Block-Based Formatting with Encapsulated PDF

Philip N. Smith
Electronic Publishing Research Group,
Department of Computer Science,
University of Nottingham,
Nottingham NG7 2RD, UK
pns@cs.nott.ac.uk

Technical Report NOTTCS-TR-95-1, 2nd January, 1995

Abstract

This report discusses the definition of Encapsulated PDF (EPDF) and an EPDF block formatter called Juggler. PDF (Portable Document Format) is the file format underlying Adobe Systems' Acrobat™ suite of products, and allows documents consisting of 'printed' pages to be distributed and viewed electronically. An EPDF file stores blocks such as headings and paragraphs as distinct objects in such a way that they can be extracted and used in other documents. Juggler is an application which performs this task, using a Document Description File (DDF) to determine the layout of the new document. Of particular interest is Juggler's method of *visual splitting* of blocks across columns and pages.

After describing EPDF and Juggler in their current states, the report moves on to discuss further extensions which could be made, including document structure representation and inter-block linking. Brief mention is made of relationships between EPDF and the standards SGML and ODA.

Contents

The main text of this document was produced using LaTeX. Figures were generated by hand-written PostScript. Pages 10 and 11 and also those for the appendices were left blank before conversion to PostScript with dvips and PDF with Acrobat Distiller. The blank pages were then replaced with the correct pages of PDF using Acrobat, and the complete document was printed to file to produce the final PostScript.

1 Introduction

Juggler is an application which can be used to generate PDF[3] documents from 'Encapsulated PDF' (EPDF) blocks in other PDF documents. The theory behind EPDF and its use is currently further developed than the Juggler application itself. This report outlines the general principles of using EPDF, illustrates the use of Juggler with a simple example and then discusses further ideas and possible future developments. A working knowledge of PDF file structure and syntax is assumed. For the sake of readability, PostScript equivalents are mentioned in place of certain PDF operators.

2 EPDF File Structure

An EPDF file, just like an ordinary PDF file contains a number of pages, each with its imageable contents referenced under its Contents key. The main difference at this level, is that whereas a page is usually a reference to one contents stream, EPDF makes use of PDF's ability to hold an array of references to contents streams. Each imageable stream is an EPDF block, and references to these are interleaved with references to positioning objects which place them correctly on the page. Although they are all stored in a single array, each pairing of positioning object with imageable block can be regarded as a leaf of the PDF page tree.

2.1 EPDF Blocks

The EPDF block is the basis of Juggler's functionality. It is essentially a Contents stream with a few extra keys. Instead of the stream imaging a whole page, however, it just shows something like a paragraph, section heading or diagram. Below is an example of an EPDF block.

```
11 0 obj
<<
/Type /EPDFBlock
/Subtype /sectionhead
/BoundingBox [ 90 720 184 732 ]
/Resources <<
  /Font << /F2 10 0 R >>
  /ProcSet [ /PDF /Text ]
>>
/Length 12 0 R
>>
stream
BT
0 Tr
0 g
/F2 1 Tf
14 0 0 14 90 722.95 Tm
0 Tc
0 Tw
[(1)-994.3(Backgr)17.1(ound)]TJ
ET
endstream
endobj
12 0 obj
89
endobj
```

Type is self-explanatory. Subtype's value can be any valid name. I have not set out a 'standard' set of subtypes, and I will mention in this report strong arguments for not doing so. The Resources are

identical to those that would appear in the Page object if this were the only contents stream, with the possible exception that each top-level resource array or dictionary must be direct (e.g. /ProcSet20R is not allowed). Juggler uses the bounding box of the block to set up a translation to place it correctly on the page, just as a DTP application places Encapsulated PostScript[2].

2.2 Building a Page of Blocks

Juggler performs the following steps when adding a block to a page (though not in this order, and not necessarily atomically):

- Copy the block to the new file, adding its resources to those already required by the page. This requires renumbering of the object, copying and renumbering of any objects it references, and sometimes *renaming* of resources to avoid name clashes. This problem is due to the fact that resource names must be declared in the Page rather than locally to each Contents stream, and is discussed further in section 5.1.
- Work out where the block needs to appear on the page and create a new object to position it. This consists of a grestore to restore to the state before the last positioner, a gsave and a translate to position the block. Thus, each translation is relative to the origin before the first block was imaged. This is usually the lower left corner of a *frame* (see later sections).
- Add references to the positioner and block to the Contents array.

Note that this is a simplification of the process. Variations on this are mentioned in later sections.

2.3 Repeated Blocks

If a block such as a logo is to be imaged on more than one page, or indeed more than once on the same page, it need appear in the file only once. It is simply referenced more than once, each time preceded by a suitable positioner. Not only does this save filespace, but it is also an important part of the block splitting method described in the next section.

3 Page layout

Apart from the source PDF files, another input to Juggler is a kind of style file, here referred to as a Document Description File (DDF). This states which blocks to use and how to lay them out. It includes a description of layout frames, inter-block spacing and how to split blocks between frames.

