Implementing Dynamic Cross-Referencing and PDF with PreTEX

Paul A. Mailhot
PreTEX, Inc.
2891 Oxford Street
Halifax, Nova Scotia, B3L 2V9
Canada
paul@pretex.com

Abstract

This paper discusses how we can create dynamic and interactive on-line documents in Portable Document Format (PDF), using TEX. PDF documents are generally device and platform independent, therefore, ideally suited for on-line publishing and information exhange. We will need to work in three programming languages: macros in TEX, and definitions in PostScriptand PDF. We begin by describing the steps necessary to process PDF through TEXand PostScript, followed by several examples of defining such TEX macros. The examples are quite easy to follow; however, some knowledge of PostScript and PDF programming is useful.

Paperless publishing

Publishing of books and documents has dramatically changed in the past few decades. Philip Taylor (1996) accurately described the emergence of computer typesetting with emphasis on Web document rendering applications, using portable multiplatform hypertext exchange standards, particularly focusing on the merits of PDF and HTML (including XML, SGML, et al.). For our purpose we can define *on-line* as any means by which a document can be electronically rendered; this is perhaps more aptly called *paperless printing*.

The popularity of on-line books and documents has spawned a diverse variety of on-line readers. These *readers*, for our purpose, are programs, applets, and interpreters which can *read* electronic data and output the image to a screen, in much the same way that one would see the output from a printer. Readers include proprietary single-platform programs, stand-alone multi-platform readers (including Frame Reader, Ghostscript, and Acrobat Reader), Web browsers (including HTML and JAVA), and Web browser plug-ins (including Acrobat). It is not at all wrong to suggest that TEX, along with a DVI previewer, is one of the first platform independent readers. Many people have generously contributed packages to the TEX cause, by which users can incorporate recent advancements in reader design such as HTML and PDF.

Portable Document Format

Portable Document Format, more commonly called PDF, was developed by Adobe Systems and is a descriptive programming language, devoid of software and hardware dependence, used to render documents. PDF is sometimes grossly misunderstood as being a subset of the PostScript format language. It is true that PDF and PostScript share many common features but each format language suits a particular task, and there are features found in each but not the other. Thomas Merz (1997) describes these similarities and differences in suitable detail. For our purpose we will concentrate on PDF documents, in particular addressing some hypertext features including bookmarks, links, and annotations.

A PDF document in general cannot be directly viewed, but rather, must be processed through a reader such as Adobe Acrobat Reader or Ghostscript. There also exist a number of application plug-ins for internet, word-processing, and desktop-publishing software which allow viewing of PDF documents. PDF documents are usually created by printing a document through a printer driver such as Acrobat PDF Writer or printing to a PostScript file which is in turn passed through Acrobat Distiller. The second method is more commonly used by the TEX community; a brief description is given by Amy Hendrickson (1998). Contributions such as `hyperref` (Thành and Rahtz, 1997) and `pdftex` (Thành, 1998) provide direct-to-PDF document processing.

PDF through PostScript

For our purpose we will concentrate on the method used by Hendrickson. The description of PDF and PostScript operators for hypertext links is found in Merz, but for our purpose we will briefly review it.

In order to create PDF links, we must first supply a set of raw PostScript instructions/definitions which will allow passage of PDF link code through Acrobat Distiller to create PDF files with links. This PDF code will be ignored by non-PDF devices such as PostScript printers. The following PostScript source code accommodates this by an `if–else` statement:

```
/pdfmark where
   {pop}
   {dictionary /pdfmark
    /cleartomark
    load put}
ifelse
```

In this PostScript code, the `where` command searches for a PostScript dict containing a definition of `pdfmark`. If the definition is found, then the `if` portion `{pop}` is executed and word `pdfmark` is popped off the execution stack by the PostScript interpreter; otherwise, `pdfmark` is defined to remove all the tokens between *mark* and the word `pdfmark` by using the PostScript operator `/cleartomark`. A mark in PostScript is the character `[`. We now have an avenue by which PDF code can be passed through both to PDF devices and non-PDF devices.

