Beyond the bounds of paper and within the bounds of screens; the perfect match of TEX and Acrobat

J. Hagen

Pragma
P.O.Box 125
8000 AC Zwolle
The Netherlands

1 Introduction

TEX, both a typographic programming language and a typesetting engine, has some powerful primitives, that enable communication between its internal processes and the outside world. Of these primitives `\special` has proved that, although more than 10 years old, TEX will certainly survive the next decades.

With `\special` we can tell other applications what special things we want them to do. It's `\special` that makes TEX one of the first systems that supports the new portable document format PDF from Adobe Systems. Adobe supports this format with the Acrobat-family of viewers and conversion programs. PDF is a portable, extensible, readable and programmable page description language. In fact, PDF is a subset of PostScript with hypertext extensions. It has the duality of TEX: it's both a programming language and a viewing engine.

At the moment PDF does not have the stability of TEX, because it's still under development. This means that future PDF-options can cause trouble with older Acrobat viewers. We think however that within a few years PDF will be stable enough to become one of the most important standards in the distribution of documents. We can still read books that are hundreds of years old. Electronic texts need a descriptive medium that is at least as stable as paper.

One of PDF's characteristics is interactivity. It supports active documents in which we can use all kind of hyperlinks. One does not have to program texts in the PDF-format directly, because it can be generated from PostScript code. The interface between the

hyperlink mechanism and PostScript is provided by the PostScript `pdfmark` operator. This operator accepts arrays of commands, that are unique to PDF.

PDF describes a page in terms of typography and not in terms of structure, like for instance HTML does. Because HTML viewers format on the fly, authors and designers have no guaranties of the typographic quality of their products. With PDF however one has complete control, but the programs that are used to produce portable documents that are highly interactive *have* to support `pdfmarks`.

When Acrobat entered the market, no programs were available that supported the generation of `pdfmarks`. One reason for this is that it's not a trivial task to include verbatim PostScript in texts. Another reason is that the necessary positioning information is not available to users. Depending on the DVI-drivers used, TEX has the means to include verbatim PostScript, i.e. `pdfmarks`. Users of TEX also have all the necessary information at hand. That's why only a few days after Acrobat arrived in fall 1993, we could produce interactive documents with TEX.

Adobe claims that with Acrobat mankind will go *beyond the bounds of paper*. When using TEX, one is accustomed to high quality portable output on paper. We think that Adobe (again) has proven that it is possible to produce high quality portable output on screens too. This is however an area where software is years ahead of hardware: most computers simply are not yet able to show this high quality, especially not portable ones.

Experimenting with TEX and Acrobat learns that new boundaries show up: those imposed by screens. Some we can use to our advantage, some we can't. Both programs enable us to find these boundaries. In this article we will show some results from our experimenting. We highlight some aspects of typesetting that are closely related to portable documents, at least we think they are. We use the terms portable document and interactive text for documents presented on computers. We talk of screen in stead of computers and displays.

Looking at the options supported by `pdfmark` we see that, in version 1.0 (1993) as well as in version 2.0 (1994), many of them are tuned to desktop publishing programs. These programs are page oriented and so are nearly all `pdfmark` options. Our experiments show however that PDF lacks some document oriented options, but we believe they will be supported in future versions.

2 Aspect ratios

One of the more prominent differences between paper and screen is the difference in aspect ratio. As a result compatible layouts are nearly impossible: paper has height and screens have width. Long lines can't be read too well, so we can only use the full width of the screen when we use big fonts. This is why in most cases the amount of text on the screen is limited.

The aspect ratio of screens influences the overall esthetics quality of the text. When we have a lot of tables, figures and/or formulas and also use white space between

paragraphs, it's a tough job to get a good layout. Because we don't have enough height (`\vsize`) available for moving things around, floats do float indeed.

So, while waiting for displays with high resolutions, we have to use small fonts to get things right. When we also pose limits on the width of the text (`\hsize`), we have a lot of marginspace available. But too much white space doesn't look to well either, especially when it's not used. In the near future some sort of standard has to be defined for the aspect ratio of portable documents. The Personal Data Assistants that are showing up definitely have a different aspect ratio than desktop and portable computers.

