The Many Faces of Acrobat & PDF

**A report of the 1 July 1999 conference of BCS–EPSG:
the Electronic Publishing Specialist Group of the British Computer Society**

About EPSG

ELECTRONIC PUBLISHING covers many fascinating subjects, in a fast-changing field. Desktop publishing, digital imaging, multimedia and now the Internet and Web challenge us to keep up to date with the rapid changes in our industry. What better way to keep up than to join a friendly group of like-minded specialists?

We know that professionals in this field need to understand a vast range of products, and that much of the real struggle is to get disparate elements to work together. Our position as a specialist group within the British Computer Society means that we are not attached to any commercial organisation, and can take a wide and independent view of events and products.

The Group generally holds about four regular one-day meetings each year, mostly in London. These are kept to a size which permits people to meet and exchange ideas in an informal and stimulating atmosphere. Each such meeting covers a single subject, which allows us to focus on areas of particular interest. A typical meeting includes an overview of the topic, talks by experts at the forefront of the field, and examples of real-life applications.

Membership of EPSG is open to all. You do not have to be a member of BCS to join (though it does cost less if you are). More information about the Electronic Publishing Specialist Group is available from the Group's office, or the Home page.

EPSG Office:
c/o Edgerton Publishing Services,
Pett Road,
Pett,
Hastings,
East Sussex
TN35 4HA,
UK

Tel +44 1424 813003
Fax +44 1424 813301

bcsepsg@eps-edge.demon.co.uk

Web URL: http://www.kcl.ac.uk/kis/support/cc/staff/malcolm/bcs-emp.html

A timely opportunity to look closer at PDF

In this special one-day seminar, the Electronic Publishing Specialist Group took a look at Adobe's Acrobat technology and the Portable Document Format, PDF, which bridge the world of on-line media and traditional print publishing. Soon after the launch of Acrobat 4.0 and the revision of PDF to level 1.3, this seemed a particularly apposite time to assess the expanding role of PDF for publishers, designers and print professionals alike.

In 1999, Adobe began to promote PDF as a superior way to deliver print jobs to repro houses and printing plants. PDF has started to become viable as a graphic file format too, assisting digital delivery of advertising art; while the integration into Acrobat 4.0 of enhanced proof annotation tools gives it an expanded role in collaborative editing. Meanwhile, the 'interactive' side of PDF continues to develop with links to Web URLs, sound and video, scriptable electronic forms etc. Researchers are developing ways of adding metadata and structure to PDF. Finally, key elements of the graphics model which underpins PDF and PostScript alike lie behind the Scalable Vector Graphics format being developed for the Web.

Fundamentals and futures

Recognising that information on Acrobat can be hard to find, EPSG developed the programme of this event to ensure that important fundamentals were reviewed for the benefit of delegates relatively new to the use of Acrobat in publishing, and a Technical Question Time session gave us an opportunity to explore common problems. However, most of our speakers showed examples of new applications that are 'pushing the envelope' of what can be achieved with PDF and SVG.

Audio recording of the event has allowed us to report the speakers' presentations in unprecedented detail, and they are presented here in the order given. However, we must confess one casualty. Chris Lilley of the World Wide Web Consortium gave a fine presentation on Scalable Vector Graphics, which was not recorded due to a glitch. As this topic was also at a slight tangent to the main thrust of the event, we have chosen not to try here to reconstruct his presentation from memory.

By presenting the proceedings of the event in this level of detail, we hope it will continue to play a useful role for an expanding circle of electronic publishers.

I take this opportunity to record my gratitude to the speakers – Peter Hibbard and Mike Clarke, Thomas Merz, Aandi Inston, Chris Lilley and David Brailsford – for their close co-operation with me and with each other before the event to ensure a balanced coverage of our topic. They truly made a terrific team.

— *Conrad Taylor, organiser of the EPSG Acrobat/PDF event.*

Edited, illustrated, designed & produced by Conrad Taylor using Adobe software: FrameMaker+SGML, Illustrator and Photoshop for Macintosh, and the Adobe Minion and Frutiger font families. Event photography by Michael Popham. Screen shot on page 20 supplied by the University of Nottingham.

Introducing the many faces of Acrobat & PDF

Conrad Taylor

Conrad is a graphic designer, and a freelance trainer and consultant in electronic publishing and information design. A member of the EPSG Committee for several years, Conrad had organised this day's event and chaired it throughout.

In his introductory talk, Conrad recalled that in 1991, Acrobat was pre-announced under the name *Carousel* at the Seybold San Francisco publishing conference by John Warnock, the CEO of Adobe Systems. Warnock argued that this technology was necessary because despite what was then starting to be called the 'Information Superhighway', document interchange was hampered by the lack of an interchange file format that would support graphically rich content, reconcile the text-encoding schemes of different operating systems, and reduce or eliminate the need to have the same fonts installed on the sender's and the recipient's machines.

Portable Document Format (PDF) was intended to play that role, turning documents into a form of 'electronic paper'. The imaging model was based on that developed for Adobe's PostScript page description language, conferring the immediate benefit of resolution-independence. If you zoom into a PDF page, you will see greater detail; also, when printed, PDF can take advantage of the higher resolution of a laser printer – unlike fax transmissions, which are stuck at a low fixed resolution.

Conrad's Web site is one place from which this document can be downloaded: http://www.ideography.co.uk/epsg/

A fine family of Acrobats

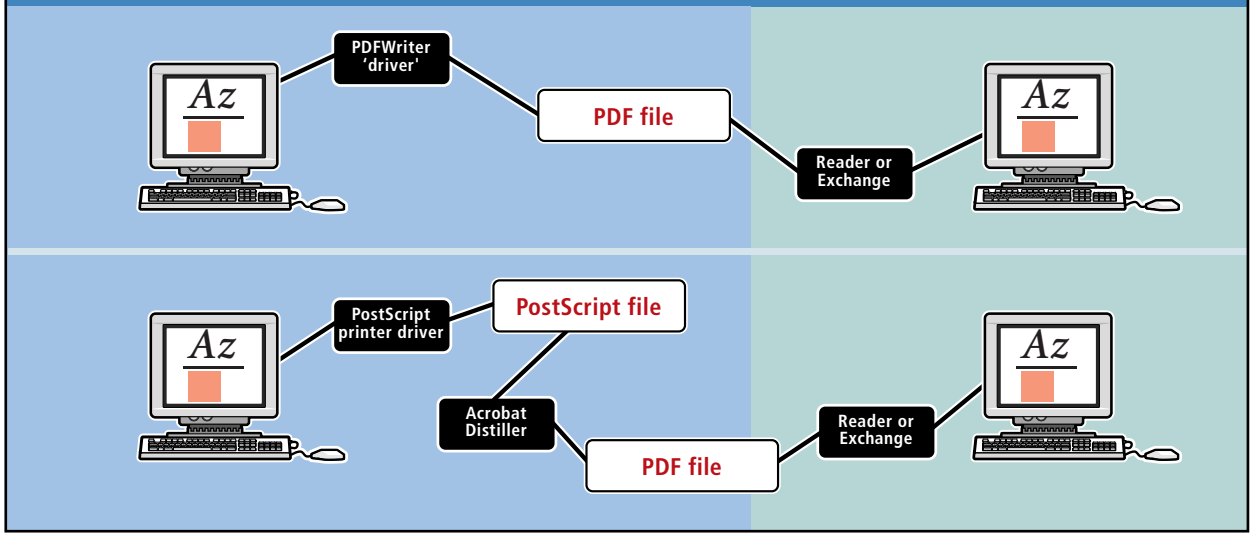
To achieve platform-independence so that PDF files could be made and viewed across several types of computer, Adobe developed and launched a suite of programs for DOS, Windows, Macintosh and Unix computers under the brand-name 'Acrobat'.

Some of these programs (PDFWriter, Distiller) were developed to help users to *create* PDF files from existing standard editing and design applications, such as Word, QuarkXPress, Illustrator, etc. Others were developed to allow users to *view* PDFs, *print* from them, and *interact* with them.

Of the two PDF browser applications developed by Adobe, Acrobat Exchange was the more powerful and expensive version with which users could also edit PDFs to some extent, adding 'Sticky Note' annotations or a list of hyperlinked 'bookmarks', cropping pages and rotating them, deleting pages or merging pages from other PDF files, building hypertext links, and performing a number of other authorial tasks. From version 4.0, this software has been known simply as 'Acrobat' – an odd decision by Adobe which has caused some confusion.

Acrobat Reader was developed as a cut-down version of the browser, initially at low cost and later turned into free shareware. Reader allows users to read and print from a PDF file, and to use any hypertext links that had been placed there; but Reader users have never been empowered to make changes to PDF files, and one cannot save back out to PDF from Reader.

Fig. 1: Two roads to PDF, via PDFWriter, or via PostScript and Adobe Distiller



Later, Acrobat Catalog was added to build searchable indexes of collections of PDF files; and Acrobat Capture was created so legacy documents available only in print form could be scanned in, recognising not only the characters in the text but also their typographic styling, and converted to PDF form.

To provide for later addition of more capabilities, Acrobat browsers were constructed so developers could add features by writing 'plug-ins' for them.

How PDFs are made

As shown in Fig. 1 on the preceding page, there are two paths in everyday use for creating PDF from existing authoring applications (though in future, more applications may be able to write PDF directly).

Windows and Macintosh computers have system-wide representations of internal graphic space (GDI for Windows, QuickDraw for Macs), and system-wide printer drivers. Acrobat PDFWriter exploits this, appearing to users as a virtual printer. If you 'print' to PDFWriter, the document's page geometry is converted to an equivalent page description inside the PDF file that is saved to disk.

As an alternative, users with PostScript printer drivers can 'print' a page description to a PostScript file, which is converted to PDF with Acrobat Distiller. This route is preferred by graphic arts practitioners, many of whom use illustrations and photographs in the Encapsulated PostScript format. The high-quality print-oriented image data in an EPS file is locked away in PostScript format, so PDFWriter is unable to render it correctly. Distiller has always been the only option available to Unix users, who do not have system-wide printer drivers. (However, from version 4.0, Adobe are not providing Distiller for Unix, only the Reader; Unix users will henceforth have to move PostScript files to a Mac or Windows machine for distillation.)

Fig 2: Minion MM is a Multiple Master font. Its set-width and weight can be varied on a sliding scale along each axis. Adobe Acrobat uses two special Multiple Master font families as a base from which substitution fonts are synthesised to match the font metrics data recorded when the PDF was made.

Whether you convert to PDF using PDFWriter or Distiller, there are dialogue boxes by which you request 'Job Options': e.g. you can define whether pixel-based images may be subsampled to a coarser raster, and what kind of compression techniques may be used on them. You may also choose whether font data is embedded into the file.

What happens to fonts in PDF?

Conrad showed slides with sections of PDF code to explain three ways in which fonts can be represented inside a PDF file.

- At a minimum, the font's name is unambiguously recorded, so that if the user reading the file has that font installed on his or her machine, it can be used to render and display the pages.
- As the PDF file is being made, the metrics of each font used are recorded: e.g. the average width of stems, the height of capitals and lower case letters, the extent of ascenders and descenders, the degree of slope of italics, the set-width of every character in the font, and the character encoding used.
- Finally, there is the option to save font data inside the PDF. Either full data for the font is written, or a subset consisting only of the characters actually used in the document.

