ELECTRONIC ACCESS TO FULL DOCUMENT TEXT AND IMAGES THROUGH LINUS

Lorinda L. Cherry and Robert K. Waldstein

Lorinda L. Cherry *is a member of technical staff in the Computing Techniques Research Department and* ***Robert K. Waldstein*** *is a distinguished member of technical staff in the Operations Department of the Libraries and Informations Systems Center. They are with AT&T Bell Laboratories in Murray Hill, New Jersey. Ms. Cherry's research interests include electronic document distribution and access, and document preparation and graphics. She joined the company in 1966 and has a B.A. in mathematics from the University of Delaware and an M.S. in computer science from Stevens Institute of Technology. Mr. Waldstein develops and maintains a variety of systems that support library network functions and give users direct access to that network. He joined the company in 1981, and has a B.S. in mathematics from the* (continued on page 90)

This paper discusses document capture, preprocessing, and full-text display of technical papers with the LINUS database system, a centralized access point to a variety of databases. Users can retrieve records of interest—including documents—based on descriptions of the contents (e.g., author names, words of a title, content phrases from text). With the LINUS full-text service, they can easily check a fact or reference in a document, as well as preview an internal document at their terminals before deciding to order a paper copy. When `linus` is run from a bitmapped terminal, a document's graphics, tables, and complicated equations are also displayed; and a mouse and menus help users move around in the document. The full-text capability is available now, even over low-speed networks. Other applications being considered or implemented include displaying archival photographs and personnel pictures tied to a personnel directory.

Introduction

Libraries have long recognized and tried to meet the user's demand to have all materials of interest readily available. But several trends, especially for libraries like the AT&T Library Network, have combined to make this either difficult or impossible:

- The number of technical journals is rapidly increasing, so both storage space and subscription charges make it costly to maintain a complete collection for the user's hands-on use.
- The library network serves a geographically dispersed population. To have all the desired materials physically available to its users requires a high degree of redundancy, with its associated costs.

Even the fastest physical mail delivery is often considered unacceptably slow by many users. But today, electronic publishing and higher speed networks are making possible the capture and on-line presentation of a document's full text. (By *full text*, we mean the complete text of a paper with its tables, line art, graphs, and equations.) Thus, items of interest can be instantly available to users.

Libraries have been looking at electronic access as a way of solving these problems and best serving the customers' needs. Recently, this has focused on the idea of an "electronic window" that gives users access to as many library functions as possible without their physically going to the library.[1,2] The LINUS database system, which runs on the UNIX® system, currently provides on-line access to library catalogs and information collections, such as descriptions of archived photographs and a personnel directory. Making full text available, either for reading on-line or for electronic delivery to a user, further expands this concept of a window to library services.

The full text of papers for on-line reading is stored in database directories as input to a page-description language. This input includes the paper's graphics, tables, and complicated, displayed equations, which are preprocessed into bitmaps. A ***bitmap*** is a binary representation of an image and can be displayed on "smart" (bitmapped) terminals, such as the Teletype® 5620 dot-mapped display (DMD) terminal or AT&T 630 multitasking terminal with graphics (MTG).

A program on the host computer interprets the page-description language and currently supports the `troff`, Monk, and TEX™ formatting languages.[3–5] (TEX is a trademark of the American Mathematical Society.) When `linus` is run from a bitmapped terminal, a program called `reader` is downloaded into a window on the terminal and manages how text is positioned in that window. Menus displayed with the mouse enable the user to move around in the document. When `linus` is run from a "dumb" terminal, a less-sophisticated page-description language interpreter is used to display the text sequentially, without the bitmapped images.

In this paper, we discuss a method of capturing full text with minimal additional overhead either to the library network or to an author or document producer. The technique described has two advantages: Because the full text is captured in ASCII form, the text can be used for in-depth subject access. (ASCII is the American Standard Code for Information Interchange.) And, the system gives a user reasonable access over existing, low-speed networks. (Panel 1 identifies acronyms and UNIX system tools used in this paper.)

Life of a Paper

Technical papers are vital to an R&D organization; in some sense, they are its product. The papers serve two major functions:

- They are a record, for legal and other purposes, of the organization's work and achievements.
- They promulgate information about each person's work to his or her colleagues.

Figure 1 summarizes the mechanism used to capture the research documents, both electronically and as physical copy. The rest of this section describes this process, focusing on the flow of the electronic version.

In AT&T, two production methods dominate when an author is ready to produce a final version of a technical paper. These methods apply whether an author, secretary, or typing pool does the production:

- Use a formatter that runs on the UNIX system to produce the document. Currently, `troff` is the dominant formatter used to produce technical documents, although the `tex` formatter also has a significant user population.
- Use word-processing software that runs on a personal computer (PC) to produce the document, or use a special-purpose, word-processing machine. Technical papers are produced on a wide variety of PC-based tools, including WordPerfect®, Microsoft® Word, MacWrite™, and SAMNA Word™ word-processing software. (WordPerfect is a registered trademark of WordPerfect Corporation; Microsoft is a registered trademark of Microsoft Corporation; MacWrite is a trademark of Apple Computer, Inc.; and SAMNA Word is a trademark of SAMNA Corporation.)