3.1 Visual Splitting

Just as a DTP application is not expected to parse EPS, Juggler does not attempt to parse block streams to understand where lines of text are and where to split the block. Instead, the DDF states in a generic way where a block of a given type may be visually split.

As an example of the process, consider breaking a paragraph (say /Type/para) at the end of a column. Juggler finds out from the DDF the best split point, and then, as well as doing a translate to get the block in the correct place, also does a crop so that anything below the split point is invisible. At the top of the next column, the other half of the block will be cropped out. The fact that the document contains two views of the same logical block is reflected in the resulting PDF file; two crop boxes are set up, each followed by a reference to the same EPDF block. This compares favourably with the Open Document Architecture (ODA)[4] in which a content portion would have to be split into two parts in the formatting process.

Splits is a subdictionary of a DDF. Its key–value pairs are block types and arrays from which split points can be calculated. For example

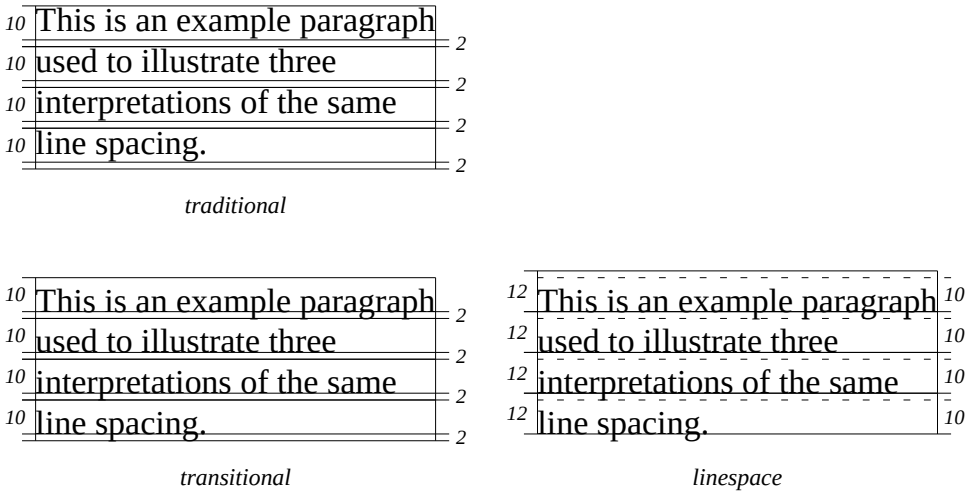

Figure 1. Three models of typeset line placement

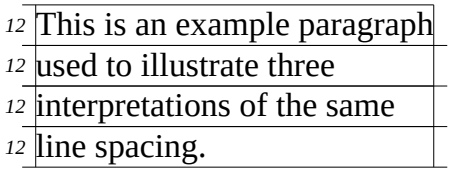

Figure 2. Bounding box position used with Juggler

```
/Splits <<
  /para [ 24 12 36 ]
>>
```

says that a paragraph can be split at any 12pt interval between 24pts from the top and 36pts from the bottom. This would be appropriate for a 10/12pt paragraph, and has the side effect of avoiding widows and orphans. If no valid split point can be found (e.g. if the paragraph has only three lines) or none is specified (items such as diagrams and tables are usually non-splittable), the block is not split.

All this means that bounding boxes must be calculated very carefully. Figure 1 illustrates how the calculated bounding box of a paragraph might vary according to the typesetting model being employed. The traditional model is where 10pt sorts are made up into lines and 2pts of lead are inserted between them. The transitional model is a Type1-ised version where extra leading is thought of as being directly below the text baseline. In the linespace model lines are simply set with baselines 12pts apart, and actual text size is of little importance.

A bounding box calculated using just the information in the linespace model would not include descenders, so splitting the block at simple 12pt intervals would chop them off. The transitional model would work *in this case*, but a font with descenders longer than the leading would still get chopped. The bounding boxes of the blocks used in the example in section 4 were calculated by taking the bounding box of the baselines, subtracting a suitable amount, n, from the bottom and adding $12 - n$ to the top. This results in a box a multiple of 12pts high which can be split at any 12pt interval without chopping off ascenders or descenders (see figure 2).

3.2 Document Description File (DDF)

A DDF contains two major parts: a dictionary containing layout information and a list of the blocks required. Juggler creates pages as necessary and adds blocks to the current page in the order in which they appear in the DDF. It is planned to identify blocks by some kind of unique naming scheme (UniqueID would be another key in the block), but at the moment the names in the block list are of the form */n:m* where *n* is the 'file number' and *m* is the object number within that file. The rest of the style file is best described with the aid of an example.