In the first and third cases, it is clear that the font data is present and can be used to display the document correctly. But what if the font is not present on the user's machine, and the author of the PDF file has not embedded the fonts?

When Acrobat browser software is installed, two special 'Multiple Master' font families are added to the system: one a fairly generic serif design, and the other a generic sans-serif. If a PDF file requires fonts that are neither on the user's system nor embedded in the document, Acrobat generates a reasonable simulation of the missing fonts using outline data from the Multiple Masters, adjusted with reference to the font metrics data recorded in the file, so that at least the text will display with appropriate weights, widths and linebreaks, if not with *total* fidelity to the original design. (See Fig 2, left, for a further explanation of Multiple Masters.)

Of course, this substitutional approach does not work with fancy fonts which generic masters cannot simulate, and certainly would not work with non-Latin fonts. In such cases, one must embed font data.

A further issue is that while it may be technically possible to embed fonts, the font license may forbid it. The Enschedé Font Foundry in The Netherlands, for instance, forbids any embedding of its font data in either PostScript or PDF files.

But what is Acrobat for?

Conrad took the view that for much of its existence, Acrobat has been 'a solution in search of a problem'. Several times, Adobe has seemed to shift its definition of what Acrobat could be used for.

Beyond paper: At first, Adobe positioned PDF as a replacement for the paper document, a step towards the paperless office; a vision clearly expressed in a 1993 Adobe Press book, *Beyond Paper*. Corporate documents should no longer be faxed, photocopied or FedExed; they should be forwarded to colleagues' screens as PDF. A document should be able to live its whole life on screen, saving forests'-worth of paper.

It is therefore ironic that PDF has become such a common file format on Web sites, but almost always to provide a document that people can print!

Proofing and collaboration: After about a year, Adobe started pushing Acrobat as an excellent tool for designers to show page-proofs to their clients. Compared to faxed page proofs, the PDFs would be in colour, and the clients could use the Sticky Notes feature to annotate where changes should be made. However, this good idea was undermined by the inability to make detailed annotations integrated to the level that a proofreader would expect. A plug-in for Acrobat Exchange called *Re:Mark* did allow direct drawing and highlighting onto the PDF file, but the developer paid inadequate attention to the Mac version, failing to bridge one of the most common cross-platform communication scenarios, between a PC-based corporate and its Mac-based designers.

Adobe have now purchased the Re:Mark technology and are integrating it into version 4.0 of Acrobat, so this use of Acrobat as a page-proofing and collaboration technology may now take off at last.

Pages for the Web: When the PDF standard was enhanced to level 1.2 and Acrobat 3.0 software came out, Adobe's new tack was that one could use PDFs as a replacement for Web pages. Thanks to the Acrobat Reader plug-in for Netscape and Internet Explorer, PDF pages could appear in the browser window and links with HTML-based pages could be bidirectional. (This aspect of PDF use was discussed in greater detail by Thomas Merz, later in the day.)

Hypertext: In its first implementation, Acrobat allowed a user of Exchange to set up hypertext links from page to page within a single document, and to create a list of 'bookmarks' similarly endowed with hypertext linking powers. However, creating these in Acrobat Exchange was tedious, and if changes to the document content were required, one was forced to regenerate the PDF and re-create all the bookmarks and links manually.

Distiller was later enhanced to understand a special kind of PostScript comment called a *pdfmark*, and some document composition programs such as FrameMaker and Ventura Publisher were enhanced to write pdfmarks into the PostScript they created. This made it possible to generate bookmarks automatically from selected levels of heading, and convert document cross-references to PDF hypertext links.

Presentations and multimedia: Another job to which Adobe harnessed Acrobat was as a player for presentations, somewhat like Microsoft PowerPoint. (Most speakers at this conference used Acrobat for this purpose.)

To improve Acrobat's suitability for this role, Adobe included an option for off-screen rendering of the next page so that it would be instantly ready for transferring to the screen, and they also introduced some inter-screen transition effects. They made it possible for the edges of type to display with an anti-aliased effect, and in fact set this as the default display mode (many users complained about 'fuzzy type' and had to be shown how to switch anti-aliasing off). Additionally, Adobe enabled the embedding of a QuickTime movie in an Acrobat page.

It's unlikely that Acrobat can usurp the position enjoyed by PowerPoint in the corporate world, as it does not offer the template-based authoring system and clip art of the Microsoft product. However, its image compression and font independence make Acrobat a good choice for presentations that are later to be posted to a Web site, as were some presentations at this event.

Delivery of documents for printing: For some years, people have been delivering simple monochrome documents to their print service suppliers as PDF files. This has been particularly true in technical publishing: for a documentation producer using

FrameMaker on a Unix system, PDF is a great way to hand off a document to a printing business equipped exclusively with Macs and QuarkXPress. However, major barriers stood in the way of PDF serving a similar role for publishers of full-colour magazines or high-quality marketing materials.

At the start of 1999, the PDF standard advanced to version 1.3, and the Acrobat software suite evolved to version 4.0 to exploit these possibilities. Amongst other things, PDF now includes the entire imaging model of PostScript Level 3, including sophisticated colour separation and colour management. Adobe, joined by printing-oriented companies as Heidelberg and Agfa, now promote PDF as the ideal way to send a print job to a print service supplier.

Has this strengthened role in the printing industry given Acrobat greater relevance today? There are strong signs that this is so, as Mike Clarke's talk was later to demonstrate; however, not all of the elements are yet in place, and Aandi Inston's talk was to show how the basic Acrobat software suite needed to be extended with plug-ins to do all of the jobs that are needed in a print workflow, such as managing colour separation, and organising pages into the imposition groups required for imaging onto printing plates.

Cautionary tales

As an information designer and a keen student of usability, Conrad offered two cautionary tales to temper enthusiasms for Acrobat.

The first case concerned research he undertook for the Child Poverty Action Group, to see whether their *National Welfare Benefits Handbook* – an annually-updated guide used by welfare benefits advisers – could be converted into a CD-ROM based electronic book. Both HTML and PDF were considered as methods of delivery, but the HTML option was set aside, perhaps prematurely, after early testing with a reference group. (This group had a strong resistance to scrolling-page displays; PDF was also seen as much easier to create than HTML.)

At first, the experiment seemed promising, especially since the Handbook is laced with cross-references. This initially suggested that electronic hypertext would produce an enhanced, easier to use product; but the lack of physical clues to location (which the mere thickness of a paper book affords) meant that users got 'lost in hyperspace', to correct which a lot of the screen had to be given over to orientation and navigation clues and devices.

Another problem was that the coarse resolution of computer screens made small type difficult to read in Acrobat: either one puts up with the jaggies or allows Acrobat to render anti-aliased type. But Acrobat's anti-aliasing, which does a good job with the large type used in presentations, is too fuzzy for small type.

To achieve some degree of reading comfort, the Acrobat prototype of the Handbook used 12pt type, so each screenful contained less than 40% of the page content of the A5 paper version. Considering the technical nature of the text, this was a problem: the atomisation of content into small pages made it more difficult to follow complex descriptions and recommendations. The project was abandoned.

Documentation horror story

Conrad noted that it is increasingly common for computer documentation to be delivered as PDF files on the installation CD-ROM. This saves costs for the manufacturer, but can be a nightmare for the user. As an example, he cited his personal experience with the SuperCard hypermedia authoring system.

The SuperCard documentation was authored in FrameMaker in a standard book format, 20cm tall by 19cm wide, double-sided, comprising 395 pages with a 90-page Addendum. To avoid the expense of printing and binding, it had been converted to PDF and dumped on the CD. Conrad wryly noted that to enjoy this economy in a usable format, he had to employ £8,500 worth of equipment and 10 hours' labour to print, guillotine and Wire-O bind it!

To add insult to injury, the SuperCard author had printed all chapters to uncropped US Letter pages – some chapters landscape, and some portrait. Every now and then as he went through the manual, he encountered chapters lying on their side! Had he not had Exchange, rather than just the Reader, Conrad would have been unable to rotate some of the pages and crop all of them to print to A4 paper.

As a final blow, Distiller had been used with inappropriate Job Options settings: all screen shots had been subsampled and JPEG-compressed into complete illegibility. (Even Adobe have been caught out this way – in making the PDF version of their PageMill 3.0 manual.)

In summary, Conrad suggested that Acrobat is a wonderful technology, but one should be sceptical of claims about its 'simplicity' or 'ease of use'. The difficulties one can encounter in using Acrobat are rendered all the more frustrating by the paucity of user documentation. He hoped that this conference would be a great opportunity to understand what one can do with Acrobat today, and in what directions it is evolving.

Data structures in PDF

Peter Hibbard, Adobe Systems

Peter Hibbard is a computer scientist who took part in the early development of Acrobat, and who set out to explain to us some of the early design principles which Adobe had incorporated into this technology. He explained that it had been Adobe's intention from the start that Acrobat should provide a means to display 'final-form' documents with a very high degree of fidelity to the intentions of the designer, in a format which could be readily transmitted and archived electronically. Among other things, this dictated that an Acrobat document should be a single file, not a collection of files like a Web site.

A step forward from PostScript

PostScript was judged to be inappropriate as a file format for this application: the files can be large and slow to interpret, and to support large documents with many hundreds of pages, an interchange format would require internal indexes, accelerators and a well defined tree-like structure of internal objects, all of which PostScript lacks.

However, it made sense to build upon the success of the already established Adobe imaging model. This was also to ensure an economical interchange of data with PostScript so that no graphical attributes of the document would be lost.

Technical underpinnings

The team responsible for Acrobat started by considering how the internal representation of a document should be defined. It was decided that at the lowest level, the system should represent document data as objects of various types: booleans, numbers, strings of characters, names, key-value pair dictionaries and data streams, and also arrays built up of these objects, with ways of aggregating these objects into a hierarchical tree structure. (The most obvious example is that a page is an object with child-objects, including its contents, and annotations added to the page.)

It was also a design goal that the document structure which they were creating should be extensible by the addition of new key-value pairs to a dictionary, and by integrating new data types.

As for the role of Acrobat software in relation to this file structure, it was agreed that it should include a reading method for interpreting the PDF data into an internal detailed representation of the document

Adobe speakers Peter Hibbard (l.) and Mike Clarke.

structure, and a writing method for writing changed structure back out to PDF. The reading method would have to support incremental reading via look-up of the index of a document's object resources, so that individual pages could be loaded quickly and economically.

In outlining this separation between a completely defining file format (PDF) and the software required to read and write it (the Acrobat family), Peter added that the original intention had been that many more applications would have the ability to read and write PDF directly, but that this has been slow to develop.

Future pressures on Acrobat

Peter made the general comment that Adobe would have to consider external forces when figuring out where to take Acrobat, and that one of the biggest challenges has come from the Web.

