Models and Languages for Formatted Documents

Chris Rowley
The Open University
527 Finchley Road
London NW3 7BG
United Kingdom
C.A.Rowley@open.ac.uk

Abstract

The largest change that has come to the world of document formatting since TEX's DVI language was designed is the need to support documents destined for multiple uses, e.g., for interactive reading on screen and for paper output. It is time to investigate what is needed, both now and in the immediate future, from a device-independent description language for formatted documents.

This paper does not provide a complete such investigation. Instead, it outlines what is required from a language that can describe the "device independent" properties of the "formatted form" of the currently imaginable range of documents. The important and innovative concept identified here is the complete integration of three central formatting aspects of on-screen documents: text, graphics and the mechanics of interaction.

Introduction

Motivation. There is currently a need for a standardised but flexible and comprehensive language that will enable the description of all aspects of formatted documents that are the output of high-quality document formatters such as the growing number based on TEX (and its derivatives); this would also then be available for the output of all the superior XML/XSL-based formatting software that, we are assured by those who control the world, will soon decorate our desk-tops.

Many aspects of formatted documents, such as graphics and colour, were deliberately excluded by TEX's designers but the largest change that has come to the world of document formatting *since* TEX's DVI language was designed is the need to support documents that are designed (to high standards) for multiple uses, e.g., for interactive reading on screen and for paper output. Many people now have experience of using TEX as the typesetting engine for such documents, producing as the multi-use output form a 'PDF document'. This could be either by use of pdfTEX or by producing PostScript and then converting this to PDF (the acronym PDF here refers to Adobe's Portable Document Format). The power of combining the programmability of TEX with a thorough knowledge of the capabilities of PDF viewers and Java programs have been brilliantly illustrated by Hans Hagen.

There probably also exist other necessary extensions to the currently used models for formatted documents that are not supported well by any current such language. Thus it is time to investigate what is needed, both now and in the immediate future, from a device-independent description language for formatted documents. In order to illustrate and crystallise these ideas, it is useful to consider how these needs are met by the current version of PDF or by other languages such as the Scalable Vector Graphics language. This would lead to a far longer paper detailing the achievements and failings of the current version of PDF and the relevance to this subject of the current thinking on SVG but here I have confined myself to occasional comments on pertinent aspects of these languages.

Background. About a year ago I was bold enough to state:

> By August 1999 I hope, with a bit of help from my friends, to have further analysed the models and concepts that need to be supported by a language for describing multi-use documents, and how well PDF provides such support.

Well, I got a lot of help, mostly from the PDF discussion list [8] and particularly from Sebastian Rahtz and Hans Hagen who, being privy to Adobe's future plans and hence in their thrall, could often only answer with the phrase "but I could not possibly comment", even about things that are already described in the literature — such is the way of commerce.

The result is this short paper outlining what is required from a language that can describe the "device independent" properties of the "formatted form" of the currently imaginable range of documents. The discussion here tries to be general but it is heavily influenced by the currently popular resources in this area: DVI [4], PDF [2, 3] (and hence PostScript [1]), together with some, such as the Scalable Vector Graphics (SVG) Specification [5], that are currently under development.

I am very much aware that the current version of this paper lacks a lot of explanation and examples; and that it contains little about practical ways to take these ideas forward and relate them to other activity in this area. However, it does contain at least one significant new idea: the complete integration of three central formatting aspects of on-screen documents: text, graphics and the mechanics of interaction; this will lead to a more comprehensible and systematic treatment of all aspects of multi-use formatted documents.

Preamble. I shall assume that the reader has some familiarity with the DVI language (at least the commonly used parts) and with the PDF language (v1.2 or later, but only the formatting-related parts).

Please note that there are many things that are not covered in this paper because, although very important for modern document science, they are not directly relevant to the current subject. For example, since we are considering a description language for formatted, multi-use documents, we completely ignore the current uses, aimed at expressing documents as logical tree-structures, of languages such as XML and HTML (although these are often used to provide an inspired mixture of semi-specified formatting and logical markup). It is also possible to combine such languages with a language such as PDF to describe "partially formatted" documents.

The major consequences of the chosen language for the design of the applications, e.g., TeX or its successors, that produce examples of it will be mentioned, but only in passing and hence incompletely. However, these are probably of greater practical importance than the details of the language itself.

The paper begins by setting up the context, describing briefly the relationship between DVI, PDF and the various models of document formatting into which they fit. It then describes various models that must be supported by a fully functional language for multi-use formatted documents and analyses the consequences of these for the structure of the language.