Most new DVI-to-PostScript interpreters have already included the above PostScript code in their document header. Those interpreters that do not include it can do so with the following TEX definition:

```
\def\initializePDF{
   \special{header=pdfparse.psc}}
```

This TEX macro tells TEX and any generic DVI-to-PS driver to insert the file `pdfparse.psc` into the PostScript output file header. The file `pdfparse.psc` would then contain the above PostScript code with the `/pdfmark` definition.

The PDF code

Now that we have taken care of passing PDF link code, we must create PDF code for creating PDF links. The PDF code works in the same manner as PostScript in that we must create definitions using predefined PDF operators. There are essentially two parts to implementing PDF links. Merz (1997:288–298) gives several examples of fully functional links which can be easily followed. The example

```
[ /Rect [ 0 0 60 60 ]
  /Page 124
  /LNK  pdfmark
```

contains all of the essential elements required for a link to pass through PostScript and be executed in PDF. The PostScript mark `[` and definition `pdfmark` delimit the PDF code. It cannot be assumed that the code between the delimiters is PDF because we may need to pass values from TEX into the PDF code. The fully functioning example

```
\def\pdfbookmark#1#2{{
%#1=section/Chapter/etc..
%#2=usually the title
\special{verbatim=
" [ /Title (#1 #2)
    /OUT pdfmark"}
}}
```

shows how we can define a TEX macro, using a PostScript special, to insert entries into an outline of a PDF document. An example of an outline in Adobe Acrobat is shown in the left side of Figure 1. We can jump to each entry of the document by clicking the icon to the left of the outline entry.

Figure 1: An example of a PDF document containing an outline. We can use the outline to jump to its location in the body of the text.

A more useful but trickier example is:

```
\def\pdfnameddestination#1#2{{
%#1=xrf-tag name
%#2=pagenumber
\special{verbatim=
" [ /Dest   /#1
    /page   #2
    /Border [0 0 0]
    /DEST   pdfmark"}
}}

\def\pdfgotonameddestination#1{{
%#1=xref-tag name
\special{verbatim=
" [ /Rect [ currentpoint
            2 neg add exch
           10 neg add exch
```

```
            currentpoint
            8 add exch
            0 add exch]
  /Border [0 0 0]
  /Dest   /#1
  /LNK    pdfmark "}
}}
```

This example supports dynamic cross-referencing in the PDF document in the same manner as LaTeX. The first macro creates a mark where the source reference is located and the second macro creates a link to that reference. The second macro does this by creating a PDF link box 10 points square in the PDF document. If, in Acrobat Reader, you click on this box, you will jump to the page where the link is defined.

One major restriction of this method is that the entire project needs to be in one PDF document.

Multiple-file PDF documents

Many books and other large projects need to be processed as multiple files, broken up so that each section is of more manageable size. But our earlier method of creating links as named references does not work outside a single file.

A better method, which works over multiple files, is to create a set of macros that reference both the filename and the page number With this method, we can set up a link to any PDF document, as long as we know the file name and the absolute page reference. This method is implemented in the following TeX definition:

```
\def\pdfxrffileopen#1#2{{
%#1= file name
%#2= absolute page number -
% not the printed page number
\special{verbatim=
" [ /Rect [ currentpoint
            2 neg add exch
           10 neg add exch
            currentpoint
            8 add exch
            0 add exch]
    /Border [0 0 0]
    /Action << /Type /Action
      /Subtype /GoToR
      /File (#1.pdf)
      /Dest [ #2 1 neg add /Fit ] >>
    /Subtype /Link
    /ANN pdfmark "}}}
```

This definition contains a few more operators but it has an overall structure similar to the earlier example. It is important to note that PDF documents are referenced by absolute page number. If, for example, the document contains front matter numbered i to xvi and text pages 1 through 23, then the absolute page numbers are 0 for i, 15 for xvi, 16 for 1, up through 38 for page 23. With a little work, one could easily redefine TeX cross-reference and index macros to include file names and absolute page numbers, thereby implementing dynamic and automatic PDF linking. One method for keeping track of absolute page numbering in TeX is to define a new counter and increment it in the TeX output routine.