The World Wide Web Consortium's proposal for the Scalable Vector Graphics file format (SVG) has set out to achieve an imaging model equal to or better than the Adobe imaging model. At a simple level, Peter noted, it is trivial to map SVG imaging structures to Adobe ones. However, it is also intended to integrate additional features such as transparency, filters, animation and interaction;. This will challenge Adobe's engineering teams to figure out how their applications can supply this functionality to publishers using SVG.

Before handing off to his Adobe colleague Mike Clarke, Peter mentioned that the beta-test version of a Java version of Acrobat Reader was now in circulation. A Java Reader could be licenced by other manufacturers for inclusion in hand-held devices such as eBooks, he suggested.

Is PDF ready for use in print publishing?

Mike Clarke – Adobe Systems

Mike Clarke works for Adobe in their Amsterdam office; his job is to 'evangelise' the use of PDF by the printing industry and its customers.

In posing his talk's title as a question, Mike said, he was reminded of a EPSG meeting of perhaps ten years ago with a title something like 'Is PostScript inevitable?' Speakers at the time had concluded that PostScript had deficiencies but seemed the best thing on offer and so probably *was* inevitable – and the rest is history. Mike expressed the hope that history was about to repeat itself, and that PDF is beginning to be accepted a serious format for print publishing.

Mike structured his talk around the requirements of three areas of activity within publishing where PDF is beginning to make an impact: the digital delivery of page components, especially advertising; delivery of jobs to commercial printers; and internal use of PDF within the workflows of periodical publishers (magazines and newspapers).

Page component delivery

When periodicals are trying to move towards a fully digital workflow, it is extremely frustrating if advertisers and their agencies still insist on delivering ads as sheets of separated film. Copydot scanning of these films to convert them back to digital data is a slow process, using expensive high-resolution scanners.

The better solution is to have the adverts delivered as digital files which can be placed within page make-up software like any other art; and the ideal file format would be one that gives reliable fidelity to the advertiser's intention, fits well in existing workflows, and ideally allows last-minute corrections.

At present, most digital delivery of adverts uses the Encapsulated PostScript file format (EPS). Mike described EPS as 'a bit of a hack': a two-part file consisting of PostScript data (which is never seen until it is output to a PostScript device) plus a low-resolution raster preview image (which gives no clues about the quality of the associated PostScript data). He identified three main deficiencies of EPS:

- The two halves of an EPS file can get detached; for instance, the PostScript data may be lost because a file 'imported by reference' gets deleted, moved or re-named, so only the preview image which has been imported into the page make-up software gets printed.
- If specific fonts are required to ensure that the EPS-format ad prints correctly, the raster preview gives no warning of this, which may result in completely inappropriate fonts being substituted during the process of outputting to film or plate (e.g. `Courier`).
- A EPS file may potentially suffer from all the typical problems which arise when PostScript has been written for a specific target printer, such as incorrect halftone screen rulings, inappropriate tonal transfer curves, etc.

Switching to PDF for digital delivery of artwork will bring benefits long before the time comes to transmit the advertisement to the periodical publisher. The file is smaller, and fonts can be securely embedded. The PDF can be sent for approval, and revisions can be requested by adding annotations to the file. (This facility has become much more versatile in Acrobat 4.0, where one can draw directly over the page image and add 'sticky notes' to those marks, and also where a complete list of annotations can be reviewed in the viewing pane to the left.)

Small last-minute changes can be made without going back to the originating application; or, if it is desired to prevent such tampering, the file can be locked. Also Mike stressed the ease of moving the file cross-platform, and the ease of running 'preflight-check' software against the PDF file structure.

Other advantages over EPS have come as a result of the upgrading of PDF to version 1.3, though Mike noted a current shortage of tools to exploit these features. For instance, job ticketing information may be added to a PDF file via PJTF, the 'Portable Job Ticket Format'. Also, because PDF 1.3 encompasses all of the imaging facilities of PostScript Level 3, it is now possible to implement colour management by embedding an ICC profile. (An ICC profile is information, written in a format defined by the InterColor Consortium, which describes the colour gamut and tonal linearisation characteristics of a system – such as a colour monitor – within which a colour image was captured, created or approved).

At present, Mike confessed, an advert digitally delivered as PDF will most likely be converted to EPS at the last minute so that it can be placed in the publication, e.g. in Quark XPress.

However, he demonstrated that Adobe's new page make-up program, InDesign, can import PDF directly. If the selected PDF has multiple pages, you can choose (with a preview) the page to be placed; if multiple cropping rectangles have been defined for the page, you can choose between them; if fonts are missing, you will be advised. Once the PDF image is inside InDesign, it can be cropped, scaled, rotated or masked in the normal way; if small changes are required, you can launch Acrobat from within InDesign to make those changes. Finally, InDesign writes a final PDF of the whole publication directly – no need for Distiller – folding all the data from the imported PDF ad into the total PDF structure.

Is it happening?

There appear to be no technical obstacles to the digital delivery of artwork as PDF; there is a great deal of variation in the uptake of this method between different European countries, but this difference owes more to politics and organisation.

In Norway, 80–90% of display ads are digitally delivered – and mostly as PDF. This has been made possible by the collaboration of over 100 advertising agencies and 125 newspapers through NADA, (*Norske Avisers Digitale Annonseleveringssystem*– the Norwegian Publishers' Digital Advertising Delivery System) which has laid down strict rules for creating PDFs, even giving out a PostScript Printer Description file (PPD) for use when making PostScript, and a Distiller startup file to harmonise the application of Job Options. At present, the NADA system is based on the transmission of CMYK colour information, but other options are being considered.

Delivering jobs to commercial printers

In recent years, relationships between printers and their customers have become more 'casual', and a printer must be able to accept a wide range of jobs.

Typically, a printer today accepts application files from customers – usually Quark XPress files. This seems the wisest policy because it gives the printer the opportunity to make last minute corrections to the job before sending it to film or plate. The most common errors are things like inappropriate halftone screen ruling settings, wrong colour separation plate assignments and the like. If the printer is given a PostScript file, it is more difficult to check it, and impossible to change it.

Is PDF ready to step into this new role? In a sense, yes. At the Seybold electronic publishing conference in Boston in Spring 1998, the 'PDF Experts Group' published a white paper detailing the inadequacies of PDF 1.2 for commercial print; but at the Seybold event one year later, the Group's representative, Stephan Jaeggi of Switzerland, was able to report that PDF 1.3 had met their objections at a technical level. (You can see the reports of the Experts Group at `www.prepress.ch`.)

PDF now supports smoothly graduated tints; masked images; and most significantly, the 'DeviceN' colourspace model, which makes it possible to define individual colour separations. This means that it is now possible to transmit a PDF that can be split into spot colour separations with duotones (photos printed in e.g. black plus one spot colour), or to support non-CMYK process colour systems such as Hexachrome.

Remaining problems in PDF print workflow

However ready PDF itself may be, there are still some elements missing in the print workflow. For example, to separate duotone information from a PDF file may require a PostScript Level 3 RIP, and not all printers have these attached to their imagesetters.

There are also problems at the application end, especially for QuarkXPress users: the composite-colour PostScript file that XPress produces is not written in a way that makes it easy to prise the colour information apart again into separations. (As Mike put it, "Xpress doesn't expect you to do serious printing from a composite-colour file".

Problems of workflows and application support for PDF print delivery should clear up fairly quickly, but any printer who wants to accept PDF files from anywhere has to be aware of the current situation.

Commercial printers may also want to draw a line by supporting only a precisely defined subset of PDF called PDF/X which is being defined by CGATS to simplify job transfer. The initial standard, PDF/X-1, requires all printing data to be in a single file, but there is a second standard being drafted to permit some information to reside in external files.

Print-IT over the Web

One Web-based technique for sending documents to be printed is offered by Print-IT, an EC-funded research project on distributed printing in which nine companies have collaborated (Indigo, Adobe Systems, France Telecom Expertel, EDS, Il Sole 24 Ore, KPN Research, Apple Computer, Drescher and Copyrama). The results are in the public domain.

Mike demonstrated this, using an Acrobat plug-in which does a preflight check of the PDF file intended for transmission, connects to the print provider's Web server and collects a PDF form by means of which which the print requirements are requested (print run, paper stock, binding, delivery address and so on). When this data is submitted, the print provider's server will generate a quotation; if this is accepted, the print requirements are rolled up into the PDF job file in PJTF format and the PDF is uploaded to the server.

PDF in periodical workflows

Finally, Mike looked at the emerging role of PDF in the workflows of large commercial publications such as magazines and newspapers, where the scale of production often dictates long distances between the editorial offices and the print plant – or plants, if the publication is simultaneously printed in several cities. The more timely the news is, the less appropriate it is to image films at the editorial offices and physically transport them to the printing plant.

In the recent past, some publications have transferred page data as high resolution facsimile files in the TIFF-IT format, one such file for each colour separation. PDF files are smaller, because PDF is based on vectors and tonal bytemaps, encodes text as text not spots, and has various forms of compression. Still greater filesize benefits can be obtained if the workflow between publisher and printer is based on composite colour, perhaps even RGB colour, relying on the power of Level 3 RIPs at the printers to do in-RIP separation, colour management, and trapping.

As a case study, Mike mentioned *Construction News*, which has delivered all its pages to the printer as PDF for more than a year. One very large publisher which has also gone down this route wholeheartedly is Bath-based Future Publishing. Particularly for large-scale operations, Mike pointed out, the simple factor of smaller file size can produce substantial savings of investment both in filestore capacity and transmission bandwidth.

In summing up, Mike asserted that we should now look to publishing application developers to provide better integration of colour management and job metadata into the PDF file structure. This *can* be done by embedding pdfmarks into PostScript files which are later distilled, but in the long term the better solution would be for more application software to write PDF files directly.

Questions to Mike Clarke

Conrad asked Mike to comment on the difficulty of generating a PDF with different Job Option settings for different images: it may be OK to subsample and JPEG-compress a photograph, but a screen-dump should receive only lossless compression. There are ways of hacking PostScript to do this, but it would be nicer to have a way of requesting exceptional overrides to Distiller Job Option settings in the page make-up program, on an image-by-image basis.

Mike agreed that this is a shortcoming of Acrobat at present. Commenting on Conrad's proposal, he said that passing control data via PostScript with the aid of pdfmark comments is a clumsy and difficult method; it would be better if applications could write PDF directly, in which case they could handling each image as the user had requested.

Another questioner asked if image compression in PDF was now so good as to make it unnecessary to use OPI (Open Prepress Interface) systems, in which scanned image data is stored separately and merged with the publication data at the point where film or plates are made. Mike thought that increasing transmission bandwidth might make OPI unnecessary as a strategy, but that some workflows which use lots of high-resolution photography would still benefit from OPI. (PDF does support OPI workflow.)

In answer to a question about Corel's support for PDF, Mike replied that Corel support PDF in their graphics applications, in a manner which he considered the perfect solution – they had licenced the PDF-creation technology from Adobe. This should be good news for users as well, he suggested: it's best if there is only one dialect of PDF.

Finally, in answer to a question about whether PDF information could drive a multi-tray printer (such as a Docutech) to print certain pages on certain kinds of paper stock, Mike replied that this is already possible to a degree in PostScript, though usually only to vary the stock used for the first page. To provide this feature, there would have to be a control interface, which would probably have to reside in the originating application. (Later in the day, Aandi Inston was to speculate that this could be a role for an Acrobat print-oriented plug-in).