3 Enhanced pagebody

Because portable documents are not as tactile as books, one needs navigation tools to manoeuvre through the text. Navigation tools are active typographic elements, that are provided by the viewers or, which is preferred, by the document itself. As we will see further on, we need some space to provide these navigation tools, and fortunately the width of screens allows for this. Figure 1 shows us an example of this.

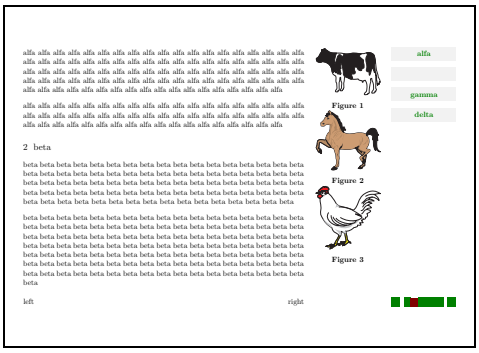

Figure 1: Screen layout

We distinguish three areas: text, margin and border. We can use the margin for marginal notes or floats and the border for navigation tools. Not shown are the left margin and left border area. Even more room is available when we extend the header and footer lines into these areas, as shown. Still more room is available above header and under footer lines, but because the height is already limited, maybe we should avoid to use them. Also not shown, and in most cases not used either, are the areas for company logos. Of course everything is implemented and available.

At the moment we are developing and adapting the relevant macros to support a dual layout, depending on a switch. In an interactive version floats are put in the margins and in a non-interactive version, they are placed as specified.

4 Parallel documents

Reading from paper still is, and perhaps will be forever, more pleasant than reading from screen. However, when we print a part of a portable document, with a layout adapted to the characteristics of screens, the results look rather silly. For instance, only half of the paper is used and minimal white space is on top and left or right.

With TEX it's not too difficult to generate different layouts from the same source. This enables us to provide documents in different versions tuned for screen or paper. We can even offer more paper variants, like single sided and double sided, color and grayscale, letter and A4.

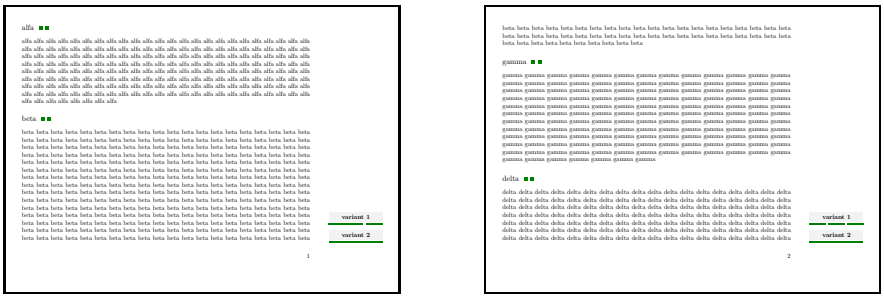

Figure 2: Document synchronization

To facilitate multiple versions we have to enhance our portable document with 'parallel document jumps'. Maybe the most simple and elegant alternative is providing small buttons after the titles of chapters and sections, one button for each parallel text. Although simple, it does not always look nice. A second solution lies in a visual analogy of TEX's `\mark`. We can offer the reader up to three possible jumps on each page: to the end of the previous page (for experts: `\topmark`), to the first on the current page (`\firstmark`) and to the last on the current page (`\bot`). The corresponding buttons can be placed anywhere on the page. In Figure 2 we see both alternatives. The titles are followed by two buttons, that let us jump to two alternative versions. The buttons in the lower right corner show the jumps per page: on page 1 there are two jumps: **alfa** (top and first) and **beta** (bot), on page 2 we have three jumps: **beta** (top), **gamma** (first) and **delta** (bot).

At the moment both mechanisms are implemented. They work well but a connection with the printing mechanism would be nice. Technically spoken, we can launch programs with PDF, e.g. a printer driver, that prints the corresponding pages.