We can combine the methods of the two previous examples by having the TeX macro `PDFnamed-destination` include another TeX macro which writes each instance, named destination, and filename, to a global output file containing calls to each document file:

```
% Global file containing
% named destinations
\input document1.nd
\input document2.nd
   .
   .
\input documentN.nd
```

In this manner, all files can be accessed somewhat independently and named references for each document can be easily updated. By referencing this file, one can determine in which file a named destination occurs.

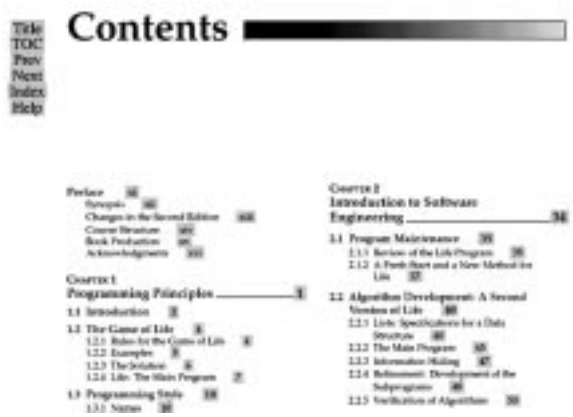

Figure 2: An example of a PDF document containing links from Table of Contents entries to their locations in the main body of the text.

Duplicate named destinations should not occur. Figures 2 and 3 show several examples of how this combination can be implemented to give PDF links. The small shaded boxes represent PDF link boxes; clicking on these boxes will jump the reader to the destination page.

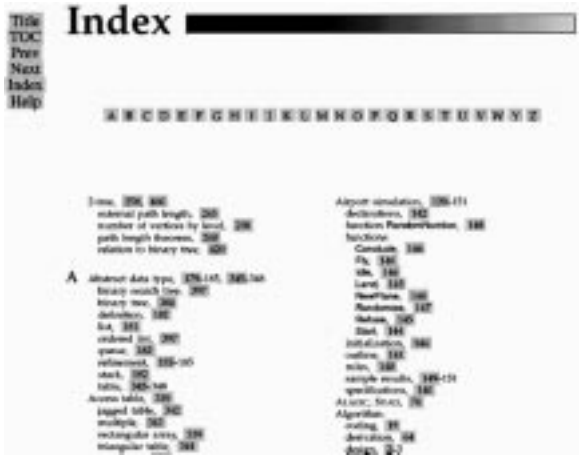

Figure 3: An example of index entries containing links to their locations in the main body of the text.

Launching applications from PDF links

One final useful example of a PDF link is to allow a user to launch an application program while reading the PDF document. The following code allows an executable to be opened directly and related files to be opened using associations:

```
\def\pdflaunchapp#1{{%
% #1 = filename (pathname is optional)
\special{verbatim=
" [ /Rect [ currentpoint
            2 neg add exch
           10 neg add exch
            currentpoint
            8 add exch
            0 add exch]
    /Border [0 0 0]
    /Action << /Type /Action
         /Subtype /Launch
         /File (#1) >>
    /Subtype /Link
    /ANN pdfmark "}}}
```

The TEX example

```
\pdflaunchapp{filename.c}
```

can be used to compile, or to open with an Integrated Development Environment (IDE), files having extensions associated with C compilers. Files such as MS-DOS executables can work, although these files can only be run in a DOS/Windows environment. The shaded boxes containing the uppercase "D" in Figure 4 indicates a PDF link to launch the program `compside.exe`. In general, this type of PDF link can be platform- and operating-system-specific; however, under certain conditions, it can be quite useful. Any link to an executable should be thoroughly tested before being distributed.

Figure 4: An example of launching an application in a PDF file by clicking the shaded boxes containing the "D".