Subsequent work will consider in more detail the specification of the language together with the design and implementation of related applications.

DVI and PDF. One motivation for this paper was my being asked at the TUG'98 conference: which is better, DVI or PDF? My reaction then was: since they are so similar, neither! But then I was thinking only of the original version of PDF; now, having discovered the joys of v1.2 and more recently the 7.4MB of the latest (v1.3) manual, I publicly recant from that position.

Although PDF is technically not a "device independent" language, it contains a large core of stuff that is, at least potentially, as "device independent" as TeX's eponymous DVI language. Both must, of course, be parsed by an application that understands the language and its underlying document model, formatting model and page model; and, although they look very different at the detailed level, the page models of these two languages (and their abstract semantics) also have a lot in common. This is one reason why the part of pdfTeX [6] that handles classical TeX files is only very locally and minimally different from classic TeX. However, PDF has a somewhat richer document model and it integrates text and graphics in its formatting model. This is one reason why pdfTeX has extra primitives.

On the other hand, PDF also has a large, and growing, part that is dependent on the very specific, and limited, features of Adobe's own viewers and font technologies. As with most languages that are being actively developed whilst being widely used, PDF is now a mixture of good and bad ideas: it is still based on some simple but general models but these are not always used to provide extensions nor have they been developed to provide more inclusive but equally clean new models. Instead, it has grown a collection of ad hoc add-ons that lack simplicity and coherence. Much of the additional functionality of pdfTeX is there only to support such very particular features of PDF, as its meta-data objects and stream compression possibilities.

Although this is a lot, PDF does only what it does; it has become a more difficult language for a (human) document formatter or programmer to work with. It is therefore currently not at all clear how to make straightforward adaptations or extensions of the PDF language and we seem able to get from it only what They want us to have in Our documents. Note that some caution is needed when evaluating the PDF language itself since much of its expressive ability is obscured by the lack of functionality and bad behaviour of the applications currently available for viewing or translating it.

Given the close symbiosis between its development and that of the Adobe's Acrobat Reader application, it is very likely that, wisely used, it is a very good language for rapid and accurate screening of downloaded documents. However, this is by no means a unanimous verdict on the utility of PDF and any such advantages have clearly been at the expense of efficient and accurate production of those documents since the current version is far from providing the uniform, clean, comprehensible interfaces needed by writers of applications.

This suggests that there is a need for Yet Another Language: one with a clean and general model and a flexible, uniform syntax. If this is incompatible with fast incremental processing then a compilation process should be inserted to transform this into something at least as good as PDF. So here are some thoughts on such a language.

Background and models

Here is a description of the use of a FDL (or Fixed Formatted Document Language) and the models of document processing that it both supports and provides.

The global model. The assumed usage of this FDL derives from the following high-level model of the document formatting process in which it is used.

- A *generating application* (GA) produces a fully formatted document and outputs it in this FDL language.
- A *processing application* (PA) uses the information about a fully formatted document described in this FDL in order to do only the following (these are informal descriptions):
  - faithfully render (in accordance with the medium) parts of the visual content of the document on one or more media;
  - when appropriate, supply information about attributes of such a rendering needed to determine the interaction state of the PA;
  - pass on, but not process, information streams to other applications (in particular, non-visual material).

At its top level, a formatted document consists of a collection of objects, the most pertinent of which are *formatted objects* (FOs).

A model for the language. What information must therefore definitely be represented in an FDL description of a document? Here is an answer.

- The contents of the document that are needed for rendering the FOs in the document on any supported output medium.
- The contents that are needed to determine the interaction state on any supported interactive output medium.
- Information about structural relationships amongst the formatted objects in the document.
- Pointers to other resources required for the rendering process (e.g., rasterisation, font and colour information).

The following is information that is not essential but is useful; it is also very closely related to the formatted document. Other (non-formatted) objects contain such information.

- Information about the logical structure of the document and its relationship to the structure of the formatted document.
- Information needed for non-rendering activities for which support is needed; in general, this is too open-ended but some of these are the logical information that is needed for activities that are traditionally associated with on-line document readers, such as indexing and searching.

There are, of course, many other things that are essential to the complete description of a document and it may well be appropriate to add to the language objects to be used for their specification. The following are some examples (from many) of information that is important to the document but is not, per se, closely related to the formatted form of the document.

- Database information about the document itself rather than its contents.
- How any visual material produced by other cooperating applications (which may not themselves process the FDL material in this document) should be placed relative to the rendering of the document.

