a page into geometric elements (e.g a group of text lines, or a photograph, that form a ‘block’ of some sort) followed by the demarcation of these blocks using paragraph breaks, inter-column gutters and so on. After a preliminary tagging of these blocks according to their apparent type (e.g. photograph, heading, paragraph etc.) it may be possible to combine this type classification with the known geometric block layout to discern which headings (for example) govern which particular text and graphic blocks. In this way one can begin document classification by inferring whether this document is a journal paper, newspaper, brochure, business letter or some other document type [4]. Previous research in this field has taken scanned bitmap images as the input to the document analysis system and classification of the document components is often guided by *a priori* knowledge of the document’s class [5–9]. It is noteworthy that there has been hardly any research in using PostScript as a starting point for document analysis. Certainly, if a PostScript file has been designed for maximum rasterising efficiency it can be a daunting task even to reconstruct the ‘reading order’ of the document. It may also be the case that previous workers have presumed that a well-structured source text will always be available to match PostScript output and, therefore, that working ‘bottom up’ from PostScript would seldom be necessary. However, we shall find that documents can be acquired in the PostScript-related PDF language in a variety of ways (including an OCR-based route directly from a bitmapped page). The extra structure in PDF, over and above that in PostScript, helps to achieve the goal of *document understanding*, which can be defined as the mapping of a logical document structure onto the geometric structure created from document analysis 4, 5]. More precisely, the goal of the present research is to work towards full document understanding and structural inference with no *a priori* knowledge of the document class. We would like to be able to recognise a document as being a journal paper, say, and then be able to analyse it thoroughly enough to confect an SGML Document Type Definition (DTD) for all of its elements. Alternatively we might wish to perform a ‘fuzzy match’ of the analysed document elements against a previously supplied DTD. We are not yet at a stage to achieve this full document understanding but the next sections describe what we have achieved in the necessary preliminary stages of document analysis and classification. We also set out those features of PDF which make analysis and classification easier if PDF is used as a starting point rather than a simple bitmap page image.

## 2 THE PORTABLE DOCUMENT FORMAT

PDF may usefully be regarded as a tree-structured version of level 2 PostScript with each node in the tree being an array of imageable objects belonging to a particular page. Associated with each page object are the ‘resources’, such as fonts, that the page uses. PDF also has facilities for hyperlinks, bookmarks, ‘yellow stickers’ and thumbnail images of pages (as an aid to navigation) [2, 10].

A PDF file can be viewed via software such as Acrobat Reader or Acrobat Exchange available from Adobe Systems Inc. The former software is available free of charge for a variety of hardware platforms but it does not allow any alterations to be made to the existing hyperstructure embedded in the PDF file. If it is desired to alter this hyperstructure, or to inspect the imageable material via the Application Programmer’s Interface (described later) then the Acrobat Exchange version will be required.

PDF may be created in three different ways:

- By the *Distiller* software from PostScript: here a special PostScript interpreter creates PDF from PostScript files.
- By *PDFWriter*: This program is a normal printer driver which can create PDF, directly, from document authoring systems.
- By *Adobe Acrobat Capture*: Capture is a document analysis package which starts with a bitmap image of a document in TIFF form and recreates this document faithfully in PDF using a customised version of PDFWriter. This processing requires the recognition of fonts, as well as words and characters. The resulting PDF is usually much more compact than the TIFF image and can be searched for arbitrary phrases and keywords by all of the standard PDF search engines.

The last-named method gives a means of generating PDF files without any recourse to some initial tagged, or WYSIWYG, source text. Capture performs image segmentation, classification, line identification, word recognition and optical character recognition, but does not attempt document classification or understanding. The program will often segment the page image during the course of its analysis but any geometric blocks created in this way will not generally be reflected in the translation to PDF. This is particularly true of pages composed entirely of text, where a single amorphous PDF page-object will be produced. Thus, Capture cannot, of itself, perform the breakdown into geometric blocks that corresponds to document analysis (as defined in the previous section). One exception to this rule is that Capture's recognition algorithms can distinguish items such as photographs, screen-dumps or complex diagrams — which need to be left in bitmap form. Such items will typically be converted to separate 'image objects' on the appropriate PDF page. However, there is no mechanism in PDF, as presently defined, for embedding any logical information about inter-object relationships into a PDF file.

One of the features of PDF is that, like PostScript, there are many different PDF programs that can have precisely the same rendered effect on the screen. A given PDF file might be structured to write out text a word at a time, a character at a time, or by some other algorithm that tries to optimise on-screen rendering. There is no requirement for PDF files to be in 'reading order' and, for multi-column text, it will often be the case that the file is rendered across the columns i.e. by hopping across the inter-column gutter.

In order to help application programmers to extract logical words from a PDF file (even if these have been hyphenated, say) Adobe Systems Inc. provide a software developers kit (SDK) which gives access via the application programme interface (API) of Acrobat viewers to the underlying portable document model which the viewer is holding in memory. There is a built-in algorithm inside the viewers which attempts to find all the logical words within a PDF document for string-search purposes. This algorithm is sound for all single-column documents but can get confused when presented with multi-column texts. Typically, errors occur when the inter-column gutter is very narrow; the word reconstruction algorithm may then, mistakenly, hop across the gutter. Furthermore, if the rightmost word of the rightmost column is hyphenated, the remainder of the word may be erroneously looked for in the leftmost column of the next line on the page.

The API also allows for the inspection of various features of the detected logical words e.g. typeface, font metrics, bounding box, logical content and position on the page. The analysis software described in subsequent sections was written in C++ as a 'plug-in' for Acrobat Exchange using the API.