4 Using Juggler — A Worked Example

4.1 The DDF

The DDF below describes a simple two-column document. Blocks are taken from two files, called `maindoc.pdf` and `anotherdoc.pdf`, in subdirectory `pdf`. There are blocks of types titleblock, head, subhead and para.

```
<<
  /Files [ (pdf/maindoc.pdf) (pdf/anotherdoc.pdf) ]

  /MediaBox [ 0 0 570 450 ]

  /StartRightPage [
    /FrameA /None 80 50 290 400 createframe
    /FrameB /None 320 50 530 400 createframe
    /FrameA setcurrentframe
  ]

  /StartLeftPage [
    /FrameA /None 40 50 250 400 createframe
    /FrameB /None 280 50 490 400 createframe
    /FrameA setcurrentframe
  ]

  /AddBlock <<
    /DEFAULT [ /CURRENT putblock ]
  >>

  /EndFrame <<
    /FrameA [ /FrameB setcurrentframe ]
    /FrameB [ showpage ]
  >>



  /Spacing <<
    /titleblock <<
      /head 21.5  % gets back on to 12pt grid
    >>
    /para <<
      /para 0
      /head 24
      /subhead 24
    >>
```

```
    /subhead <<
      /para 12
    >>
    /head <<
      /para 12
      /subhead 12
      /head 12
    >>
  >>

  /Splits <<
    /para [ 24 12 24 ]
  >>
>>

/1:7 /1:11 /1:14 /2:8
/1:17 /1:20 /1:24 /1:28
/1:32 /2:8 /1:35 /1:38
/1:41 /1:44 /1:48
```

The syntax of the main dictionary is the same as for PDF dictionaries. The only quirk is that some arrays, such as the Start...Page arrays, are executable (procedures), while others, such as MediaBox are just PDF-style arrays containing no operators. All the top-level names are standard, and some entries (Files, StartRightPage, AddBlock and EndFrame) must be present. Arithmetic operators add, sub, mul and div may be used in any procedure, as well as standard stack operators such as exch, dup, pop, copy and roll. Other operators are defined for particular procedures.

MediaBox This plain array is copied over into the root Pages object in the new file. If it is not specified, a default of [00595842] (A4 size) is used.

StartRightPage This is a procedure to be executed when a right-hand (usually odd-numbered) page is begun. The only permitted non-standard operators are createframe and setcurrentframe. createframe takes a frame name and four numbers for a bounding box as its parameters. The None name is currently ignored, but may have significance in a later version (see section 5.6.4). setcurrentframe sets the frame to be regarded as the current frame. Blocks may be added to a frame with the reserved name CURRENT, and the current frame may be changed. This permits flow from one frame to another. The StartRightPage procedure must create at least one frame, and must set the current frame.

StartLeftPage The semantics of this procedure are identical to StartRightPage, except that it defines the frame layout for left-hand (even-numbered) pages. If omitted, all pages use the StartRightPage procedure.

AddBlock Each key in the AddBlock dictionary is a block type name (e.g. /para) or the reserved name DEFAULT. Each value is a procedure which is executed when a block of the key type has to be added to the page. At the time of writing, the only allowable operator is putblock. This puts the block into the named frame. If there is no entry for a particular block type, the DEFAULT entry is used.

EndFrame Each key in the EndFrame dictionary is a frame name. The procedure associated with a frame name is executed as soon as that frame becomes full. The procedures may set the current frame or execute showpage which causes the page to be added to the output file and a new page to be started if necessary.

Spacing The Spacing dictionary specifies the amount of extra space which should be added between two blocks. If a block of type a is to be followed by a block of type b the extra space is given by the value

associated with key b in the subdictionary associated with key a. Values are specified in points. In the example DDF, if a para is to be followed by a head, 24pts of extras space must be left between them. If a head is to be followed by a para, 12pts must be left. If no extra space is specified for a particular pairing, no extra space is left.

Splits This specifies the split points (see section 3.1) for each block type. If none is specified, the block will not be split.

After the dictionary comes the list of block identifiers which was described in section 3.2. Note that in this example, nearly all the blocks come from `maindoc.pdf`. However, a block from `anotherdoc.pdf` is inserted twice into the new document.

4.2 Running Juggler

Juggler is a stand-alone, command-line program written in C++. It has been compiled on a Sun4, and takes as its arguments the name of the DDF and the name of the PDF file to be produced. The two PDF input files have been included in this document in appendices A and B. A transcript of the juggling process is shown below, and the resulting file is shown on pages 10 and 11.

```
pns@quill> juggler example.ddf example.pdf

---CREATING PAGE 1---
Splitting block

WRITING PAGE 1

Writing frame 1
WriteBlock(), objid 7 0
WriteBlock(), objid 11 0
WriteBlock(), objid 14 0
WriteBlock(), objid 8 0
Renames required: <<
/F2 /JugF2
>>

Writing frame 2
WriteBlock(), objid 17 0
WriteBlock(), objid 20 0
WriteBlock(), objid 24 0
WriteBlock(), objid 28 0
WriteBlock(), objid 32 0
Renames required: <<
/F1 /Jug1F1
>>

---CREATING PAGE 2---
Splitting block

WRITING PAGE 2

Writing frame 1
WriteBlock(), objid 32 0
WriteBlock(), objid 8 0
WriteBlock(), objid 35 0
WriteBlock(), objid 38 0
```

```
Writing frame 2
WriteBlock(), objid 38 0
WriteBlock(), objid 41 0
WriteBlock(), objid 44 0
WriteBlock(), objid 48 0
pns@quill>
```

Each `WriteBlock()` message corresponds to the placing of a block on the page. The `objid` is the object identifier from the original file. The first time a `WriteBlock()` message appears for a block, that block is copied to the new file. Each subsequent time it is merely referenced from the Contents array in the Page object. The transcript shows that object 32 is split between page 1 and page 2 and that object 38 is split between frame 1 and frame 2 on page 2.