"PDF on the Web and on the fly"

Thomas Merz

Thomas introduced himself as a freelance writer, journalist and software developer based in Munich. He explained that because he aims to get computing applications to work, and seeks to explain to others how to use them effectively, he likes to study 'behind the scenes'. This curiosity provided some of the motivation for his writing software that generates PDF without using the Acrobat tools.

The title of his talk referred to the two topics he addressed: firstly, an exploration of the use of PDF as a format for pages on the World Wide Web, and secondly an explanation of his techniques for generating PDF on demand.

Integrating PDF into the Web

Many publishers already use Acrobat Distiller to convert print-oriented documents to PDF, and then make these files available on their Web sites.

This is relatively easy, so much so that when Thomas recently published his second book, *Web Publishing with Acrobat/PDF*, some people asked him what the big deal was: don't you just make the PDF and put it on the Web? Why write a book about it?

From the point of view of a Web user, at least if they use a Netscape or Microsoft browser, PDF and HTML Web content can now be much more tightly integrated together, thanks to the Acrobat Reader plug-in and the Web-oriented functionality which came with Acrobat 3. Hyperlinks can now go from PDF pages to HTML ones as well as the other way round, and PDFs can appear embedded in the Web browser window. This opens the door to more imaginative and interactive view of PDF on the Web.

The PDF format has been enhanced in other ways to aid its use on the Web. Everything one can do with an HTML form can now be done on a PDF form: it's fair to say, indeed, that the PDF form features are a superset of those in HTML. There are also multimedia features in PDF, so that a PDF document can be used to present audio and video clips.

Another development which blurs the boundary between the Web and PDF is WebCapture, which is part of the full version of Acrobat 4.0. This is an HTTP client and HTML formatter which can be used to render a Web site as a PDF document, converting all of the links, form elements and so on into their PDF equivalents – as David Brailsford was also later to explain.

Why use PDF on the Web?

The major reasons cited for using PDF on the Web is that, compared to HTML, it offers precise and flexible control over page layout; it can transfer font data with the file; it is also more appropriate for long documents and where good quality printability has to be assured. However, a number of criticisms are often advanced as to why PDF might be inappropriate for use on the Web:

- PDF files can be quite large, with correspondingly slow transfer times.
- The PDF file structure is thought inappropriate for streamed delivery to a browser, because important file directory information is at the tail-end of the file and so arrives last.
- Link management is thought difficult, because the link data structures are buried somewhere in the file structure, in a way which makes it difficult to update and repair links.
- Internal links within PDF documents sometimes fail to work when they are viewed in the browser window via the Reader plug in (this is a problem of implementation in some versions of browsers).
- Although millions and millions of copies of Acrobat Reader are now deployed, it's still the case that most users of the Web don't have the Reader – and won't be bothered to go download and install software just to see one particular site.
- In comparison to the ease of generating HTML content dynamically from a database, it is very difficult to generate PDF on demand to fulfil the same purpose of data-driven publishing.

Thomas proposed to lay some of these ghosts to rest, or at least quieten them.

Controlling PDF filesize: One major problem that publishers have is learning how to control the size of Distiller-generated PDF, through intelligent use of the Job Options. In Thomas' experience as a trainer and consultant, too many publishers simply place on the Web the same high-quality PDFs that they have generated for repro. To make smaller files, it is wise to redistill the source PostScript, choosing Distiller Job Options settings which reduce the resolution of document images through downsampling, and apply an appropriate method and degree of compression.

(Thomas compared the Job Options dialogue box to a Boeing 747 flight deck – with 36 alternatives, many casual users are daunted and confused.)

Enabling streaming: We don't want wait until a whole 300-page manual has been downloaded, just so we can read the first page. This problem has been addressed by enhancements in the Acrobat software (version 3 and above), in PDF file structure (version 1.2 and above), and in HTTP, the Hypertext Transfer Protocol, which in version 1.1 was amended to allow an HTTP client to request a byterange *within* a file, rather than the whole file. Popular Web browsers can issue byterange requests (Navigator 2.0 and above, and Microsoft Internet Explorer 3.0 and above); and several common Web servers support byterange serving (e.g. Apache 1.2 and above, Microsoft Internet Information Server 3.0 and above, and Netscape Enterprise Server 2.0 and above).

If a PDF file has been optimised for byterange serving, it is possible for a browser to send an HTTP request which results in the server sending *several chunks* of the PDF file – say, the directory data at the end of the file, plus the sections necessary to display just the first page. Indeed, it's possible to follow links within this document, each jump to a new page resulting in a similar byterange request and partial download of PDF data.

Link management: This is currently a hot topic for publishers, and Web site management software is now available which adequately supports management of link networks within and between HTML pages. Some of these products are beginning to deal with the links in PDF files on the site too, but there's still scope for much more progress to be made here.

Dynamic generation: Thomas remarked that making dynamically-generated PDF is his special interest; he was to address it in detail later in his talk.

PDF on the Web: advanced topics

Search engines: If you have a large Intranet with a substantial amount of information stored in PDF form, you would certainly want your users to be able to find relevant documents through a search engine. The good news is that many vendors of search engine technology (e.g. Fulcrum, Muscat, Verity, to name but a few) have extended support to PDF: as documents are added to the Intranet, the text in the PDFs is extracted and indexed to support full-text searching. This works well, and is a good in-house solution. However, it seems that indexing of PDF content is not supported by any of the public search engines on which millions of Web users rely every day.

Adobe has prepared technical resources which would allow a PDF page which had been retrieved via such a search to have highlighting added so that searched-for words would stand out. This works in certain software combinations, but not all, so this feature too may be one that can be implemented only in the controlled environment of an Intranet.

Forms features: Thomas considers that the forms capabilities of PDF 1.2/Acrobat 3.0 and above are underestimated, and as yet underused. They can transfer forms data using the same method as HTML forms (as specified in Internet RFC 1866), which means that implementing them can be done without changing a single CGI script on the server. As an alternative, Adobe's Forms Data Format (FDF) can be used both to send forms data to the server – and to receive additional content in response, based on entries made in form fields, which will update the layout of the form. (In his book *Web Publishing with Acrobat/PDF*, Thomas goes into greater detail about uses of FDF and also the JavaScript programmability of Acrobat forms.)

Adobe Document Server: This recent innovation is software which resides on a Web server, and which rasterises PDF data so that page content can be transferred to the browser as an image, for instance as a GIF file. This may seem strange for Adobe to do, given that Acrobat is available for most computers and now even in Java form, but there will always be some users who can't be bothered to install extra software, and some environments such as Web-TV boxes where installation of software is impossible.

Dynamic PDF: the problems

The strong trend on the Web is towards sites which generate HTML pages 'on the fly' on the basis of content stored in a database. Generating content from databases offers all sorts of possibilities, such as creating custom presentations of data for a client, and also makes it easier to manage the updating of certain kinds of content. Can this model of Web publishing be extended to PDF?

Database-driven HTML is not hard to do: the mark-up is simple, and HTML tags can be stored in the database fields, or a script may add them as the data is extracted and concatenated. In contrast, to understand PDF you should have some background in PostScript, and the PDF specification is over 500 pages long. The file structure has a big and complex

overhead, and the role played in a PDF file by the directory is also problematic (because if the offsets specified by pointers in the directory point to the wrong place in the file, the file just won't work).

Of course, one way round this is to use a database publishing system to generate a PostScript file, then feed that to Distiller to create PDF. (One example of this is the *Miramo* system from Datazone, which translates database content into Maker Interchange Format, then uses FrameMaker to format it and generate the PostScript). However, the convoluted, computationally-intensive nature of the process is too slow for real-time delivery, and would soon bring a Web server to its knees. Another barrier is that Distiller is not available for the Linux platform, which is such a popular choice for Web servers.

In any case, Thomas noted, full-blown Distiller with all its prepress-supporting capabilities is an 'overkill' solution for the kind of PDFs which one would typically require in this scenario; for the Web, one could get by with 5% of Distiller's functionality.

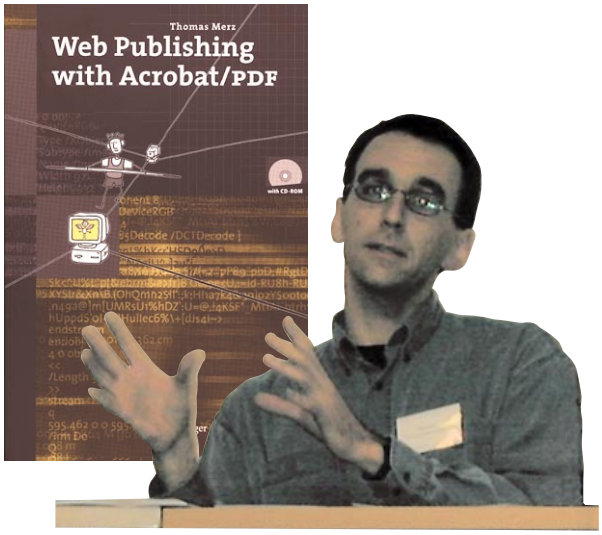
A solution for Dynamic PDF

Thomas then described his own programming work in this area. This started in 1997, as he decided to learn about PDF in a greater than usual level of detail – by creating PDFs himself.

One can write simple PDF files with a text editor, just as one can do with PostScript. However, it is hard to produce usable and significant PDFs this way. The internal directories and object flagging structures which make PDF such a useful format are a particular barrier to hand-crafting PDFs – it is so easy to make mistakes, and the mistakes are disastrous.

Initially, writing his own PDF-generating software was for Thomas a way of learning how PDF works, and it let him experiment with some features of PDF that are documented in the PDF specification, but not yet implemented by any Acrobat software. (As examples, he cited placing PDF bookmarks and annotations in Eastern European languages, encoded as Unicode; attachment of non-PDF files to PDFs; and the use of custom icons for Sticky Notes.)

Thomas' software, now at version 2.0, is called PDFlib, and is implemented as a C software library. It automates the production of much of the basic typographical and layout features of PDF, with vector graphics and images, forms and hyperlinks – the sort of thing that would be needed for generating Web-oriented PDF – while ignoring prepress issues such as colour management. The result is a fast, multi-threaded system that can generate PDFs very quickly – with a simple interface and a short manual!



Thomas Merz is the author of two authoritative and readable books about Acrobat, published by Springer in both German and English. The ISBN of *Web Publishing with Acrobat/PDF* is 3–540–63762–1.

In preparing the second release, Thomas took into consideration that many people who are uncomfortable programming in C are happy to work with simpler scripting languages such as AppleScript, Visual Basic or Perl. Therefore he generated a second 'layer' of PDFlib that acts as an interface to these scripting environments; the Perl interface, for example, would allow a Webmaster to write a 20-line Perl script which would direct PDFlib to generate a PDF file. The Visual Basic interface opens the door to the use of Microsoft's Active Server Pages, Active-X controls and other techniques.

PDFlib may be found at `www.pdflib.com`, in Open Source format, and may be freely downloaded. Thomas expects no payment for personal use of PDFlib, but charges a license fee to developers who integrate it into into commercial solutions.

