[12 0 0 -12 0 0] /F2 fm
[10 0 0 -10 0 0] /F1 fm
[12 0 0 -12 0 0] /F2 fm

..80 similar lines omitted for brevity..

[/F1 fm]
[/F2 fm]
[/F1 fm]
[/F2 fm]
[/F1 fm]
[/F2 fm]
[/F1 fm]
[/F2 fm]
[/F1 fm]
[/F2 fm]
[/F1 fm]
[/F2 fm]

..80 similar lines omitted for brevity..

timeoff

(The time for cached flipped font dict (short name): ) print
timeon
/F1 /RE-Times-Roman /Times-Roman RE
/F2 /RE-Times-Bold /Times-Bold RE

F1 10 ss
F2 12 ss
F1 10 ss
F2 12 ss
F1 10 ss
F2 12 ss
F1 10 ss
F2 12 ss
F1 10 ss
F2 12 ss

..80 similar lines omitted for brevity..
F1 10 ss
F2 12 ss
F1 10 ss
F2 12 ss
F1 10 ss
F2 12 ss
F1 10 ss
F2 12 ss
timeoff
A.3 Code to Compute Timings in Table 3

In the following code, some of the script has been removed to save space in this document.

%! %%%Title: select.ps %%%Creator: developer support group %%%Date: 21-sep-90

% this file tests selectfont vs findfont vs scaled font dicts %
/bdef {bind def} bind def

/realtimewhere {pop}{/realtimewhere /usertime load def} ifelse
/timeon{/oldtime realtime def}bdef
/timeoff{realtime oldtime sub 1000 div ==}bdef

/RE { % /NewFontName [NewEncodingArray] /FontName RE -
  findfont dup length dict begin
  { % copy font dictionary into new dict
    1 index /FID ne
    {def} {pop pop} ifelse
  } forall
  /Encoding exch def
  /FontName 1 index def
  currentdict definefont pop
  end
} bdef

/s1 /selectfont load def

/SF { % /FontIndex FontSize /Fontname SF -- scale, set, and cache the font
  findfont
  exch scalefont dup
  3 1 roll def
  setfont
} bdef

/sf /setfont load def

/SN { % scale /Font SN -- Selectfont with no dictionary caching
  findfont exch scalefont setfont
} bdef

/MyVec StandardEncoding 256 array copy def

%%EndProlog

(The time for Selectfont is: ) print
timeon
   /F1 MyVec /Times-Roman RE
   /F2 MyVec /Times-Bold RE
   /F1 10 s1
   /F2 12 s1
   /F1 10 s1
   /F2 12 s1
   /F1 10 s1
   /F2 12 s1
   /F1 10 s1
   /F2 12 s1
   /F1 10 s1
   /F2 12 s1
   ..80 similar lines omitted for brevity..
   /F1 10 s1
   /F2 12 s1
   /F1 10 s1
   /F2 12 s1
   /F1 10 s1
   /F2 12 s1
   /F1 10 s1
   /F2 12 s1
   timeoff

   (The time for Font Dictionary Caching is: ) print

itimeon
   /Fx MyVec /Helvetica RE
   /Fy MyVec /Helvetica-Bold RE
   /F1 10 /Fx SF
   /F2 12 /Fy SF
   F1 sf
   F2 sf
   F1 sf
   F2 sf
   F1 sf
   F2 sf
   F1 sf
   F2 sf
   F1 sf
   F2 sf
   ..80 similar lines omitted for brevity..
   F1 sf
   F2 sf
   F1 sf
   F2 sf
   F1 sf
   F2 sf
   F1 sf
   F2 sf
   timeoff
(The time for Selectfont emulation minus caching is: ) print
timeon
/F1 MyVec /Courier RE
/F2 MyVec /Courier-Bold RE
10 /F1 SN
12 /F2 SN
10 /F1 SN
12 /F2 SN
10 /F1 SN
12 /F2 SN
10 /F1 SN
12 /F2 SN
10 /F1 SN
12 /F2 SN
...80 similar lines omitted for brevity..
10 /F1 SN
12 /F2 SN
10 /F1 SN
12 /F2 SN
10 /F1 SN
12 /F2 SN
10 /F1 SN
12 /F2 SN
timeoff
Appendix B: Changes Since Earlier Versions

Changes since March 20, 1991

- Document was reformatted in the new document layout and minor editorial changes were made.
Index

F
font switching
   flipped coordinate system  7
font dictionary
   cached scaled  14
   caching  13–14
in-line code  5
inverted coordinate system  7
   makefont  8
   scalefont  8
name lookups  6
optimization  5–15
re-encoding fonts  8–10
   definefont  8
   makefont  9
   short name  9
selectfont  11–12
   emulation  11
setting a font
   bound PostScript procedure  6
   in-line  5
   techniques  5–14
FontDirectory  8

S
sf procedure  6, 13
sfc procedure  13
sfnt procedure  13