The `Renames required:` message appears when the resources required by a block have to be renamed in order to avoid name clashes with blocks already added to the page. This process is described in more detail in section 5.1.

Second Year Ph.D. Report
Juggler: State of the Act

Philip N. Smith

Autumn 1994

1 Background

In my first year report I reviewed various methods of document representation and discussed the possible use of Adobe's Portable Document Format (PDF) as a common format for revisable multiple-source documents. Much of the discussion was of how it might be possible to enable block level editing of PDF documents which contained no block information whatsoever. While this is a worthwhile goal, I have come to the conclusion that a far more useful and useable system could be based on a format including a little more information. This report is a working document discussing technical details as well as general principles, and assumes some knowledge of PDF and Acrobat.

This is a paragraph taken from another EPDF file. I've set it in bold type just to make it easier to find, but it is of type /para **just like all the other paragraphs in this document.**

2 Introducing Juggler

Juggler is an embryonic system for laying out 'Encapsulated' PDF (EPDF) blocks onto pages. An EPDF block (a heading or paragraph, for example) is in the same form as a PDF content stream, except for the addition of a few extra keys. The working definition of EPDF is such that an EPDF file is also a valid PDF file. However, the extra information allows Juggler to extract individual blocks from any number of files and to lay them out onto new pages. At all stages, the design of EPDF attempts to be 'Acrobat-friendly' i.e. it should be possible to generate EPDF without too many alterations to existing software such as Distiller.

In general terms, an EPDF file consists of a large set of blocks, and a set of pages which specify *views* of those blocks in particular locations. If a block must be shown on several pages or several times on the same page, it need appear in the file only once; it is just viewed several times.

2.1 A pointless subsection

To implement this in a valid PDF file, heavy use is made of the fact that PDF allows the contents of a page (what you see) to be described by any number of streams of page description commands. Though

in most PDF documents the value of the /Contents key is a reference to a single stream of commands, it may also be an array of references. In EPDF each page contains such an array. The referenced objects are alternately views specifiers and imageable blocks. The view specifiers usually contain coordinate transformation commands to position the next block correctly on the page.

This is a paragraph taken from another EPDF file. I've set it in bold type just to make it easier to find, but it is of type /para just like all the other paragraphs in this document.

Once created, a block never changes. It is given a unique identifier and can be copied into other files. Even when a block has to be split between columns or pages, Juggler maintains the integrity of the block. Two distinct views of the same block are defined using the clipping path operators. In one view the lower portion becomes invisible, and in the other the upper portion is clipped out. This means that the logical block structure of the document is maintained independently of the particular layout. This compares favourably with ODA in which a logical block must contain two distinct content portions if it is to be split between two layout blocks.

Hypertext links are an important feature of PDF but link simply from one area of a page to another. In EPDF a block can contain links to other blocks, and these can be converted to ordinary PDF links for a specific layout. Interestingly, this creates exactly the same kind of forward referencing problems for Juggler as have been found in work on the CAJUN project: if a block has a link to another block, that link can't be made until it is known whether and where the target block appears.

3 Conclusion

Although the Juggler application is still at an early stage of development, it is easy to see that use of EPDF gives rise to many possibilities. I hope to investigate some of these possibilities over the next year, through further development of Juggler.

There will come a point, however, when the main benefit of using PDF — the fact that almost anyone can produce it, using any application — is outweighed by the extra work involved in using it for purposes other than those for which it was originally intended: purposes for which other systems have already been developed.

5 Further Discussion

This section discusses various topics relating to Juggler as it is now and as it might be. I have attempted to arrange them into some kind of order, but opinions may differ as to what kind of order this is.

5.1 Resource Renaming and the Page Objects

PDF has been designed to make display and browsing of documents as efficient as possible, but generation is not quite so simple. When moving an object from one file to another it has to be given a new number before writing. All objects to which it refers must also be renumbered and copied.

The greatest problem for Juggler, however, is the real likelihood of resource name clashes: the named resources required by each block on the page must be listed in the Page object's Resources dictionary. A block from one file might refer to Times Roman as F1 while a block from another might use F1 to refer to Helvetica. If the two are to be placed on the same page by Juggler, a new name needs to be allocated in place of one of the F1s, and every occurrence of F1 in the block's stream must be changed.