5 Typographic interface

The user-interface of a book is contained in the book itself. The only tools we use when reading are our hands. The look and feel of a book is determined by the designer. One of the characteristics of programs and user-interfaces is that they change. The beauty of books is that they don't change. Apart from their content, they give us insight in the era in which they were written and produced. Also, in high quality books, the layout is adapted to the content and the intended use. Although a lot of the underlaying principles are fixed and based on years of typographic experience, books look different. So why should we give every portable document the same interface? And what exactly is a typographic interface?

Because the medium (with such quality) is quite new, there is no unique answer to this question yet. We can think of hot spots, menus or active words. But an interface does not have to be explicit. One may expect that clicking in a table of contents of index may result in some kind of jump. At the moment there are many non-interactive documents around. This means that we must provide readers with some cues, at least for the next few years. For the moment color suffice.

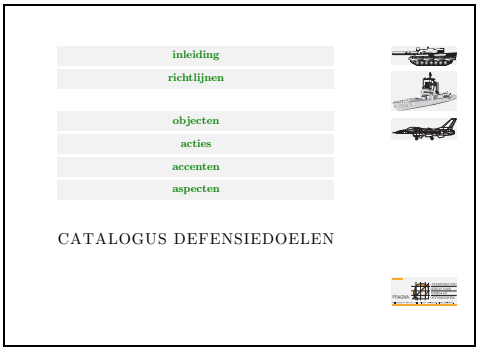

Figure 3: Typographic interface

The interface of a portable document must be determined by designers and not by programmers. The non intelligent part of the user interface, like *goto first*, *next*, *previous* or *last page*, can be programmed by means of references. Options like going to the previous jump or searching for words on the other hand, are to be provided by the program. At this moment these 'intelligent' options are not accessible as `pdfmark`.

We can think of some sort of dynamic typography, but still some constraints should be defined. One day, when computers will be real fast, TEX can be used for real-time formatting.

With Acrobat, texts can be presented full screen, i.e. without windows, standard buttons and other items. This feature enables typographers to determine the interface. Some day there will be portable computers, tuned for reading text, with simple manual

controls, a high resolution display ($\geq$ 300 dpi), no battery problems and lots of memory. Personally I hope printed matter will be around for at least my lifetime. I just don't want to think of electronic bookstores yet. All those 'metaphores' or visual look-alikes will loose their meaning when one doesn't know the originals any more.

6 Selective printing

An integrated user-interface confronts us with a problem: when we print (part of) a document, we also print the user-interface. This means that viewing programs must be able to hide marked parts of the text from printing. Acrobat 2.0 supports embedded printer-only commands, so will probably support its counterpart, embedded viewer-only commands, in the near future .

7 Layers of control

We can think of many extensions to both Acrobat and our TEX macros. One for instance is 'layers of control'. By this we mean that certain properties of the document, like typographic menus and statusbars, can be hidden or made visible by users. Typographic elements that enable us to navigate through the text, don't always add to the beauty of the text. It would be nice if users could turn them of when they are not necessary or when they irritate when reading.

8 Tables of contents

The concept of tables of contents slightly changes when developing portable documents. Acrobat provides bookmarks. These are entries to some sort of system table of contents. Because we don't have typographic control over such lists and because they claim too much space on the screen we don't use them. To be honest, we don't even need them because TEX can generate them.

Of course, clicking on an entry in a table of contents, has to result in a jump to the corresponding chapter or section. Because active words are colored (as users expect), we only make the numbers of chapters and sections active. Too much color simply doesn't look so well. We don't click on pagenumbers, because in many cases we don't show them.

Tables of contents are active by default. The necessary links are generated without any interference of the author. That's the way it should be and that's the way it's done. But to make this possible, we had to enhance the table of contents macros, that are written to an auxiliary file, with two extra arguments: an internal referencing number and, to accommodate Acrobat 1.0, an extra real pagenumber. We can't use section

Figure 4: Table of contents

numbers as reference, because they are not unique – how about five chapters 1 in five parts of a manual – and sometimes they are not even there.