Who uses PDFlib? Downloads have been in the region of 1,000 a month, so it's quite a large user base; and though the number of commercial users is small, it includes some big customers, including a number of insurance companies and banks; database software companies who have integrated PDFlib into their report generators; an e-commerce site which sells customised mailing lists; and various print-on-demand applications.

Looking ahead to the future of PDFlib, Thomas hopes to introduce more rule-based layout capabilities to the library, a kind of page formatting machine onto which could be dumped some structured data and some formatting rules. He wants to give priority to linearising and optimising PDFs for byterange serving, and looks forward to adding structure to PDF and further font-handling capabilities.

Augmenting Acrobat with print-oriented plug-ins

Aandi Inston – Quite Software

Aandi is a software developer who produces utilities for the printing industry – for instance, a PostScript editing tool called PSalter. In recent years, he has been developing plug-ins for Acrobat.

Aandi made it clear that he would not discuss plug-ins for Acrobat Reader; Adobe have set the rules for developing plug-ins so that if you want to do somehing really useful, the customer will have to have a copy of what he referred to as "the application formerly known as Acrobat Exchange".

From a user's perspective, plug-ins are things that allow Acrobat to do more more things. For the programmer, the view is usually the other way round: Acrobat is no more than the necessary framework so that plug-ins can do useful things to PDF. The plug-in architecture is useful for programmers, because it frees the programmer to concentrate on just what the plug-in is supposed to achieve.

But why write plug-ins? To express it differently, What's missing from Acrobat? Aandi explained that it isn't that Adobe created Acrobat as an incomplete or inadequate product; it's just that there are things one would want to do with (or to) PDF that aren't what Adobe have defined Acrobat as being for – though 'what Acrobat is for' is something that Adobe has changed several times over the years.

From the perspective of the printing industry, said Aandi, what's missing in Acrobat is: production tools, production tools, and production tools.

The pain of separation

Pre-press, the business of preparing electronic files and imaging them to film or plates for the printing industry, is like a horror movie, said Aandi. It's full of dead things that refuse to lie down, especially when it comes to making colour separations into process colour and spot colour.

In the late 1980s, Adobe got together with other companies and put together an interim solution for handling spot colour and process colour separations, by defining 'Color Extensions' to the PostScript language to patch what had originally been conceived of as a black-and-white printing solution. This solution requires a separate page description to be sent to the Raster Image Processor (RIP) of the imaging device, for each colour separation to be generated. As a method, it worked, and was widely adopted by companies writing DTP applications.

But Adobe went back to the drawing board, and in 1990 came up with a new colour separation solution in PostScript Level 2. In this method, a composite colour PostScript file is transferred to the RIP, and the separation is done inside the RIP. This method is better, it's more robust – and to this day it is still not much used! We are, therefore, stuck with the heritage of a lashed-together solution from the late 1980s, mixed in with some bits of the old system. Acrobat works only with the new system, but can understand parts of the old system. "There's sticking-plaster everywhere," Aandi declared.

The work-flows which have the greatest difficulty in making a transition to the Level 2 model are those that use Desktop Color Separation (DCS) files, as invented and promoted by Quark, Inc. In this kind of imaging workflow, colour separation software preseparates an image, typically a photograph, into one greyscale representation per component colour, so that each CMYK image is represented in the file-store by four files, plus a low-resolution 'preview' file with embedded PostScript comments for import into the DTP application. As XPress prints a CMYK page to an imagesetter, the PostScript comments found in the preview file prompts the system to retrieve the high-resolution greyscale data from the appropriate DCS component file and merge it into the PostScript for each separation.

This method has the virtue (from Quark's point of view) of making it easy to manage colour separations, but as an approach it is incompatible with the Level 2 composite way of doing things. However, despite drawbacks in the DCS way of handling colour, it has become so deeply entrenched in the print industry that Adobe has had to continue to support it in their applications, and indeed DCS support was even further extended in Photoshop 5.0.

Some publishers use Acrobat within this older model by printing preseparated PostScript to disk, then converting this to PDF. When viewed within Acrobat, this results in one greyscale page for each separation (e.g. cyan, magenta, yellow, black, pink, spot varnish...). Such files can be used for imaging film, but people not used to printing will have difficulty visualising what the result will be. Combining these to obtain a preview requires expensive equipment and materials (e.g. printing film positives, and using these to image a Cromalin proof).

Plug-ins for outputting separations

Lantana's plug-in *Crackerjack* is the best-known colour separation plug-in for Acrobat. Essentially, it provides Acrobat with a super print dialogue box where separations, colour management, crop mark placement, PPD selection, and so on can be requested. Crackerjack totally embraces Level 2 colour, and the PostScript that it generates is separated in the imagesetter RIP and sent to separate films or plates. This method is good, it's fast, and it's predictable; it even works with RGB image data.

However, this Level 2 approach to separation has its drawbacks:

- In-RIP separation works only on printing devices equipped with RIPs that are PostScript Level 2 compliant; and there are still older imagesetters out there which aren't.
- Furthermore, in-RIP separation is usually not a feature that has been included in laser-printer RIPs, which makes it difficult to proof your separations at low cost in the design studio.
- In the older model, it was easier for an application to specify overprinting, for instance to ensure that black elements would always overprint. In the Level 2 model, black type on a coloured background would result in negative type-shaped knockouts being produced in the colour plates, which causes an unpleasant halo effect when printing is even slightly out of register.
- Level 2 conversion of RGB to CMYK is rather crude, being based on the assumption that cyan is the inverse of red; whereas in the real world, the pigment available to represent cyan is not truly diametrically opposite to red. (In principle, the recently introduced PostScript Level 3 supports more sophisticated colour management, though there is little evidence that this is being exploited.)

An alternative colour separation plug-in is *PDF Output Pro* from Callas Software on Germany. This takes a different approach from Crackerjack, and generates Level 1 PostScript. This is ideal if your workflow is oriented towards output devices which cannot handle Level 2 PostScript, and it also has the benefit of allowing you to make colour separation proofs on a desktop laser printer.

However, this is slower: whereas Crackerjack sends data for a page once, PDF Output Pro must send data four times (or more if extra spot colours are involved). Also, it is impossible to separate RGB image data this way: images must be pre-converted to CMYK colour space.

Preflighting

In the publishing industry, the term 'preflighting' or 'preflight checking' has come to mean doing all you can to make sure that files are good before you output them to expensive film – or even more expensive plates. The aim is to eliminate human errors and system deficiencies. This is just as important for PDF files as for PostScript, because a PDF will contain what the user has put into it, mediated perhaps by Distiller settings when the PDF was being produced.

Callas' *PDF Inspector* is an example of an Acrobat plug-in for preflighting, and Aandi showed a screen shot. "At first sight this dialogue looks horrendous – can there really be so many problems with a file? But it makes sense if you think of it as an auditing tool." For example, you could set up PDF Inspector to alert you if colour or greyscale images have a resolution of less than 100 dpi (in which case they will not print well) or more than 300 dpi (in which case the resolution is excessive and the file will take too long to print, and may even exceed virtual memory limits in the RIP). You might also ask to be alerted to the presence of spot colours in a CMYK job, which would output unwanted film separations. (Callas also market *PDF Batch*, which can batch-process PDF Inspector jobs.)

To use a utility like this effectively, and to know how to set up the parameters for each kind of print job, you need to understand the printing process, and most preflighting products are for use by output bureaux and service printers receiving jobs as PDF from their customers.

Late editing

Aandi characterised late editing of PDF as 'the Devil's work', because PDF was never conceived of as being late-editable. It is almost always better to revise a job in the original authoring application, which understands such things as kerning and hyphenation and justification and text flows, and regenerate fresh PDF. Nevertheless, Aandi admitted that he himself has found it convenient to do things like move page numbers at the last minute.

Acrobat itself offers primitive editing of existing text strings, but the Enfocus plug-in, *Pitstop*, adds graphical editing. One can drag and drop elements, but Pitstop is limited in aspects such as adding new photos, and colour management.

That's where Quite Software comes in. *Quite A Box Of Tricks* contains a number of things: images can be converted from RGB to CMYK, either using fixed inks or ICC profiles. Images can be converted

to greyscale or have their resolutions changed (e.g. downsampling for proofing or for the Web), and hairlines can be thickened.

Imposition

Imposition is the process of combining images of individual pages so that large multi-page sheets of imagesetter film or plates can be printed in a single pass. As far as Aandi knows, Quite Software has the only Acrobat plug-in to do this: *Quite Imposing*.

He explained that one beauty of doing imposition in Exchange is that the end result is a PDF, and so can be previewed and checked. Also, the job can be done incrementally, instead of having a huge array of set-up dialogues and a 'Go' button.

For example, one little feature that is needed in large saddle-stitch impositions is 'creep', a procedure which shuffles the pages slightly to one side on the sheets to compensate for the displacement that occurs in binding. In Quite Imposing, this can be done as a discrete step.

In imposition, it is important to define bleeds (where images extend off the page boundaries). To handle this, Quite had to invent a little extension to PDF for version 3.0. Adobe have since established a standard in Acrobat 4.0, similar to Quite's, and so they have now adopted Adobe's standard.

There is an important point here, said Aandi. PDF can be seen as opening up the workflow, so that one can choose any tool that one prefers to get the job done. This only works so long as everyone sticks to agreed standards, rather than inventing proprietary solutions. The corollary of this is that it is important for Adobe, as the 'guardians' of the standard, to track and anticipate needs as they develop it – which can't be an easy job.

What plug-ins can and can't do

Plug-ins must do their work within the framework of Acrobat software. That means you can only do what Adobe have allowed you to do.

Some things are very easy to do: adding links, for example. Some things are harder: editing page contents is possible, but it's hard work because of PDF's data model. Other tasks might be impossible.

What Acrobat lets you do also changes over time. In Acrobat 4.0, Adobe added the ability to edit the API, and they allowed plug-ins to control things like device set-up on each page, which previously had been impossible.

Future plug-ins?

Aandi said that his list of possible tricks for plug-ins is long – and he's sure the best ones are those he hasn't thought of yet.

- It would be good to provide more output control, probably device specific plug-ins for e.g. selecting different media bins or collation/binding features on a Docutech printer.
- More late editing ability is required; it has to come. More colour control is needed too, e.g. tagging images with a colour profile.
- What about colour separations? It may be an old fashioned idea altogether, but it won't go away. How about a plug-in to make a separated PDF from a colour one? That's already on Quite Software's hot-list.
- Or, how about the other way, making composite PDFs from from separated ones? This would have some utility in making previews and proof copies. Technically, it's almost impossible to combine preseparated vector data into a composite vector image, but one could combine vector data into fixed resolution raster images so one could at least preview the results.

Are plug-ins' days numbered?

But are plug-ins days numbered? Two things say they might be. First, Adobe might incorporate the best ideas into the base Acrobat product. That may inconvenience (or even bankrupt) some developers, but the user benefits, and plug-ins continue.