In the example in section 4 the block from `anotherdoc.pdf` has its resource name F2 changed to JugF2 to avoid a name clash with an earlier block (see transcript on page 8). This is the block as it appears in the original file:

```
8 0 obj
<< /Length 9 0 R /Type /EPDFBlock /Subtype /para /BoundingBox [ 89.99
98 684.2 296.393 732.2 ]
/Resources << /Font << /F2 6 0 R /F1 7 0 R >> /ProcSet [ /PDF /Text ]
 >>
 >>
stream
BT
0 Tr
0 g
/F2 1 Tf
10 0 0 10 100.08 722.95 Tm
0 Tc
0 Tw
[(This)-216(is)-240(a)-216(paragraph)-264(taken)-240(fr)24(om)-240(an
other)-240(EPDF)]TJ
-1.008 -1.2 Td
[(\256le.)-480(I've)-288(set)-288(it)-264(in)-288(bold)-264(type)-288
(just)-264(to)-288(make)-288(it)-264(easier)]TJ
0 -1.2 Td
[(to)-192(\256nd,)-240(but)-192(it)-192(is)-192(of)-192(type)]TJ
/F1 1 Tf
9.552 0 Td
(/para)Tj
/F2 1 Tf
2.472 0 Td
[(just)-216(like)-192(all)-168(the)-216(other)]TJ
-12.024 -1.2 Td
[(paragraphs)-264(in)-240(this)-240(document.)]TJ
ET
endstream
endobj
```

and here it is after being copied to `example.pdf`:

```
22 0 obj
<<
/Length 25 0 R
/Type /EPDFBlock
/Subtype /para
/BoundingBox [ 89.9998 684.2 296.393 732.2 ]
/Resources <<
/Font <<
/JugF2 23 0 R
/F1 24 0 R
>>
/ProcSet [ /PDF /Text ]
>>
>>
stream
BT
0 Tr
0 g
/JugF2 1 Tf
10 0 0 10 100.08 722.95 Tm
0 Tc
0 Tw
[ (This) -216 (is) -240 (a) -216 (paragraph) -264 (taken) -240 (fr)
24 (om) -240 (another) -240 (EPDF) ] TJ
-1.008 -1.2 Td
[ (\256le.) -480 (I've) -288 (set) -288 (it) -264 (in) -288 (bold)
-264 (type) -288 (just) -264 (to) -288 (make) -288 (it) -264 (easier)
 ] TJ
0 -1.2 Td
[ (to) -192 (\256nd,) -240 (but) -192 (it) -192 (is) -192 (of) -192
(type) ] TJ
/F1 1 Tf
9.552 0 Td
(/para) Tj
/JugF2 1 Tf
2.472 0 Td
[ (just) -216 (like) -192 (all) -168 (the) -216 (other) ] TJ
-12.024 -1.2 Td
[ (paragraphs) -264 (in) -240 (this) -240 (document.) ] TJ
ET
endstream
endobj
```

Notice how the whole stream is completely re-written; Juggler reads it a token at a time and changes names as necessary. Note also that this process is performed once only per block. The renaming is done as the block is written to the new file for the first time. It is possible that if the block is asked for again, later in the document, further renamings might be necessary. However, by that stage the block will already have been written out. At the time of writing, if such a combination of events occurs, Juggler halts, complaining that it can't perform the necessary renamings. It would, however, be quite possible to write a second copy of the block, with different names. This would at least result in a complete document, even if it wouldn't be as neat as one containing just one copy of each required block.

A final point about resource renaming is that there is no need to worry about changing the actual resource. For example, the font in object 60 in the original file contains /Name/F2. Although the PDF

Reference Manual states that this name should match the name in the Page resources, it serves no useful purpose and is ignored by all Acrobat PDF viewers. This means that the object can be copied through to the new file without changing that entry. More importantly, it also means that two blocks can refer to the same resource by different names. This often happens when two paragraphs from the same source document are put into a new document, but one of them has to have a resource name changed. In such a case, the Page resources will contain both names, each associated with the same resource object.

For the sake of completeness, here are the two Page objects from `example.pdf`:

```
52 0 obj
<<
/Type /Page
/Parent 3 0 R
/Contents [ 4 0 R 6 0 R 8 0 R 11 0 R 13 0 R 16 0 R 18 0 R 20 0 R
22 0 R 26 0 R 27 0 R 29 0 R 31 0 R 33 0 R 35 0 R 37 0 R 39 0 R 42 0 R
44 0 R 46 0 R 48 0 R 51 0 R ]
/Resources <<
/ProcSet [ /PDF /Text ]
/Font <<
/F4 9 0 R
/F2 14 0 R
/JugF2 23 0 R
/F1 24 0 R
/F3 40 0 R
/Jug1F1 49 0 R
>>
>>
>>
endobj

85 0 obj
<<
/Type /Page
/Parent 3 0 R
/Contents [ 53 0 R 55 0 R 48 0 R 57 0 R 22 0 R 59 0 R 61 0 R 63 0 R
65 0 R 67 0 R 68 0 R 70 0 R 65 0 R 72 0 R 74 0 R 76 0 R 78 0 R 80 0 R
82 0 R 84 0 R ]
/Resources <<
/ProcSet [ /PDF /Text ]
/Font <<
/F4 9 0 R
/Jug1F1 49 0 R
/JugF2 23 0 R
/F1 24 0 R
/F2 14 0 R
>>
>>
>>
endobj
```

Object 22 is the paragraph from `anotherdoc.pdf` and appears on both pages. Object 48 is the paragraph split from page 1 to page 2 and so appears once in each Contents array. Object 65 is the paragraph split between frames on page 2 and so is referenced twice in the same array. Font F3 which appears only on page 1 is Times Italic, used for the word *views* halfway down the second column.