We see that there can be more than one reference to a chapter or section: a hidden one, supplied by the system and used for tables of contents, and visible, user defined ones. Actually in ConTEXT we can give lists of references, just because chapters can handle more subjects and labels for references have to be meaningful: `\chapter[a,b,c]{Alphabet}`.

9 Structuring elements

Not every document has chapters and sections. Quality System manuals for instance have their own ways of structuring, often with an alternative numbering of sections. These manuals contain a lot of references, like references to forms and specific procedures.

A consistent system of numbering is a necessity for robust (automatic) referencing and local tables of contents. This means that, when producing documents with alternative numbering, we have to supply TEX with structuring commands like `\nextchapter`, which means as much as: 'here we go to another level 1 structuring element'. When using these commands, we can still produce a table of contents for this specific chapter. This problem is not unique to portable documents, but it showed up producing them.

10 Multiple indexes

Indexes, of which there may be many, have to be active too. Because pagenumbers are not unique as reference, macros had to be extended. Because an entry in an index can contain more references, resulting in more pagenumbers after a typeset entry, we make

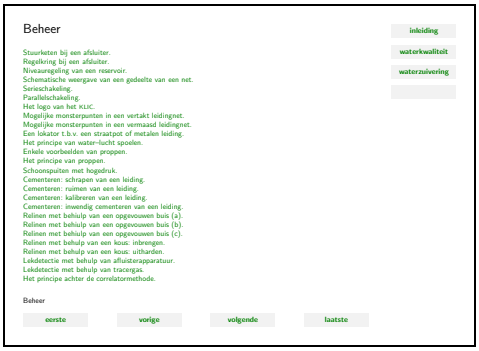

Figure 5: Structuring elements

those references active. Of course we could make a jump list out of the pagenumbers (see 'reader profiles' below), but this has no real use.

Pagenumbers in portable documents seldom have a meaning, so why should we make them active and/or even show them? But isn't it a bit overdone to keep pagenumbers in a document for the sole purpose of a register? That's why we have some alternatives. Because the number itself doesn't say much we can use substitutes like •, ■ or letters. It looks better and works fine.

11 Cross referencing

A whole book can be written on referencing and the mechanisms to support this. There are references to structuring elements, like chapters and sections, and to typographic ones, like figures and tables. Referencing can be done by page or by text: *see page 5* or *see also figure 3.1*. Portable documents offer some more, like references to locations: highlighted words in the text that one can click on. Clicking brings us to the world behind these words. These three types of references can be characterized by `\on`, `\in` and `\to`, shorthand for *on page*, *in something* and *to somewhere else*. There is another one: `\from`, that stands for *from document*.

Of course there are more types of references. In portable texts we can activate (jump to) programs. We can refer to lists of publications (`\by`), to items in lists, to margin words and to enumerations, like questions and answers (all `\on`'s and `\in`'s). Although it's best not to refer too much, electronic documents can't do without. It's just one way more to navigate them.

12 Local and global referencing

Take a table of contents or an index. In a book, we can hold our fingers at an index page and walk through the text with our other hand. In a portable document, there is no clear (tactile) concept of the beginning and the end of a document. On the other hand, we can offer as much tables of contents and indices as we want, because it doesn't spill paper. We can for instance start each chapter with a table of contents and offer the reader the possibility to jump to this table from each page in this chapter. For this purpose we can provide a button in one of the margins.

It is quite logical to label a table of contents with the reference `[contents]`. But with many tables of contents, let's say one for each chapter, we can't distinguish between them when we refer to one. This problem can be solved by introducing reference prefixes. By giving each reference an invisible prefix, like `chapter this:`, `section that:` or just a system supplied number, we have unique local tags and references. When a reference can't be solved locally, the underlying mechanism can always check if there is an not-prefixed global one. In our example this can be the table of contents of the whole document. It's even possible to walk through all prefixes until one is found, but because this is not very transparent, we have this option disabled.