Customized PDFlink radio buttons

We can now create documents which contain dynamically linked cross-references in the form of PDF links. We should now enhance our PDF documents by creating menus or user-defined buttons which will allow warping to different locations of the book. In Figures 1 to 4 we see a menu with such headings as **Title**, **TOC**, **Prev**, **Next**, **Index**, and **Help**; all these represent links to various locations in the book. These are inserted onto each page by including a TEX command in the output routine. The TEX output routine

```
\def\output{\shipout\vbox{\pdftoolbar
\makeheadline
\pagebody
\makefootline}%
\advancepageno
\global\advance\absolutepageno by 1
\ifnum\outputpenalty>-\@MM
\else\dosupereject\fi}
```

is taken from the set of macros used to create the pages in Figures 1 to 4. Note that there is also a reference to absolute page numbering in the sixth line, as we suggested earlier. The TEX macro for the PDF link toolbar

```
\def\pdftoolbar{{
\vbox to 0pt{\hbox to 0pt{\hskip -15pc
\vbox{\hsize=8pc%
\pdftitlepage
\pdftoc
\pdfprev
\pdfnext
\pdfindex
\pdfhelp
}\hss}\vss}}}
```

is quite simple. Depending on its implementation, however, must not affect the remainder of the page body. The TEX macro `\pdftoc`, for example, is simply a line containing a PDF link using the `\pdfxrffileopen` macros.

Acrobat 4

In this paper we have attempted to define a set of TEX macros which will create on-line PDF documents which are platform independent. Since we cannot foresee advancements in software development, with the release of Acrobat 4 and future versions, there remains the potential for better and more visually appealing on-line documents. The examples used in this paper are based on existing PDF features found in the on-line document *Portable Document Format Reference Manual, Version 1.2, November 1997,* by Adobe Systems. This edition is rather old, but one can get the most recent PDF version at the Adobe website `www.adobe.com`.

PreTEX and PDF

PreTEX is a preprocessor and macro package for TEX, developed by PreTEX, Inc., which supplies an author with many tools to simplify the writing and management of larger (book length) projects. (For a discussion of PreTEX, see the article by R. Kruse elsewhere in this issue.) One of PreTEX's important expansions of the resources available to an author is the inclusion of PDF links by using TEX macro definitions similar to the examples above. In this way, a book can be published both on paper and in an electronic form that incorporates automatically generated PDF links for cross-references, index entries, launching application programs, and the like. The current set of PDF macros are not stable enough; they will be made freely available in the spring of next year (1999).

Cheers.

References

Adobe Systems Incorporated. *PostScript Language Reference Manual*. Addison-Wesley, Reading, Massachusetts, 1993.

Adobe Systems Incorporated. *Portable Document Format Reference Manual*. Addison-Wesley, Reading, Massachusetts, 1993.

Hendrickson, Amy. "Real Life LATEX: Adventures of a TEX Consultant." *TUGboat* **19**(2), 162–167 (1998).

Kruse, Robert. "Managing Large Projects with PreTEX: A Preprocessor for TEX." 1999. (See elsewhere in these Proceedings.)

Merz, Thomaz. *PostScript and Acrobat/PDF* Springer-Verlag, Berlin, Heidelberg, 1997.

Taylor, Philip. "Computer Typesetting or Electronic Publishing? New trends in scientific publishing." *TUGboat* **17**(4), 367–381 (1996).

Thanh, Hàn Thế. "Improving TEX's Typeset Layout." *TUGboat* **19**(3), 284–288 (1998).

Thành, Hàn Thế, and Sebastian Rahtz. "The pdfTEX user manual." *TUGboat* **18**(4), 249–254 (1997).

A TEX Haiku

```
\expandafter\def
\csname def\endcsname
{\message{farewell}}\bye
```

—Sebastian Rahtz

A TEXnician's Haiku

This haiku for TEXnicians consists entirely of TEX control sequences; furthermore, it forms a valid TEX assignment statement—provided that a certain control sequence that is normally undefined is defined in an obvious way. *Can you identify the control sequence in question*?

```
\catcode\csname
\romannumeral\parshape
\endcsname\month
```

—Michael Downes

A TEX Cheer

```
Flush to the left
Flush to the right
\vskip, \hskip, type type type
```

Michael Sofka