5.2 Block Reuse

Having 'juggled' some files into a new file, the blocks in that new file can be made available to another juggler process in just the same way as the originals. To all intents and purposes, a block is unchanged when moved from one file to another. The only differences are object numbers and possibly resource names, but these have no effect on the appearance of the block. Block reuse would be simpler if each had a unique identifier. Then a block would be identified in the same way in all files, and the author of a DDF would not have to worry about finding the object number of the block in the particular file from which it was to be extracted.

5.3 Unique IDs and Distributed Documents

A combination of original file ID (PDF-1.1) and object number in that file could be used as an almost unique, if not particularly attractive, identifier.

As mentioned above, giving blocks unique identifiers would be a major step forward in easing the creation of DDFs. Better still would be to have a separate 'block server' application or module to search available files for a given block. There would then be no need for a Files entry in the DDF and blocks would be supplied by the block server from the most convenient source. With the use of suitable protocols, blocks could even be acquired from remote sites if they were unavailable locally.

If this were implemented carefully, the DDF could be regarded as the document and the creation of a complete PDF file a means of display. People could exchange DDFs rather than PDFs knowing that as long as the EPDF blocks existed somewhere, the document could be displayed.

5.4 Structure Hierarchies

Although each block in an EPDF file may be regarded as a structural element, there is no statement of the way in which these elements are arranged into a structured document. We have PostScript's descriptive power of appearance and a simple DTP application's notion of element types, but nothing like the ability of SGML[1] to describe the structure of a document. Discussion of structured documents is here confined to hierarchical, tree structures such as this report.

If we add an optional Kids entry to the EPDFBlock specification, we have a basis for defining tree structures. Consider the common structure consisting of a section containing subsections containing paragraphs. Rather than having block types sectionhead, subhead and para, we can call them section, subsection and para. The Kids entry in a section would be an ordered array of unique identifiers for its subsections. What we might call the *imageable part* of the block would be the section heading, and the layout rules would be the same as before, but the section would in some way 'own' its subsections. Similarly, a subsection's kids would be paras.

The list of blocks in the DDF could be greatly reduced by allowing preorder traversal of such trees. Asking for a section (as opposed to a sectionhead) would (or at least could, if requested) result in all the blocks of the section being added to the new document, in the correct order.

Various issues need to be addressed. For instance, how easy would it be for Juggler to build these trees from a list of blocks? Some notation for the generic document structure would be required. The process would either be two-pass (one pass to work out the structure and the second to write out the blocks with their Kids entries) or would require many blocks to be held in memory.

Further, if a section is extracted from a document and another subsection is added, which, if any, of the unique identifiers should be changed? Though its imageable part remains the same, the structure of the section has changed. Therefore it should be given a new unique identifier. It's original subsections, however, have not changed and should therefore keep their existing identifiers.

Though clearly not as powerful a representation method as generalised markup languages such as SGML, the use of Kids entries would at least allow hierarchical document structures to be simply stored in a PDF file.

5.5 Correctness Checking with the Spacing Dictionary

As aforementioned, if there is no entry in the Spacing dictionary for a particular ordered pair of blocks, no extra space is inserted between them. However, this dictionary, which forms part of the description of what the document looks like, can also be used to help check that it is structurally correct. For instance, in the example in section 4 no spacing is specified for the pair subheadhead. If this combination was encountered, Juggler could give a warning that there might be an inconsistency.

Though limited in power, this form of correctness checking requires no additional data. In particular, no generic structure definition is needed.

5.6 DDF Improvements

DDF functionality is at the moment fairly limited, particularly in its ability to lay out occasional material such as footnotes. This section mentions a few possible improvements.

5.6.1 Headers and Footers

While headers and footers could be defined block types and be placed in special frames, there is little point in doing this as the content, and number of header and footer lines depends entirely on the particular pagination of the document being generated. It should be Juggler's job to generate headers and footers from data supplied in the DDF

5.6.2 Justification

It would be nice to be able to right-justify or centre a narrower block within a frame. For example, the title block in the example on page 10 ought to be centred. At the moment, all blocks are left-justified within the frame.

5.6.3 Repeating Content

It would also be nice to be able to specify a block, such as a logo or textured background, that should appear on every page. This could perhaps be defined in the Start. . . Page procedures.

5.6.4 Expanding Frames

Expanding frames are one way of handling occasional material such as footnotes. An expanding frame would be created with zero height to begin with, but one argument to createframe (which is currently always None) would give the direction (/Up or /Down) in which the frame should expand. The AddBlock procedure for footnotes would add the footnote to the expanding frame causing it to expand, and would call another procedure to adjust existing frames. If adjusting the other frames would cause blocks to require repositioning, all the page's blocks would have to be relaid, this time starting with the frames at their revised sizes.