Maybe the mechanism described isn't easy to understand without seeing it at work in practice. One has to take our word that from the users point of view, things are simple. The mechanism, that most of the time operates behind the screen, is also enhanced with cross-document referencing. We don't think we would have needed and developed such a complete and complex mechanism when only paper text was to be produced.

13 Multiple word references

Multiple word references normally only occur with `\to` or `\from`. On paper it's no problem if a reference crosses a line or page boundary. In an portable document we have to make each word in such a reference active. At the moment, we don't break up individual words. Because words belonging to the references are to be given explicitly and references are to be broken up in individual words, the mechanism of typesetting references is a bit more complicated than the one normally needed for paper texts.

One could state that Acrobat should handle this, but we don't agree. Acrobat has no knowledge of the meaning of text and the typographic used in it. Acrobat is perfectly able to recognize words and its search engine works fine – in version 2.0 it even accepts legislatures like ff, fl and fi! – but it would be a nearly impossible job to guess what a typographer means. Do we want to let the user click beyond the end of a line (specified with a `pdfmark`) or do we want to continue on the next line? Enhanced `pdfmarks` would be necessary to accomplish this.

14 Exact positioning

Version 1.0 of Acrobat only had a primitive referencing mechanism: one had to supply pagenumbers. Normally, when tagging a location to refer to, the pagenumber is remembered. A reference mechanism uses the page numbers as they appear in the text. When we number by chapter, pagenumbers look like *6.14* or *2–4–37*. Acrobat 1.0 expected real pagenumbers. Because of this incompatibility in pagenumbers, we had to adapt our macros. With version 2.0, referencing by label became available, so the reference mechanism could be simplified. In fact we didn't and now we have a dual mechanism, just in case we have to support more document formats.

Figure 6: Exact positioning

Before Acrobat came available, we were already experimenting with `dviwindo` from Y&Y. This viewer offers a very simple, but powerful hyperlink mechanism, that only uses labels. It offers more: jumping to some location results in an exact positioning of the cursor. When this feature comes available in Acrobat too, PDF-documents will be more powerful. Think of dictionaries in which clicking on references like *see that other word too* brings us to right page and positions the cursor at *that other word*.

Acrobat offers the option to magnify the target location of a jump. It is also possible to show only part of the page. Because we layout our portable documents for screens, we don't use this feature. We think it is better to show the reader the page as it is, but future applications can make us change our mind. For instance, I think it is possible to define cyclic references, e.g. clicking on a figure means going to the figure itself. Because it is possible to magnify the target location, we could use such a cyclic reference to magnify the figure to full-screen size (`click`: magnify, and `click again`: previous view). To accomplish this in Acrobat, one needs to specify the part of the page that is to be magnified. Because TEX has no absolute coordinates available – as a result of the dynamic character of composing pages – this feature can't be used. A solution to this would be an extension of PDF in which relative coordinates can be used.

15 Multiple documents

Referring to other documents was not hard to implement, but again, it required an adaption of our reference mechanism. We already mentioned prefixes. A reference to another document is done by an extra prefix. In `[texbook::somewhere]` we have the document prefix `texbook::`. When using multiple documents it is not necessary to have the other documents references available when we are only using `\to`. When we want to refer to text (`\in`, like *figure 3.2*) or page (`\on`, like *see page 12* of *the TEXbook*), we do have to load the references of the other document. Each reference generates one macro, in which the several components of the reference are packed (pagenumber, text and real pagenumber). As long as we do have enough memory available, this is no problem. Because our macro-package ConTEXT is over three times the size LATEX and has a memory extensive user-interface, the memory left for names of macros is already low. Of course all references can be packed in one macro, but only at the cost of processing time. (Packing works fine for two-pass optimization. We use lists of references for marginword placement, float reordering, optimal list breakup, version control and more.)

16 Common data

Using more documents together means that they must be as consistent with each other as possible, specially when they change a lot, like some kind of manual. This means that information sensitive to changes has to be as abstract as possible and has to be defined only once. This is one of the powers of TEX, but at the cost of memory. As we've seen before, time will solve this.