This paper will thus analyse in detail only the information in the first group (of four items). It will also discuss some ideas concerning the information in the second group but will argue that the FDL needs to be able to express only how the provision of such information relates to information in the first group, leaving the specification of most of this information to other, more suitable, languages; these languages have been, or will be, developed elsewhere and can be used in a wider context.

The FDL is not intended to provide a revisable document format. Thus it will contain no provision below the level of the FOs for the specification of user-level graphical objects so that they can be directly manipulated, as in a drawing application or a document editor. This should not rule out support for extensions that, like PDF, provide some very limited but useful form of structured revisions of FDL documents; however, this is not a primary property of the FDL.

Further restrictions. In order to appease the editor of these proceedings and to put a reasonable limit on the time I spend writing, I shall here restrict the analysis and discussion in the following ways.

- The top-level formatted object (FO) described by the FDL will be a two-dimensional, unrotatable rectangle (this is a convenient but not essential restriction).
- The graphical model will have no concept of transparency, i.e., no graphical layers: this reflects only the current limit on my resources for investigating the issues involved and should be relaxed as soon as possible. It is also one of the areas where PDF's support falls short of current requirements.
- There is no concept of time nor of an external environment beyond the idealised two-dimensional output medium; hence the FDL itself does not describe the non-typographic content of sound/video; and documents cannot be defined to look different on Wednesdays, on Macs or on Vancouver Island (although some such requirements could, of course, be implemented by the PA).
- There is no concept of service levels to be negotiated between a client, knowing its local PA resources, and a document server.

Note that the first restriction does not limit the scope for specifying what is displayed since, within these top-level FOs, complex clipping paths can be specified.

Note also that the PA can use information expressed in the FDL to do complex things such as affording different views of the document and controlling time-dependent actions to produce *son-et-lumière* shows, etc., but these do not need to be described directly within the FDL.

Moreover, the information needed to control associated multi-media actions should be encoded in languages designed explicitly for describing such objects and these languages should not be part of the FDL.

A model for the medium. The abstract model of the visual medium is therefore a rectangular subset of a mathematical Euclidean plane on which are defined attribute functions such as "colour". Thus other technical issues not dealt with here are the precision of numerical values and the closely related provision of rasterisation information. These are very important in practice but it is best to keep them clearly separate from the raster-independent, arbitrary precision part of the model. In addition (or rather subtraction) many of the complexities of colour and tone rendering are not present in the model since these are intimately connected to the rasterisation process.

Having so peremptorily dismissed rasterisation from this formal model, I must quickly explain that everything in the model is predicated on a model of device-dependent *rendering* that involves a rasterisation of this idealised plane.

Analogies. One can liken a simple implementation of this global model to a translator (the GA) and its agent (the PA), where: the translator compiles application-oriented document formats into a well-defined, machine-oriented representation of the visual form of the document; the agent processes this lower-level code. In this simpler paradigm, the "model for the language" is analogous to the operational semantics of that machine-oriented representation and the "model for the medium" would be the abstract architecture of the machine.

A heuristically better, but less precise, analogue is with database models that include pre-compiled views and data indexes.

Analysis

At the lowest level such an FDL needs to be able to express, within the above limitations, full details of the following, and nothing more:

- everything that could be visually displayed by any PA using any supported visual medium;
- everything that is needed for the detection of interaction events by any PA that supports interactivity with such a visual mediuml

This information can be usefully divided into a number of related topics but they are all, ultimately, graphical abstractions.

Graphical specifications. Here we separate the concept of text (i.e., glyphs from fonts) from other visual items; however, we do not separate the interaction-related graphical information from the visual parts.

The underlying model for all this information is the specification of regions in the visual medium (idealised as a mathematical plane). These are the only fundamental graphical objects that are used.

Although there are many other possibilities, there is no clear reason to depart from, or extend, the commonly used collection of methods for specifying regions in terms of cubic paths; see, for example, the paper version of the original PDF specification [2]. This almost universal method is also used by PostScript and SVG.

Paths. These are used solely as a way of defining regions (stroking, etc., define a narrow area along the path). They are typically piece-wise cubics, other common forms such as conics being provided by the language only as syntax for their cubic approximations.

Note that these paths are mathematical idealisations that are then used to define the somewhat more concrete regions by means of various operators such as clipping, stroking and filling. Note that the use of these words here does not imply that any painting of the defined region is yet specified.