With careful thought, this additional functionality would allow quite complex frame layouts. Other possibilities are conditionally-created frames, and even the ability to switch between completely different frame sets.

5.7 Non-Strict Bounding Boxes

Bounding boxes in EPDF need not be strict. They are used to position blocks and for visual splitting. The most important thing is that they are defined consistently so that all blocks of a particular type can be treated in the same way. In the earlier example, paragraphs are set 10/12pt and the bounding box is always a multiple of 12 points high. It always extends the same distance below the bottom baseline and 12 points less that distance above the top baseline.

What might not be so obvious is that the bounding boxes of headings and subheadings are also 12 points high. In the case of section headings, this probably means that the larger text extends above the top of the

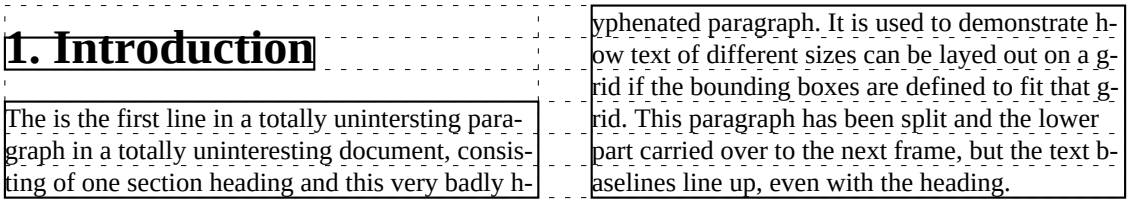

Figure 3. Laying out blocks to a grid using non-strict bounding boxes

box, but this doesn't matter as section headings are never cropped. The advantage is that everything can be laid out on a 12 point grid, ensuring that the lines in the two columns do line up. This is illustrated in figure 3

5.8 Reformatting Blocks

Although juggler allows any size of block to be added to any size of frame, there is currently nothing it can do about blocks which aren't the same width as the others. It could be argued that in EPDF a paragraph with a 10 pica measure is a different kind of block from one with a 30 pica measure. However, since the two are so similar in other respects, the ability to reformat a block which is known to be a paragraph of some kind, would add greatly to Juggler's usefulness. One could, of course, extract the text using the text extraction tool in Acrobat and regenerate the paragraph with an EPDF-aware application, but it would be better to have this done automatically as part of Juggler's layout process.

This raises the philosphical question of whether the block should retain its original unique identifier after reformatting. If reformatting really is transparent, then it makes no difference to Juggler what measure the block has, and so measure is not a distinguishing feature. It would then be analogous to resource names — it doesn't matter what the names are, so long as the resources are the same in all other respects.

On the other hand, the rest of Juggler deals only with bounding boxes and split points which can be applied just as well to pictures as to paragraphs. If the size of a picture or a paragraph changes then it is a different block. Also, reformatting is unlikely to be transparent because of problems caused by hyphenation.

5.9 Textual References

One major deficiency in the Juggler–EPDF system is its inability to resolve textual references such as 'see section 6' commonly found in printed documents. There is nothing in a block to say that it contains such a reference, let alone where the text is or to what it refers. Adding such information would not be straightforward and would not fit nicely with the otherwise wholly graphical nature of the formatting. If a reference had to be changed (for example to 'see section 10') a paragraph might require reformatting.

Though in strictly-planned documents where all sectional numbers are known in advance, this might not be too much of a problem, such textual references should be avoided if blocks are to be reused in other contexts. More use should be made of hypertext links, and in particular, inter-block links.

5.10 Inter-Block Links

The last topic in this pot-pourri of a section is that of hyper-links between blocks. Links in an ordinary PDF document link from an area of a page to a view of a page. In EPDF they should really be defined as linking from an area of a block to another block. When a block containing a link is put into a new document, an equivalent PDF-style link should be added to the Page resources.

The EPDF form of a link would have a Rect key like PDF's, specifying the location of the link box in the same coordinate system as the block's bounding box. The Dest entry would just be the unique identifier of the destination block. This data would have to be transformed into a new Link object for the new PDF file. The problems of forward references are similar here to those experienced on the CAJUN project[5], with the added complication that the destination block might not even appear in the new file.

6 Generating EPDF

Clearly some information required in EPDF cannot be written by current versions of Distiller or PDFWriter and so has to be added afterwards using a separate program. The only remaining problem is to generate separate objects for each block. This is accomplished by placing each on a separate page.

The blocks used in the example in section 4 were generated using LaTeX with a few macros revised to begin sections and paragraphs on separate pages, and to pass through the block types. An addition to the PostScript prolog which redefines `show` to keep track of baseline positions and string widths causes Distiller to print to standard output the bounding box and type of each block.

7 Final Remarks

The design of EPDF aims to enable block-based reformatting of PDF documents with minimal changes to existing PDF-generating software. Juggler is an application which performs this task. This report has described the current state of EPDF and Juggler, and also given a number of pointers to possible future developments.