Second, the whole idea of using plug-ins interactively conflicts with the grandiose workflows that Adobe, Agfa and others are promoting, such as *Extreme* and *Apogee*. These are closed, batch systems, predefined, designed to be highly reliable, using in-RIP separation and trapping, and including colour management and imposition. This is based on the premise that we will be able to predefine our workflows in advance and walk away, making the idea of using interactive tools rather quaint, like handicrafts.

However, these grand schemes are at odds with common trends in print publishing: short print runs, different requirements for each job, and less skilled preparation of the files. The best way to sort out the problems this brings printing businesses on a daily basis is upstream from the imagesetting process – which means that print-oriented Acrobat plug-ins are probably here to stay.

Find Quite Software at www.quite.co.uk

Moving Web and Print closer together: the role of SVG and structured PDF

David Brailsford – School of Computer Science and IT, University of Nottingham

Professor David Brailsford is no stranger to EPSG, being one of its founders in the mid eighties. He had chosen to speak last, after some speakers had considered primarily print publishing and others the Web, to consider how these two worlds might be brought closer through XML, SVG and 'structured PDF'. Work on these has been conducted at the University of Nottingham, in close liaison with Adobe Systems.

These two worlds cannot converge totally because they have quite different technical requirements, and come from quite different traditions. Obviously, the world of 72 dpi landscape computer displays, with transparent layers, animated graphics and interactivity, is miles away from the world of CMYK high resolution printing on paper with a Heidelberg press. However, SVG and PDF offer hope of convergence, and structured PDF offers possibilities for extracting text from a PDF in a form better suited to editing, reflowing, and repurposing to other formats.

Between two cultures

David presented his favourite presentation graphic of all time, which we've reconstructed in Fig. 3. "All human life is here, as far as I'm concerned," he said. "This encapsulates where I've been for fifteen years: not sitting on the fence, but in the middle of a battlefield!"

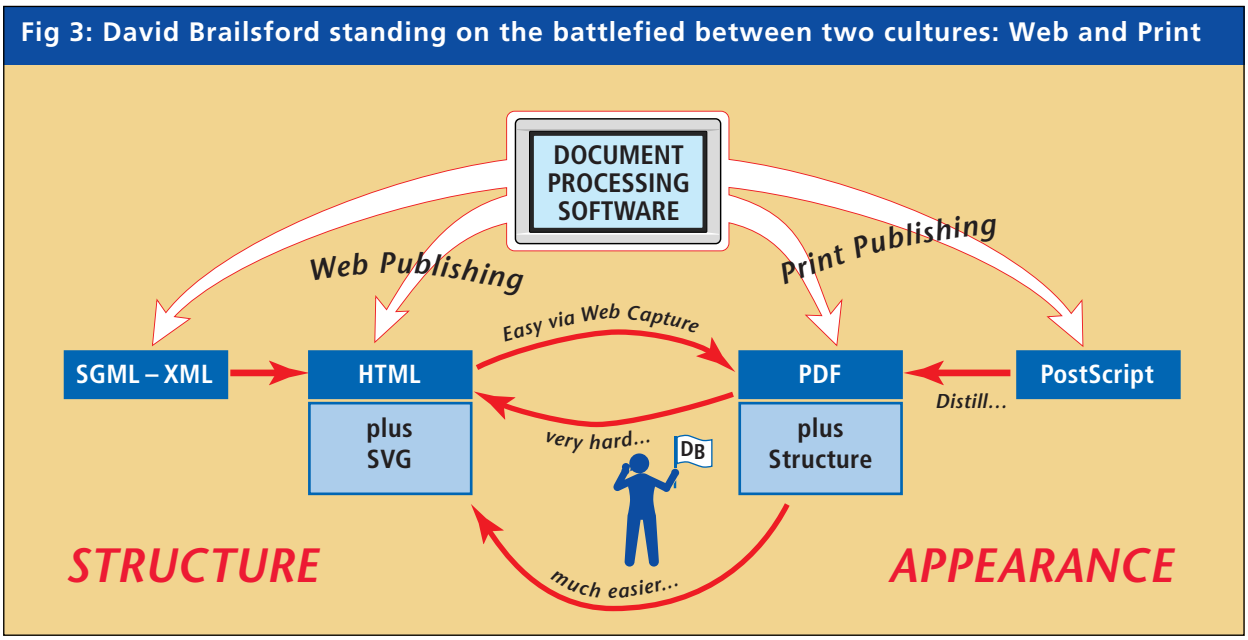
At the top of his diagram is the world of document processing software: Quark XPress, PageMaker, LaTeX, PageMill, WebObjects, you name it. From this point of origination one can either go down left field, towards encoding and transmitting the *structure* of a document, perhaps with Web publishing in mind; or down the right of the field, where structure tends to get lost, and what gets encoded and transmitted instead is the exact *appearance* of the document, generally for print publishing.

Some tools at the top are more suited to one direction than another: PageMill leans towards the Web and QuarkXPress to print. But more and more, document preparation systems are getting to the point where they could go either way.

At the bottom of the diagram, worlds are very much apart. Far to the left lie SGML and XML, fully capable of defining document structure, methodically extensible, and beloved of publishers in the scientific, technical and medical (STM) fields. HTML also lies on this side of the field, even though it has been arbitrarily extended by commercial interests, mostly in ways that show concern with controlling appearance rather than defining structure.

On the right side of the field is the well established world of print publishing, within which PostScript now plays a near universal role in controlling the

Fig 3: David Brailsford standing on the battlefied between two cultures: Web and Print



output of imagesetters, platesetters and high-end laser printers. That PostScript can be converted into PDF, which has a better structure, with an explicit tree-like index of the objects within the file. This improves access to pages, and enables other features such as annotations and hyperlinks.

Each of these worlds is inhabited by denizens steeped in its culture. On the left, we are likely to find people who express a lack of concern for document appearance; what matters to them is being able to define and extract structure (for example where headings, paragraphs and citations begin and end), so documents can be stored in a database, or edited and repurposed for different media. Much of the formative input to this culture has come from the world of computer programming languages.

On the right side, where appearance is deemed critical, formative input comes from hundreds of years of graphic design and typesetting traditions; plus, more recently, a huge input from computer graphics, a field in which John Warnock of Adobe Systems has been a very distinguished practitioner.

So far apart are these cultures at times that David has met publishers who say that though they now handle both Web and print publishing, the people who do these activities sit in separate rooms, think and work differently, and won't talk to each other or go to the same pub!

Conversion woes

On the PDF listserver (for details of joining this, visit `www.PDFzone.com`), people often ask, How can I convert my HTML into PDF? The answer is often given that this is moderately easy to do. Indeed, the Web Capture facility that is part of Acrobat 4.0, earlier mentioned by Thomas Merz, will do just this: you point it at a Web site and it will take a snapshot of that site as a PDF file. You can configure this process in various ways, for instance defining the page size to which the site will be formatted. You can also control how far along the link structures Web Capture will extend its pursuit of data, e.g. following links within the site but not those which go off-site; or visiting links off-site but only to a defined number of subsequent links. It's a very impressive utility.

However, the question which is more commonly asked is, How can I go back the other way, from a PDF file to HTML? In general this is astonishingly difficult, and often impossible, for a wide variety of reasons. For one thing, the rich graphic language of PDF is not supported in the markup-based environment of HTML; for another, with no explicit referencing of structure, structure can be inferred only from such subtle cues as the positioning of elements on a page, the size of type and so on. Moving in this direction would be much easier with Structured PDF, as David was later to describe. At least it would then be easier to convert to 'vanilla HTML', and if one could assume fully XML-enabled browsers one could convey structure in greater detail.

So, what is 'structure'?

In his next three slides, David reviewed the essential features of SGML, HTML and XML. For much of our audience this was old news, but many present were drawn from the graphics world, for whom these initals represent something novel, so we report it here. This also gave us an opportunity to review what people mean by 'structure' in the context of electronic publishing.

SGML: the *Standard Generalized Markup Language* is a metalanguage: it does not define a set of markup tags, but it sets the rules for how you can construct your own tagsets so that they (a) define the kinds of document structure and content appropriate for your application, yet (b) work in a comfortable way with SGML-aware software such as editors, databases and so on. These tagsets and their syntax must be formally defined in structured text files called DTDs (*Document Type Definitions*).

Since its inception some 15 years ago, SGML has attracted a core of dedicated followers; it is used by a number of large publishers in the STM community, and by technical documentation groups. SGML applications tend to work best when the tagsets express pure document structure and eschew any attempt to bind elements to a particular form of appearance. The sensible way control the appearance of structured documents is via external stylesheets that define how each kind of tag should be formatted and displayed for a particular kind of output.

David considers that the committee that devised SGML made one critical mistake: they permitted tag minimisation – for instance, the omission of closing tags under conditions where it might be possible to infer them. This has made it difficult to write software that handles SGML, because complex inference strategies have to be built into the parser element of these programs; and without access to the DTD, little sense can be made of document structure.

This is an example of a hungry alligator eating
a two-column article having french fries within
a narrow gutter width that produces a profound relationship with
potentially disastrous effects his girlfriend who
via text selection in PDF. becomes annoyed,
Indeed the whole mess resembles freaking out totally.

Fig 4: 'Gutter-hopping' happens when 2-column journals are imaged in baseline-sort order; this can totally confuse attempts to extract the text from the PDF in logical order.

HTML: the *Hypertext Markup Language* was based on the use of SGML notation, with tags designed for showing documents in Web browsers and establishing links between them. Development of the tagset has been at the mercy of vendors such as Netscape and Microsoft, who sought to extend it in private ways. Despite the efforts of W3C, there are still tags that work differently in different browsers, or not at all in some (e.g. Inline Frames work in MS Internet Explorer but not at all in Netscape Navigator).

Although HTML has evolved as a compromise between defining structure and controlling appearance, the graphics model is quite limited. Elements are positioned rather crudely within the browser window; images are limited to embedded GIF or JPEG files; and one has had to look beyond standards to private plug-in technologies such as Macromedia's Flash for support for vector graphics.

XML: the *Extensible Markup Language* has been recently and rather quickly adoped as a simplified version of SGML, rebuilt to serve multiformat publishing including hypermedia such as the Web. Some technical problems with SGML have been overcome: for instance, by rejecting tag minimisation, which makes it easier for software to check that an XML document is 'well formed' even in the absence of a reference DTD file.

Because XML is extensible yet rigorous, it has gained wide approval as a base syntax for expressing the features of 'next generation' browsers and new capabilities for the Web; and its notation has proved equal to the task of expressing style-sheet and link-net definitions (in XSL and XLink), metadata (in the Resource Description Framework or RDF), and even interactive object-based graphics (in SVG).

Why PDF needs more structure

What SGML, HTML and XML share in common, and what makes them 'structured', is their ability through mark-up to declare the functions of parts of a document (e.g. this is a section, this is the main heading in that section, this is a citation etc.), and to make explicit the logical hierarchy between them.

To bring the 'left field' and 'right field' of David's diagram closer together, PDF needs more of this kind of structure. (That's not to say that PDF lacks any structure; it does have an internal structure consisting of objects such as pages, image streams and text streams, bookmarks and 'article threads', all nicely indexed in a tree-like index called the pages tree. However, the structure of PDF is based around display tasks; it is quite agnostic about the editorial structure of the text.)

