

# 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](mailto:paul@pretex.com)

## Abstract

This paper discusses how we can create dynamic and interactive online documents in Portable Document Format (PDF), using TeX. PDF documents are generally device and platform independent, therefore, ideally suited for online publishing and information exchange. We will need to work in three programming languages: macros in TeX, and definitions in PostScript and PDF. We begin by describing the steps necessary to process PDF through TeX and PostScript, followed by several examples of defining TeX such 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 [*TUGboat*, Dec. 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 *online* as any means by which a document can be electronically rendered; this is perhaps more aptly called *paperless printing*.

The popularity of online books and documents has spawned a diverse variety of online readers. These *readers*, for our purpose, are programs, applets, and interpreters which can *read* electronic data, and output the image to a screen much the same way one would see the output from a printer. Readers include proprietary single-platform programs, stand-alone multiplatform readers (including Frame Reader, Ghostscript, and Acrobat Reader), WEB browsers (including HTML and JAVA), and WEB browser plugins (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 [PostScript & Acrobat/PDF] 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 plugins 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 [*TUGboat*, June 1998]. Contributions such as `hyperref` and `pdftex` [*TUGboat*, Dec. 1997 and Sept. 1998] provide direct-to-PDF document processing.

---

\* [This article has not yet been edited. – Ed.]

## PDF Through PostScript

For our purpose we will concentrate on the method used by Amy Hendrickson. The description of PDF and PostScript operators for hypertext links is found in [PostScript & Acrobat/PDF] by Thomaz 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 in which PDF code can be passed through both to PDF devices and non-PDF devices.

The *PostScript Language Reference Manual* printed by Adobe Systems contains a complete description of PostScript.

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 include it with the following  $\TeX$  definition:

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

This  $\TeX$  macro tells  $\TeX$ /DVIPS to insert the file `pdfparse.psc` into the output PostScript file header. The file `pdfparse.psc` would then contain the above PostScript code containing 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

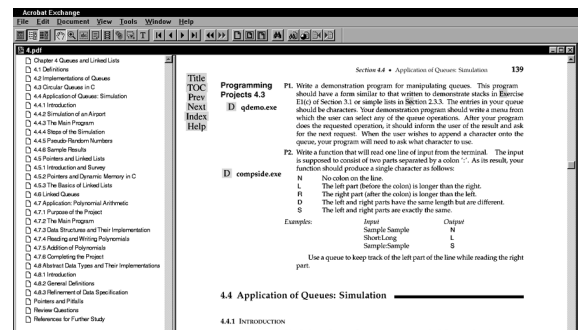
parts to implementing PDF links. Merz gives several examples of fully functional links which can be easily followed. The example

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

contains all of the essential elements required for a link to pass through PostScript and execute 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 L<sup>A</sup>T<sub>E</sub>X. 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 page to reference. This method is implemented in the following T<sub>E</sub>X 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 a similar overall structure as 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 would be easily able to redefine T<sub>E</sub>X 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 T<sub>E</sub>X is to define a new counter and increment it in the T<sub>E</sub>X output routine.

We can combine the methods of the two previous examples by having the T<sub>E</sub>X macro `PDFnameddestination` include another T<sub>E</sub>X 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. 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, and clicking on these boxes will jump to the destination page.