17 Parallel constructs

One of the first things we implemented in ConTEXT was a block-move mechanism. We use this mechanism to relocate blocks of text, independent of their location in the ASCII-text. We put the answers after the questions, hide the answers and recall questions and answers in an appendix. It is also possible to label blocks and call them up by name. The mechanism, that is completely handled by TEX itself, keeps track of the original structure of the text. This is necessary because the numbers of questions and answers can be prefixed by the numbers of chapters. It's also necessary to keep track of the local character of blocks: do we call them up at the end of each chapter and/or at the end of the document. In our example questions and answers can be seen as parallel typographical constructs.

This already (in terms of macros) complicated mechanism seemed complete. But portable documents make it possible to link corresponding blocks together: click on a question and go to the answer and vice versa. Here we see a kind of reference that is unique to portable documents.

18 Alternative pagenumbers

One of the disadvantages of portable documents is its higher degree of abstractness. Numbers of pages, megabytes and scrollbars don't mean as much as a handful of paper or a bookshelf of volumes. Numbers of pages have lost their meaning, because we don't go to pages but to locations.

Figure 7: Subpagenumbering

Because we don't want our readers to get lost, we have to offer them some kind of status-information. One kind of pagenumbering that makes sense is subpagenumbering. We can for instance give the reader some indication about the length of the current section he or she is reading. We can offer small bars, one for each subpage (screen) and highlight the current one. These bars are active: click on bar two means going to subpage two. Of course we can give cues like *page* n *of* m, but in that case we miss the interactivity.

19 Status bars

We can mimic the more traditional user-interfaces. One of the advantages of this is that users know what to expect. Although a text is no tape- or videoplayer, everyone seems to know what to do when ◁ and ▷ appear. Acrobat uses ◁◁ and ▷▷ to navigate through the list of jumps. This analogies is a bit strange, because video recorders use this symbol for fast for- and backward and CD-players for going to a previous or next track. We now have at least three incompatible interfaces.

When we take volume 8 of 20 volumes of an encyclopedia from the shelf, we have some impression of what we are reading. Size and structure are obvious. Portable documents are much more abstract. One has to visualize its size and make some mental map of it. On computers, scroll bars can give some insight in the size of the information at hand. At the cost of a lot of overhead in terms of `pdfmarks`, it is possible to provide

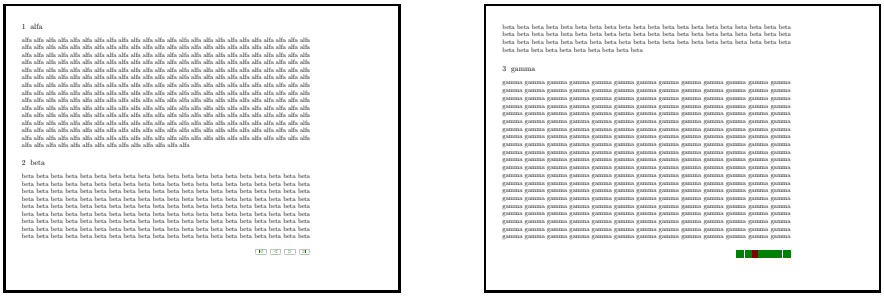

Figure 8: Statusbars

scrollbars in PDF. Because TEX is such a strong typographic language, only our fantasy is the limit.

20 Active figures

If we can make text active, why not make figures active too? In instructional texts it would be handy when pointing to some part of an object (like a drawing of a machine, a photograph of some tool or a flowchart) would bring us to an explanation of this part (and vice versa).

It is common practice to let TEX place figures. TEX handles figures as an abstraction: a figure has dimensions and a name. Normally, figures are made with specialized programs and not by TEX. Usually `\special` is used to communicate the characteristics of figures (name, scale and/or dimensions) to the DVI-processor.