An HTML file is presented as a linear string, in reading order. But in PostScript, there can be an astronomical number of different ways to 'paint' a page, and PDF is at the mercy of the PostScript that gets fed to Distiller. As a humorous example, David displayed two columns of text with a narrow gap between (see Fig. 4 above). The column gap could be mistaken for uneven wordspace within a single column, and even the human reader is fooled by this for a while, until it is obvious that the passage doesn't make as much sense read 'straight across' as it does when read as two columns.

Distiller, in generating PDF, takes as its guide only the order in which text strings were imaged on the page, and some typesetting systems output multi-column text in 'baseline sort' order, not true reading order. (One such system was Miles 33, which evolved to produce multi-column journal output on third-generation CRT photosetters without page buffer memory; this forced Miles to expose the bromide in baseline-sort order. When Miles made the transition to PostScript, they didn't change their composition engine, so PostScript is generated in the same order.) It is issues like this, which are the fault of page make-up programs, that can make it difficult to extract text from PDFs in a logical and useful sequence.

A comparison of hypertext linking in HTML and PDF also shows up the lack of logical structure in the latter. In HTML, a link start-point is anchored to text

or a graphic, and may also point to a specifically-named anchor located in a text stream. In PDF, a link 'hot spot' may be positioned over some text or a graphic, but it is only a bounding box defined in the page co-ordinate space, and is not structurally linked to the text or image it 'covers'.

Likewise, the destination of a link in PDF is just bounding box, a 'view' of a particular page, in no way linked to any object on that page. PDF 'bookmarks', although they give the illusion of structure, are also simply linked to page views.

What does Structured PDF mean?

Structured PDF offers a way of recording within PDF that such-and-such an object is a table, or a figure, or a paragraph, as one can in SGML or XML. It also records the correct logical reading sequence of these objects within another tree-like index – the structure tree – irrespective of the imaging order in the pages tree. Thus an array of PDF chunks, not necessarily contiguous in the PDF file itself, might form a paragraph object in the structure tree, even if it breaks over a column or a page. This requires that structure markers be placed within text objects, pointing to a location in the structure tree; conversely, pointers in the structure tree must point back to the objects.

Even the disastrous 'baseline-sort' mess shown in Fig. 4 can be rescued by the *Marked Content* markers that were newly introduced in PDF 1.3. Each line can be bounded by a *Begin Marked Content* marker and an *End Content Marker*, given an ID, and hooked into the structure tree in correct sequence.

Structural markers would certainly help the kind of 'round tripping' between PDF and HTML which people have been asking for, especially if PDF could contain a default set of markers equivalent to HTML tags such as <P>, or <H*n*>, or list-items. Indeed, the Web Capture utility described above has an option to write structure markers into the PDF it makes while capturing Web sites, so that the logical structure of the HTML is retained.

What can we expect from Structured PDF?

The idea of structure within PDF is the subject of intense internal debate within Adobe Systems and the final outcome is not clear, but by the time Acrobat 5.0 comes out, Adobe will very likely have defined the 'default tagset' for structured PDF, roughly at the level of HTML: header 1, header 2, list-item etc. If that were the case, tools could then be developed to extract content in a logical sequence and repurpose it to HTML or SGML; or maybe even to reflow that text in a sensible way, for example for a little Palm Pilot screen, something one would not dare to attempt without an unambiguous record of the document structure.

Furthermore, it seems likely that publishers who already use SGML and a custom DTD would be able at some time in the future to 'drop down' as many of their own custom tags as they desire into the output PDF. The provision in structured PDF for 'role mapping' would allow that publisher to designate which default PDF tags are broadly equivalent to the publisher's custom tags, so that standard Acrobat tools that work with the default tagset would also work with the custom tags, albeit not to the same degree of discrimination (thus several custom tags might all map to a single PDF 'emphasis' tag).

PDF structure elements will also be like XML elements in having attributes, and in being able to share arrays of attritutes via a *class map*. Such attributes could include revision numbers, and a record of which applications were used to create or modify each element in the PDF, and at what time.

Because a structured PDF file contains two hierarchical index trees – the pages tree and the structure tree – cross-indexing will be required between them so that the abstract content elements will be mapped to objects on real pages.

The structure tree bears fruit

The research collaboration between the University of Nottingham and Adobe Systems is already bearing some fruit. Some of the work at Nottingham on how to recognise document structure from page appearance has influenced Acrobat Capture, the program which takes scanned images of pages and builds PDFs from them, using outline fonts. Capture does try its best to set out the PDF text in logical reading order – and it gets it right most of the time.

Structure will be even more important to the next release of Capture, because now Acrobat tools can used structure markup to leave little memos for themselves. Another priority will be to add intelligence to how Capture recognises the structure of data in tables, etc, and to find ways of refining the structure partially detected by Capture while scanning in further work sessions that can benefit from knowledge of the captured document as a whole.

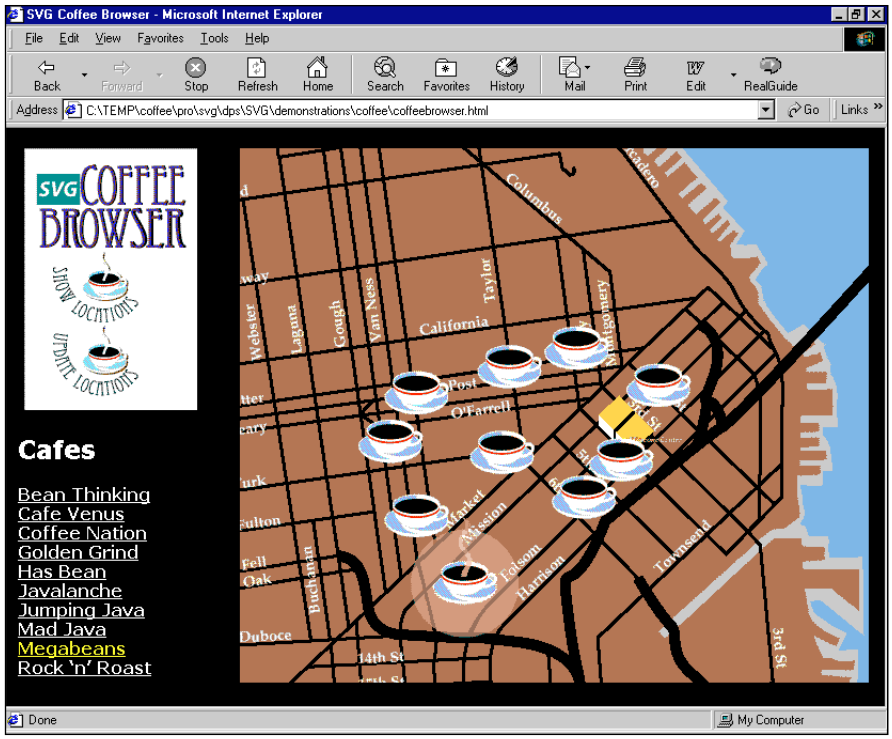


Fig. 5: the interactive SVG 'Coffee Browser'

This screen capture shows a point part way through David's demonstration. He had zoomed in to the area around the Moscone Centre (marked yellow) and clicked on 'show locations', which caused the coffee-cup icons to appear. Clicking on the text link for 'Megabeans' caused its icon to develop a transparent halo highlight, and animated steam began to waft from the cup.

Scalable Vector Graphics

The other theme in David's talk was the ability of SVG to play an important role in bridging the worlds of structure and appearance.

When the World Wide Web Consortium asked for proposals for a standard for vector graphics for the Web, Adobe contributed their April 1998 proposal for the *Precision Graphics Markup Language*, PGML, not surprisingly based on the Adobe imaging model that lies behind PostScript and PDF. David said that Adobe's involvement with this standard was always going to be important if there were to be any hope of unifying the worlds of print and the Web; had the emergent standard been PostScript-hostile, efforts to convert graphics between the two worlds might have been faced with insuperable barriers.

Also in April 1998, Macromedia published the internal format which they use in their 'Flash' vector graphics format for the Web, and in the following month Macromedia joined Microsoft and others in a proposed alternative to PGML, called the *Vector Markup Language* (VML).

In September 1998, at a session of the Seybold San Francisco electronic publishing conference, where David had been invited to speak about graphics for the Web, Chris Lilley announced that all interested parties would be getting together under the aegis of the World Wide Web Consortium to merge the various competing proposals into a single standard, to be called *Scalable Vector Graphics*. By this time, discussion had moved on to acknowledge the need for such aspects as interactivity, animation and transparency in the standard.

SVG promises to bring something extraordinary to the Web: a page-definition-like model for significant components of Web pages, offering considerable fidelity to a designer's intentions, and with more software-like interaction features than could be offered by a static graphics model, even such as PDF.

To demonstrate the power of SVG, David showed the 'coffee browser' sample that had been presented at the Seybold San Francisco meeting. The vector definitions for this interactive map of downtown San Francisco were produced by exporting them from Adobe Illustrator via a plug-in.

The demo map features hot-spots which, when moused over, causes animated icons and overlaid captions to appear to describe coffee-shops in the area, and one can zoom in to view the details: in the process, it became obvious that the type in the image was in outline formats, and would also print beautifully. Certain parts of the graphic functioned as buttons that could toggle other graphic and animation features on and off, for instance to display and hide highlighting. And lest it be thought that SVG supports only vector graphics, the 'far' view of the map includes a colour bitmap image of the Golden Gate Bridge, clipped within an elliptical mask.

The University of Nottingham continues to play a part in SVG development, working to provide public-domain tools such as input and output SVG drivers for Ghostscript, and researching ways of

converting other 'dynamic' Web formats such as Macromedia's Flash to SVG.

Some open questions

In closing, David said that his involvement in this work on SVG and PDF had raised some interesting questions. When future Web browsers understand XML and SVG, he wondered, will there be a role left for PDF plug-ins for Web browsers? Perhaps not, as any 'page-like' collection of elements on a Web page, the display of which one absolutely must control, could be provided as a SVG graphic rather than as PDF. That is not to say that PDF will wither away; but paradoxically, it may be the recently addressed high-end graphics requirements of the printing world that will be the salvation of PDF.

Discussion: no structure without effort

In starting discussion after David's talk, Conrad asked him to comment on the phenomenon of people wanting to get structure in their documents 'magically', without the extra work of creating that structure in originating applications.

In response, David commented more on the technical ability of the software used to record and convey that structure 'downstream'. It is very easy to drop structure down into PDF from LaTeX, and he wouldn't be surprised in FrameMaker could be made to do the same trick easily; but getting structure out of QuarkXPress is more difficult.

David recalled sharing a conference platform with Quark founder Tim Gill, who described SGML as "a solution in search of a non-existent problem". Yet now there are many newspapers, who have been archiving editions for years by distilling unstructured PostScript from unstructured Quark, who are now experiencing the greatest difficulty in retrieving article content from PDFs in proper reading order. Intelligent structure-diagnostic technologies such as those pioneered in Acrobat Capture may be the way to rescue PDFs in this state.