### 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

```

Title	TOC	Prev	Next	Index	Help
<b>Contents</b>					
Preface	xii				
Synopsis	xiii				
Changes in the Second Edition	xiii				
Course Structure	xiv				
Book Production	xv				
Acknowledgments	xvi				
<b>CHAPTER 1</b>					
<b>Programming Principles</b>	<b>1</b>				
1.1 Introduction	2				
1.2 The Game of Life	4				
1.2.1 Rules for the Game of Life	4				
1.2.2 Examples	5				
1.2.3 The Solution	6				
1.2.4 Life: The Main Program	7				
1.3 Programming Style	10				
1.3.1 Names	10				
<b>CHAPTER 2</b>					
<b>Introduction to Software Engineering</b>	<b>34</b>				
2.1 Program Maintenance	35				
2.1.1 Review of the Life Program	35				
2.1.2 A Fresh Start and a New Method for Life	37				
2.2 Algorithm Development: A Second Version of Life	40				
2.2.1 Lists: Specifications for a Data Structure	40				
2.2.2 The Main Program	45				
2.2.3 Information Hiding	47				
2.2.4 Refinement: Development of the Subprograms	48				
2.2.5 Verification of Algorithms	50				

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

Title	TOC	Prev	Next	Index	Help
<b>Index</b>					
A B C D E F G H I J K L M N O P Q R S T U V W X Y Z					
2-tree, 258, 460					
external path length, 265					
number of vertices by level, 258					
path length theorem, 260					
relation to binary tree, 420					
<b>A</b> Abstract data type, 179-185, 345-348					
binary search tree, 397					
definition, 182					
list, 181					
ordered list, 397					
queue, 182					
refinement, 183-185					
stack, 182					
table, 345-348					
Access table, 339					
jagged table, 342					
multiple, 343					
rectangular array, 339					
triangular table, 341					
Airport simulation, 139-151					
declarations, 142					
function RandomNumber, 148					
functions					
Conclude, 146					
Fly, 146					
Idle, 146					
Land, 145					
NewPlane, 144					
Randomize, 147					
Refuse, 145					
Start, 144					
initialization, 144					
outline, 141					
rules, 140					
sample results, 149-151					
specifications, 140					
ALGOL, SIAD, 76					
Algorithm:					
coding, 119					
derivation, 64					
design, 2-3					

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

```

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 MSDOS executables can work, although these files can only be run in a DOS/Windows envi-

ronment. 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, but, under certain conditions, it can be quite useful. Any link to an executable should be thoroughly tested before being distributed.

Title  
TOC  
Prev  
Next  
Index  
Help

**Programming Projects 4.3**

D qdemo.exe

D compside.exe

Section 4.4 • Application of Queues: Simulation 139

P1. Write a demonstration program for manipulating queues. This program should have a form similar to that written to demonstrate stacks in Exercise E1(c) of Section 3.1 or simple lists in Section 2.3.3. The entries in your queue should be characters. Your demonstration program should write a menu from which the user can select any of the queue operations. After your program does the requested operation, it should inform the user of the result and ask for the next request. When the user wishes to append a character onto the queue, your program will need to ask what character to use.

P2. Write a function that will read one line of input from the terminal. The input is supposed to consist of two parts separated by a colon ‘:’. As its result, your function should produce a single character as follows:

N	No colon on the line.
L	The left part (before the colon) is longer than the right.
R	The right part (after the colon) is longer than the left.
D	The left and right parts have the same length but are different.
S	The left and right parts are exactly the same.

Examples:

Input	Output
Sample Sample	N
Short:Long	L
Sample:Sample	S

Use a queue to keep track of the left part of the line while reading the right part.

Figure 4: An example of launching 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 containing **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
```

```
\pdf toc
\pdf prev
\pdf next
\pdf index
\pdf help
}\hss}\vss}}}
```

- [3] Merz, Thomaz, *PostScript and Acrobat/PDF*  
Springer-Verlag, Berlin, Heidelberg, 1997

is quite simple. Depending on its implementation, however, must not affect the remainder of the page body. The  $\text{\TeX}$  macro `\pdf toc`, for example, is simply a line containing a PDF link using the `\pdf xrr file open` macros.

#### Acrobat 4

In this paper we have attempted to define a set of  $\text{\TeX}$  macros which will create online 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 online documents. The examples used in this paper are based on existing PDF features found in the online 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](http://www.adobe.com).

#### Pre $\text{\TeX}$ and PDF

Pre $\text{\TeX}$  is a preprocessor and macro package for  $\text{\TeX}$ , developed by Pre $\text{\TeX}$ , Inc., which supplies an author with many tools to simplify the writing and management of larger (book length) projects. (For a discussion of Pre $\text{\TeX}$ , see the article by R. Kruse elsewhere in this issue.) One of Pre $\text{\TeX}$ 's important expansions of the resources available to an author is the inclusion of PDF links by using  $\text{\TeX}$  macro definitions similar to the examples above. In this way, a book can be published both on paper and in 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 fall of this year (1999).

Cheers

#### References

- [1] Adobe Systems Incorporated, *PostScript Language Reference Manual*, Addison-Wesley, Reading, Massachusetts, 1993
- [2] Adobe Systems Incorporated, *Portable Document Format Reference Manual*, Addison-Wesley, Reading, Massachusetts, 1993