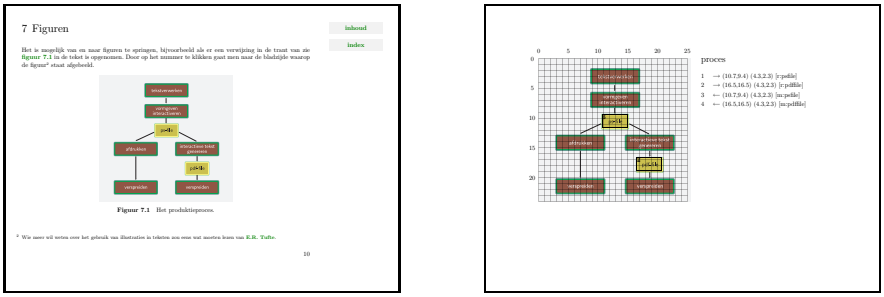

Figure 9: Active figures

It's not very handy to fall back on drawing-packages when we want to make figures interactive. First, drawing packages don't support this. Second, if they do, scaling is

not possible outside such a package unless buttons defined by `pdfmark` do scale too. Fortunately, TEX can do what we want.

One way to make figures interactive is to overlay them with an invisible grid with a fixed number of (let's say 25) ticks. This grid can be scaled along with the figure. We now can define active areas (tagged by a label) in terms of coordinates and dimensions. The same can be done with areas that are referred to. These reference areas (of type `\to`) can be used in the common way.

21 Reader profiles

Tables of contents, menus and indices are examples of structuring elements of texts. In huge, complex and continuously changing documents, there is another one: the reader profile. In terms of text, such a profile tells specific readers what parts of the document are worth reading. In portable documents, viewing programs can take us by the hand and show us what's to be read.

To me, Acrobats 'article thread' mechanism seems originally meant for connecting columns in periodicals. When we look at examples of interactive periodicals, we often see a one to one similarity of the paper and portable version. This means that articles start on one page and continue on a following one. Because of their cramped pages, the article mechanism is useful: one starts at the column, that is or can be magnified for better reading, and jumps from column to column and page to page. This is just one more example of the page-oriented character of PDF.

We can abuse this mechanism for reader profiles. When we produce the text, we can tag parts of it by logical names. A collection of such tags, connected to a starting point, forms a reader profile. Clicking on the starting point starts the article reading mechanism and the reader just has to click to follow the path. At the moment the mechanism works but misses some userfriendliness, this as a result of Acrobats pagewise behavior. We are in need for some mechanism that spans pages and provides some background color of gray when viewing the threaded text.

22 Version control

A variant of reader profiles is version control. We can mark the adapted parts of a document and offer readers the option of reading only these. This mechanism works in concert with a mechanism for selective printing of updated pages. Version control works by numbers and not by tags. This makes it possible to offer more than one reading path: version 1.2 and higher, 1.3 and higher, 2.0 and higher and so on.

A combination of version control and reader profiles are supported too. A reader can read only the updated parts of his or her profile. This is useful in for instance updated manuals.

23 Color

Color can also be used to make texts more attractive. Of course color can be used in figures. Used with care and with consistency in mind, color can enhance texts. It's common use to mark active words in a text by color. When using colored figures, we can use a colorbar to tell readers which parts of a figure are active. This means that we must use our color with care, especially in figures. Although not unique to portable documents, the color mechanism used in TEX must support consistent use of colors. One has to keep in mind that not Acrobat but TEX is responsible for the quality of typesetting. This nearly always means that the users of TEX are to blame for poor esthetics and not TEX and Acrobat.

24 Conclusion

As we mentioned before, PDF and its accompanying Acrobat programs, have to prove their stability yet. Users of TEX know that quality and stability are no guarantee for a wide acceptance. On the other hand the nearly perfect match of TEX and Acrobat will undoubtedly lead to acceptance in the TEX-community.

We already mentioned that the PDF format is still under development. Some obvious options are still missing and the present options are not yet explored to their limits. One of the mayor problems we will face in the (near) future is incompatibility. We can use new options, but can not be sure if readers have the most recent version of the viewers. (Paper documents don't have this problem, except when we change the language.) Maybe this is no real problem because the basic viewer is in the public domain and can be distributed with the documents.

A maybe even bigger problem is that we don't know what new viewing devices will come available. With tools like TEX at least we can explore the bounds of the devices already available.