This sparked off a brief flurry of discussion which highlighted the difficulty of 'round-tripping' edited content back to the original contributors and authors at the end of a production cycle, given that many of them work in Microsoft Word. One member of the audience pointed out that Acrobat 4.0 for Windows includes a utility for making PDFs from Word documents, which can embed structure based on the Word styles used. However, in many publishing houses these source Word files then move into the de-structuring QuarkXPress environment for print publishing – at the end of which process, any structure present in the source files has been lost.

Technical Question Time

The final session of the conference was organised as a 'brains trust'. Cards had been circulated in the morning so that people present could submit written questions to a panel consisting of all our speakers.

About 50 questions were offered, which Conrad grouped into topic areas. Each question was read out, sometimes with clarification from its author, and was then directed to a member of the panel; other panel members joined in after the initial response. The session continued vigorously for almost an hour.

We present here a selection of the topics that are likely to interest a broader section of our readers.

Questions about making PDFs

Q: Please explain why PDF files are so much smaller than the original PostScript files. What has been deleted or refined?

Conrad explained that one factor is data compression, which is usually applied to a PDF's internal objects when it is being made. Text, vector graphics and simple bit-maps undergo 'substitutional' compression: regularly-occurring patterns of data are detected and placed in a 'dictionary' and each pattern is keyed to a much shorter token which is then substituted wherever that pattern occurs in the data. Because this kind of compression can be reversed symmetrically, it is called 'lossless'. (LZW and ZIP are examples of lossless compression schemes.)

Greyscale and colour raster images in PDFs may be compressed more strongly using the JPEG algorithms; however, this is a 'lossy' scheme and should be applied with care to avoid unacceptable distortion to the images. Image resolutions may also be subsampled, reducing the number of pixels in the images through a bicubic interpolation method. All these factors have an impact on both file size and final quality, and they are controlled via the Distiller Job Options.

Thomas commented that there are two additional factors in PDF which can contribute to smaller file sizes. For a start, PDF graphics operators are compact: where PostScript uses **moveto**, PDF uses **m**. Less well known is PDF's superior resource management. Thomas took as example his own presentation slides, all of which share a common background. In PostScript, this is redundantly described for every page; but when you 'optimise' a PDF, shared resources like this are stored once, and referenced as required by pages that need them.

David clarified that this optimisation only works for page elements that are identical on every page and are found at the top of the description of each page – that is, the bottom-most element on each page, as on 'Master Pages'. In Acrobat 3.0, you can optimise only during a 'Save As' operation from Exchange, but in version 4.0 one can optimise while distilling.

Mike added that PostScript is a programming language which application developers can extend by defining new operators. Therefore, when an application writes a PostScript file, it first sets up all the dictionaries and procedures that it might possibly need: a PostScript file from Illustrator that prints only 'Hello World' will nevertheless contain fifteen pages of procedure-defining PostScript, to prepare for *anything* that a user might require to image, such as graduated tints. PDF works instead with a concise set of pre-defined operators; these don't have to be defined in the PDF file, which can get straight down to the job of describing the file's contents.

In fact, PostScript doesn't have to be verbose, it's just that applications often write PostScript that way. If you take a large PostScript file and distill it to PDF, then print to PostScript again from Acrobat, you'll find that the resulting PostScript is often much smaller because it doesn't contain unnecessary code.

Q: There are many ways I can make a PDF: print via the Chooser, distill PostScript, or straight from Quark 4.0. Which is best?

Thomas said that in general it is best to produce PostScript and distill it. He pointed out that many current releases of software have a menu item 'Save as PDF', but in fact they produce PostScript in the background and call on Distiller to produce the PDF. Which is OK if that's what they do; but if they use PDFWriter instead, the results won't be so good. In future, if the PDF logic is built into a program, it can write PDF directly; this will be the case with Adobe InDesign.

David added that one reason why PDFWriter makes such bulky files is that it is a one-pass process, converting each page separately; it is unable to look at the whole document. It is to be hoped that when applications take on board the task of writing PDF directly, they don't go for this quick-and-dirty approach.

Q: Acrobat's bookmark list is OK to use but unwieldy to create. Is it possible to create such a linked contents list during the creation of the PDF [from FrameMaker]?

Conrad replied that because these publishers use FrameMaker, the answer is readily to hand. If Adobe PostScript drivers are installed, the option is given when printing PostScript to disk to define which FrameMaker paragraph styles – typically, several levels of heading – should be converted to bookmarks, and to set the hierarchy between them. As FrameMaker generates the PostScript, it embeds *pdfmark* comments which Distiller will automatically convert into the required bookmarks.

However, if what is required is a conventional-looking contents page with automatic hypertext links to chapters and section headings, the best approach in FrameMaker is either to produce the contents page as a 'generated TOC' file in a FrameMaker Book, or to create a contents page and place the required headings as cross-references, where the Xref format quotes the referenced heading's text in full. This results in a FrameMaker hypertext link which will also be converted by Distiller.

David added that Structured PDF will add optional 'structured bookmarks' which will point to a structural marker embedded in the relevant text element, not simply to a bounding box on a page as at present.

Chris revealed that W3C's discussion documents are circulated as HTML files with lots and lots of internal cross-references, but also as PDFs with the same links. To do this, they use a program called *html2ps* which works with a print-oriented Cascading Style Sheet file to format the HTML for print, and as it produces the PostScript it adds all the *pdfmark* codes necessary for preserving the links when the PostScript is distilled to PDF.

Thomas added that he has taken great care to document *pdfmark* thoroughly in his book. Additionally, the relevant chapter is freely available on his Web site, and the *pdfmark* code can be copied and pasted from there.

Questions about plug-ins and tools

Q: What tools exist for checking the integrity of a PDF file specifically for viewing in Acrobat Reader, and more specifically, something that will run in batch mode?

Aandi replied that the issue here would appear to be whether the complexity of pages recorded in a PDF was such that memory limits are exceeded when Reader tries to image the page. For example, if a photo has been 'cut out' by masking it inside a Bézier-curve-described clipping path, and that path consists of thousands of nodes because it has been done sloppily in Photoshop by selecting with the magic wand tool and then applying a 'Make Path' command, Reader could well have problems attempting to render the clipping path. But it would be difficult to predict how this would work for each user of Reader, as variations in operating system and available memory would be significant in each case. One could write a plug-in that could detect such complex pages, but Aandi didn't know of any.

Q: How can we do PDF to text conversion, for import into word processors?

Aandi said that one of the most common kinds of question is about converting PDFs to other formats such as Word, text, RTF, SGML, whatever – so that further changes can be made. The panel agreed that the best advice is always: **Keep the original files!** It is quite extraordinarily difficult to go back out from PDF, for reasons previously explained in David's talk.

However, where problems exist and people are prepared to spend money to solve them, developers will try their best. BCL (`www.bcl-computers.com`) have a plug-in called Jade which does text extraction from PDFs, and a British company called Iceni have a similar

tool. With such tools it is even possible to extract the contents of a PDF table to an Excel spreadsheet; but one must remember that such programs are playing a guessing game based on graphical clues, given that the PDF itself lacks that degree of structure.

In further discussion by the panel, **David** said that he's often asked why text processing applications couldn't use PDF as an interchange format, which seems like a very reasonable proposal until you consider the myriad kinds of structure that editing programs have to support (such as paragraph integrity, pagination behaviours, cross-references, footnotes), support for which is entirely lacking within PDF. Structure within PDF would help PDF to be an interchange format but would only be part of such a solution.

Q: We are often asked to import vector graphics e.g. from FreeHand into PowerPoint, with limited success. Could we import PDFs instead?

The panel agreed that Bill Gates is unlikely to be too enthusiastic about supporting PDF as an import file format for PowerPoint. The problem is that if the FreeHand vector graphic is saved as EPS, what you see when you import the EPS into PowerPoint is just the preview raster image at low resolution. **Chris** said that it's better to save from FreeHand in Windows Metafile format (WMF), which gives a better rendering of the vectors. **Thomas** suggested replacing PowerPoint as a presentation tool altogether, using Acrobat itself, as most of our speakers had done.

Questions about printing

Q: In Acrobat 3.0 Reader there was a problem in printing to PCL printers; it meant that the PDF files had to be created with specific Distiller settings. At first sight it looks as if the Acrobat 4.0 Reader will print more reliably both to PostScript and PCL printers – can we rely on this?

Mike explained that an Acrobat browser faced with the task of printing a page to a non-PostScript printer simply creates a vast bit-map in internal memory and dumps that to the printer. But **David** reported that from conversations with Ken Anderson of the Acrobat engineering team he has learned that a lot of effort did go into more efficient and reliable printing to non-PostScript printers for version 4.0. Despite the big mismatch between the PostScript model of page description and the Windows GDI and Hewlett-Packard PCL models, Acrobat 4.0 is converting as much as possible to PCL, and this may explain why the questioner had seen improvements.

Q: Why is there no facility for printing in black and white only to speed up printing of pages with backgrounds, like PowerPoint does?

Thomas commented: it's difficult, because PDF supports so many colour models, not just RGB. **Aandi** said that one the one hand, one could convert the PDF to greyscale, and his plug-in kit *Quite A Box Of Tricks* includes a tool for this (which turned out to be difficult to implement). However, he thought that most colour-printer print drivers would now include an option for printing in greyscale.

Q: I train academic users to use online journals, and I meet printing problems such as lack of memory. Should I suggest that they buy new printers? Print fewer pages?

Mike commented that if one is having problems in printing from journals saved as PDF, one might want to check whether the images have been saved into the PDF at unnecessarily high resolutions. It's certainly possible for a PostScript printer to run out of memory when trying to render very complex pages.

Conrad pointed out that if your page contains a very long Bézier-curve path, such as he had produced while drawing world maps in Illustrator, a PostScript printer has to render this by an iterative process that divides each Bézier curve segment into two, and so on, until a polygon is created with a very large number of points. Each iteration causes twice as many coordinates to have to be stored in virtual memory, and eventually the memory requirement can exceed the installed RAM in the printer.

Aandi added that printers are actually often able to swallow raster images of huge size and resolutions, but in his experience using many fonts on a page is more likely to cause memory problems to occur. Also, he has found that if you do not embed the fonts in your PDFs but rely on the Acrobat Multiple Master fonts to build substitute instances, these artificial fonts can actually cause more memory problems than if the fonts actually used had been embedded in the file!

Questions about fonts

Q: A major problem for us in Acrobat 3.0 is the loss of text information when using TrueType fonts. If we make PDF using company-customised TrueType fonts, the resulting PDF cannot be searched, and text blocks cannot be copied and pasted.

Thomas said he had expected "the dreaded TrueType problem" to come up. There is no easy answer. Adobe tried to tackle it by rewriting their printer driver. The problem is related to what happens as TrueType font data is translated into temporary *Type 42* PostScript font data. Re-encoding takes place as the fonts are subsetted, both in the printer driver and in Distiller, and once the fonts have been re-encoded, they may look fine on the page and they print well, but they cannot be repurposed in any way; they cannot even be searched or indexed because the numeric strings stored in the non-standard encodings do not stand for the expected characters. The only advice Thomas could give, without guarantee of success, is to use the very latest Adobe PostScript printer drivers and the latest version of Distiller.

After the conclusion of the meeting, EPSG's Annual General Meeting was held.