Regions. Having thus defined a region, it can be (abstractly) painted in some way or it can be given a label for use in defining interaction events; these are not exclusive possibilities. Note that the specification of the region is identical for both visual information and for these interaction labels. This is all that is needed from the FDL in order to support all currently used types of interaction. A region can both have a label and be painted, and these two properties are completely independent. It may be sensible for all regions to be labelled objects so that all their properties, including painting related operations, would be accessed via the label.

Events. The first stage is to define events; although there is a good case for a generic language to be used here, the present level of development of the technology suggests that ad hoc languages closely linked to particular devices may be needed for some time. That used by SVG is a good example of such limited expressibility.

Thus the FDL needs *event definition objects* where the following information can be put, using a suitable language:

- definitions of the interaction events,
- the actions associated with interaction events.

An important and common case of an action is to display a particular view of the current, or some other, FDL document; the features needed to support this are described below.

Painting. Much of what comes under the detailed specification of a painting method is relevant only to the details of the rasterisation but some, such as colour information, also need methods for device-independent specification. The most general colour information is the specification of a colour gradient function, to specify how a region should be painted; this is a mapping from the abstract visual medium to a colour space. There is a need for further investigation into what types of mappings are needed here; SVG will support a small range of mappings, including linear, radial and periodic (for patterns).

This could be extended to support the far more general concept of getting such resources from an external paint server (not to be confused with the Mix-Yer-Own machine outside the local Do-It-Yourself store); this is analogous to the commonly used indirect ways of specifying glyphs and other font resources.

Text. Although glyphs are also graphical objects, the methods by which they are specified are typically so completely different that treating the two similarly becomes fatuous. In particular, the choice and positioning of glyphs typically requires external resources and, hence, other languages. In the case of PDF and PostScript, this is the only supported underlying model for text: both positioning and rendering information for typical fonts can only be specified via a fixed external font resource that must be accessed via a fixed-size encoding table. Such external font resources are used by a specialised glyph-rendering part of the PA and also, often, by the GA: it is clearly essential, but often difficult to achieve, that these two applications use identical information.

Whilst there are good reasons to support most of these existing models and formats for glyph production, the FDL must support a far wider range allowing, if feasible, for future new glyph resources and font technologies as they come into use. Thus it should support the specification of all of the following:

- font-resource independent specification of a glyph within a font;
- explicit positioning of glyphs;
- relative positioning of sequences of glyphs (using font resources to calculate exact positioning): at least for all standard typesetting modes, both horizontal and vertical, possibly also for typesetting along more general graphics paths.

Although perhaps not strictly part of the FDL itself, a clear requirement arising from these is the ability to attach arbitrary external resources to a FDL file.

Higher-level structure. The basic formatted objects (FOs) can be related in various ways, including these three of immediate importance:

- *Logical arrangements*: these can be very general relationships but include traditional page sequences; these do not prescribe anything about the formatting of the individual objects.
- *Formatted arrangements*: use and reuse of objects within others.
- *Global information* that allows viewers/printers to define different views of a document in terms of these objects: e.g., print sequences, relative positioning of windows on a screen, suppressing the rendering of the content of whole objects (another area where PDF is currently deficient).

I see no reason to put any restrictions within the language on the nature of these relationships, thus at least a *general labelled-graph* language providing arbitrary linking information is needed here.

Such relationships and their specification need further investigation and development. Specification of the formatting relationships will immediately require an extended model of the medium that supports layers and transparency.

There is currently some small-scale research activity concerned with logical information extensions to PDF, in particular the work on structured-PDF at Nottingham University. It is unclear whether Adobe have any long-term interest in moving PDF in that direction (or, indeed, whether they have any interest at all in the language itself as anything beyond a cryptic internal language for the Acrobat blackboxes).

Trade-offs. Many of the choices that need to be made in developing the detailed syntax of the language lead to decisions that, whilst not affecting the semantics or power of the language, do affect the following measures of its utility. The first two items in this list are independent of any particular document, whereas the others will vary according to the type of document and its uses:

1. the expected functionality of the GA,
2. the required functionality of the PA,
3. the relative size of the FDL file,
4. the relative speed of the generation of the FDL file,
5. the relative speed of accessing information in the FDL file,
6. the relative speed of processing information from the FDL file.

In general, decreasing 1, and hence, typically, 4, will increase 2 and, often, also 3, 5 and 6. For example, if the FDL supports a large range of higher-level graphical objects, such as transformations, arcs of conics or smooth piecewise-cubic paths, then the GA does not have to be able to turn these into basic cubic paths but the PA must be able to process them.

Of course, increasing the amount of information (e.g., font resources) that does not need to be stored in the FDL file also decreases 3, but it also requires the PA to be able to access these resources effectively.