In PDF, nothing is known of what a block is; just what it looks like. Through the simple addition of a type field for a block, Juggler can treat different objects in different graphical ways. However, it still doesn't know what the blocks are: it just knows where to put them. Any attempt to type blocks more strongly, to create a set of 'standard' types (for tables, figures, graphs etc.) and to insert semantic information would make PDF more difficult to generate and would make the format specific to certain types of content. At the moment, PDF can represent *any* kind of printable material. It should remain that way.

The types in EPDF can be compared with the elements in an SGML document, the DDF with the DTD. Types will be standard for a particular set of documents or particular project, but should not be standardised for EPDF.

PDF, like print, is considered a final form of a document. It can be generated by anything which can print. Juggler and EPDF go some way towards enabling reuse of this final form.

References

[1] Charles F. Goldfarb. *The SGML Handbook.* Oxford University Press, 1990.

[2] Adobe Systems Inc. *PostScript Language Reference Manual*. Addison-Wesley, Reading, Massachusetts, second edition, December 1990.

[3] Adobe Systems Inc. *Portable Document Format Reference Manual.* Addison-Wesley, Reading, Massachusetts, June 1993.

[4] ISO/DIS 8613 Information processing. *Office Document Architecture (ODA),* 1986.

[5] Philip N. Smith, David F. Brailsford, David R. Evans, Leon Harrison, Steve G. Probets, and Peter E. Sutton. Journal publishing with acrobat: the cajun project. *Electronic Publishing—Origination, Dissemination and Design*, 6(4):481–493, December 1993. Proceedings of the Fifth International Conference on Electronic Publishing, Document Manipulation and Typography (EP94).

A maindoc.pdf

The next page shows all the blocks from `maindoc.pdf`. For reasons described in section 6 the file itself contained one block per page. For reasons of economy and conservation, Juggler was used to put them onto one page for this appendix.

Second Year Ph.D. Report
Juggler: State of the Act

Philip N. Smith

Autumn 1994

1 Background

In my first year report I reviewed various methods of document representation and discussed the possible use of Adobe's Portable Document Format (PDF) as a common format for revisable multiple-source documents. Much of the discussion was of how it might be possible to enable block level editing of PDF documents which contained no block information whatsoever. While this is a worthwhile goal, I have come to the conclusion that a far more useful and useable system could be based on a format including a little more information. This report is a working document discussing technical details as well as general principles, and assumes some knowledge of PDF and Acrobat.

2 Introducing Juggler

Juggler is an embryonic system for laying out 'Encapsulated' PDF (EPDF) blocks onto pages. An EPDF block (a heading or paragraph, for example) is in the same form as a PDF content stream, except for the addition of a few extra keys. The working definition of EPDF is such that an EPDF file is also a valid PDF file. However, the extra information allows Juggler to extract individual blocks from any number of files and to lay them out onto new pages. At all stages, the design of EPDF attempts to be 'Acrobat-friendly' i.e. it should be possible to generate EPDF without too many alterations to existing software such as Distiller.

In general terms, an EPDF file consists of a large set of blocks, and a set of pages which specify *views* of those blocks in particular locations. If a block must be shown on several pages or several times on the same page, it need appear in the file only once; it is just viewed several times.

2.1 A pointless subsection

To implement this in a valid PDF file, heavy use is made of the fact that PDF allows the contents of a page (what you see) to be described by any number of streams of page description commands. Though in most PDF documents the value of the /Contents key is a reference to a single stream of commands, it may also be an array of references. In EPDF each page contains such an array. The referenced objects are alternately views specifiers and imageable blocks. The view specifiers usually contain coordinate transformation commands to position the next block correctly on the page.

Once created, a block never changes. It is given a unique identifier and can be copied into other files. Even when a block has to be split between columns or pages, Juggler maintains the integrity of the block. Two distinct views of the same block are defined using the clipping path operators. In one view the lower portion becomes invisible, and in the other the upper portion is clipped out. This means that the logical block structure of the document is maintained independently of the particular layout. This compares favourably with ODA in which a logical block must contain two distinct content portions if it is to be split between two layout blocks.

Hypertext links are an important feature of PDF but link simply from one area of a page to another. In EPDF a block can contain links to other blocks, and these can be converted to ordinary PDF links for a specific layout. Interestingly, this creates exactly the same kind of forward referencing problems for Juggler as have been found in work on the CAJUN project: if a block has a link to another block, that link can't be made until it is known whether and where the target block appears.

3 Conclusion

Although the Juggler application is still at an early stage of development, it is easy to see that use of EPDF gives rise to many possibilities. I hope to investigate some of these possibilities over the next year, through further development of Juggler.

There will come a point, however, when the main benefit of using PDF — the fact that almost anyone can produce it, using any application — is outweighed by the extra work involved in using it for purposes other than those for which it was originally intended: purposes for which other systems have already been developed.

B anotherdoc.pdf

The next page shows the single block contained in the file `anotherdoc.pdf`.

This is a paragraph taken from another EPDF file. I've set it in bold type just to make it easier to find, but it is of type /para **just like all the other paragraphs in this document.**