This section does not analyse the possibilities for the use of alternative formats since these affect equally any language. Some relevant techniques are data compression, which is comprehensively supported by the PDF standard, and binary formats that can be read quickly, as typically used by DVI but not currently available in PDF.

Summary

Outline. A formatted document, as described by an FDL specification, is a collection of reusable FOs with labelled relationships. These FOs contain positioned graphical objects including, recursively, further FOs; but they have no further internal structure. No distinction is made between the graphical objects used for painting and those used to define interaction events.

Interaction events and associated actions are not described in the FDL itself but it provides objects specifically to contain these descriptions. It also provides objects for describing external resources and the possibility to attach such resources to an FDL file.

Although glyphs are graphical objects, they are most often accessed via external resources so they must be treated very differently within the FDL.

All the organisational structure of the formatted document is defined in the FDL by general named relationships between the FOs; other logical information is not described in the FDL itself.

General principles. In developing the details of such a language the following principles should be adhered to as much as possible.

- *Indirection*: always A Good Thing.
- *Modularity*: but do not try to separate too rashly things that should be intimately connected.
- *Flexibility*: do not impose unnecessary restrictions on the GAs or PAs.
- *Extensibility*: of course! But only within the limits of the above outline.
- *Clarity*: and ease-of-use as the cream on the cake!

The way forward

The next step is to refine and formalise the ideas described here and to investigate extensions of these models that will support, in particular, a powerful concept of layers in the output medium.

Some possible (and mutually supportive) ways in which the TEX community should be able to assist in this, and beyond, are as follows:

- Further extend DVI along the lines suggested by the NTG TEX Future Working Group [7]. This already supplies a syntax for those graphics objects supported by PDF; semantics satisfying the FDL model can be simply specified for these and further necessary objects.
- Work with the W3C group to influence the development of the SVG specification so that it can be used as (part of) an FDL.
- Design future typesetting systems (such as NTS) to output such a powerful, clean FDL and write drivers that use it directly or translate it to a more processor-friendly and currently supported language, such as PDF or the API (A.. P.. I..) of a printer/viewer sub-system.

Or maybe I should move on to even more radical ideas whilst PDF and SVG/XML slog it out in the market place and TEX/DVI continues to dominate the quality niche? (Note: The last word must be treated *à la française* to make the pun work.)

Postamble

Preparing the conference talk and the subsequent discussions have shown that there are other important questions not even touched on above; thus, this paper should be treated as "preliminary thoughts".

In particular, it became very clear when I was designing and producing the examples for the talk that, even by using all currently available applications (including some pre-release versions), I could not implement all of those features of a formatted document that are currently agreed to be desirable. Moreover, it has taught me that, even at the high level of my models and languages, there is a lot more to the interactions amongst graphics, text, and screen formatting than I had considered so far.

For example, what should happen when a user resizes a window that contains both graphics and text, possibly intimately connected? The possibilities for each element are as follows (at least): resize, clip, reflow.

Resizing may make sense, within reasonable limits, for some graphics but maybe not for others; it is rarely the best thing for text. Contrariwise, reflowing is not usually feasible for graphics but may be sensible for text, again within some limits.

So who decides what is allowed? The author should at least be able to define the reasonable limits but maybe the user should have some control over what he is looking at.

Thus, more work, more ideas and, sadly, more papers, are needed.

References

[1] *PostScript Language Reference*, 3rd ed., Adobe Systems, 1999.
`http://www.adobe.com/prodindex/postscript/`.

[2] *Portable Document Format Reference Manual*, v 1.2. Adobe Systems, 1996.

[3] *Portable Document Format Reference Manual*, v 1.3. Adobe Systems, 1999.
`http://www.pdfzone.com/resources/`.

[4] *The DVIType Program.* Stanford University, 1982.
`http://www.CTAN.org/tex-archive/systems/knuth/texware/`.

[5] *Scalable Vector Graphics (SVG) Specification.* W3C, 1999.
`http://www.w3.org/TR/WD-SVG/`.

[6] *The pdfTEX Manual.* 1999.
`http://www.tug.org/applications/pdftex/`.

[7] NTG TEX future working group. TEX in 2003: Part II: Proposal for `\special` standard. *TUGboat* 19(3) pages 330–337, 1998.

[8] PDF e-mail discussion list.
`http://tug.org/mail-archives/pdftex/`.

[9] Hagen, Hans. Examples of the use of pdfTEX to produce interactive documents. 1999.
`http://www.pragma-ade.nl/`.