Panel 1. Acronyms and Tools

ACM	Association for Computing Machinery
`areader`	translates `troff` or TEX code to ASCII form for display on a terminal
ASCII	American Standard Code for Information Interchange
`attextract`	installs a paper's full text in the ITDS database
CCITT	International Telegraph and Telephone Consultative Committee
`cip`	interactive drawing system
`chem`	phototypesets chemical diagrams
`cpio`	combines multiple UNIX system files into a single file for transmission to another machine
DMD	dot-mapped display (terminal)
`docsubmit`	enables author to submit a document's text to ITDS
`dvit`	translates a device-independent file into `troff`
`eqn`	language for describing mathematics
`grap`	language for describing and plotting graphs
`ideal`	language for describing and drawing diagrams
ITDS	Internal Technical Document Service
`itis`	ITDS internal documents database
`latex`	macro package for the `tex` formatter
`linus`	login that gives users on-line access to ITDS databases
LINUS	AT&T Library Network database system
`me/mm/ms`	standard macro packages for `troff`
Monk	database driven text formatter
`monk`	style-sheet based formatting program
MTG	multitasking terminal with graphics
PC	personal computer
`ped`	language for describing and drawing pictures
`pic`	language for describing and drawing pictures
`plot`	language for drawing graphs and pictures
`prefer`	retrieves and formats bibliographic citations in the `monk` style
`proof`	presents `troff` output for viewing on a bitmapped terminal
OCR	optical-character recognition
`reader`	manages the display of full text on a bitmapped terminal
`refer`	manages bibliographic citations
`tbl`	language for describing and drawing tables
`tex`	formatter that produces a device-independent file
`tped`	translates `ped` output into `troff`
`troff`	typesetting language and text formatter
`trplot`	translates `plot` output into `troff`

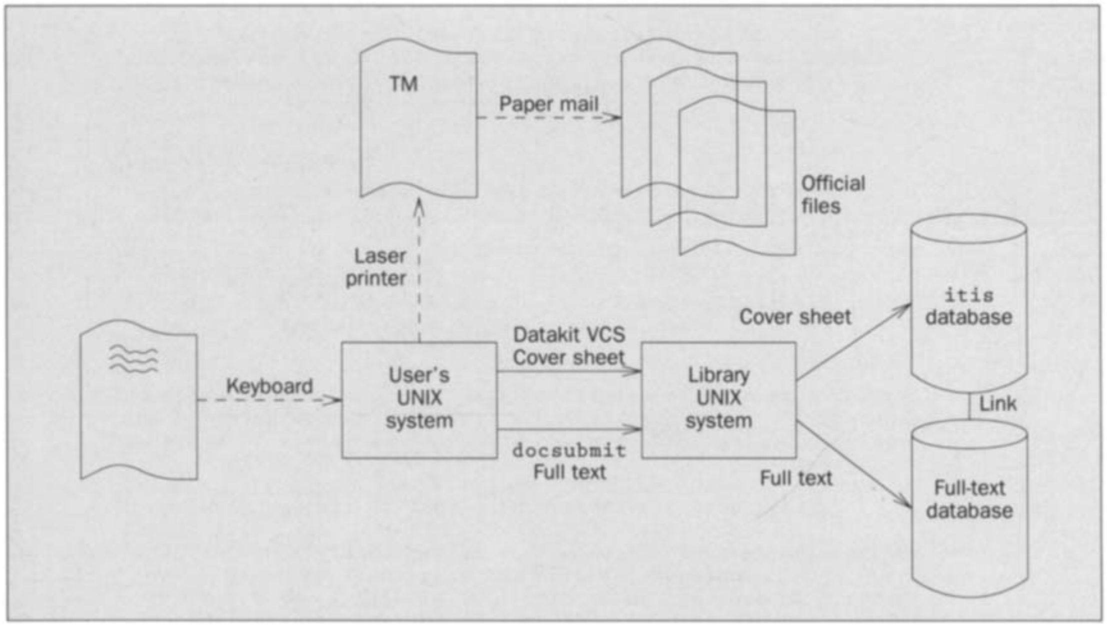


Figure 1. Life cycle of a technical paper. Datakit® VCS is a virtual-circuit switched network, `itis` is the ITDS bibliographic database, and TM is a technical memorandum.

Currently, the full text produced by these PC packages or the word-processing machines is not being captured because of several problems connected with the document's capture. Not the least of the problems is the need to give these users a tool that is simple to use in their computer environments and will transmit a packaged version of the text.

Once a fully acceptable version of the paper exists, a final "official" copy of the document is produced on a quality printer. This paper copy is sent to the Internal Technical Document Service (ITDS), which maintains the official, legal files of technical documents at AT&T.

For a paper to be accepted (and filed), the ITDS must have an official cover sheet for the document. This is a cover page that contains the paper's bibliographic information (e.g., title, author, number of pages), information about its release, and the abstract. Every document filed must also have a unique document number, which appears on the cover sheet. The cover sheet is captured electronically, as much as possible; those not received electronically are entered by the ITDS document check-in staff. Information for this paper—i.e., its bibliographic description and abstract—and for all documents filed with ITDS is stored in one database, known as the `itis` database. This database is one of several accessed via the LINUS system.

Electronic capture of full text of documents filed with ITDS is a relatively new step in the life of a document. After a document is filed with the ITDS and entered in the `itis` database, electronic mail is sent automatically to the author to request the document's full text. If we expect authors to cooperate in this capture, the process for submitting the full text must require minimal effort on their part. So, versatility and ease of use are crucial to the command they would use to do this. (We will describe the `docsubmit` command shortly.)

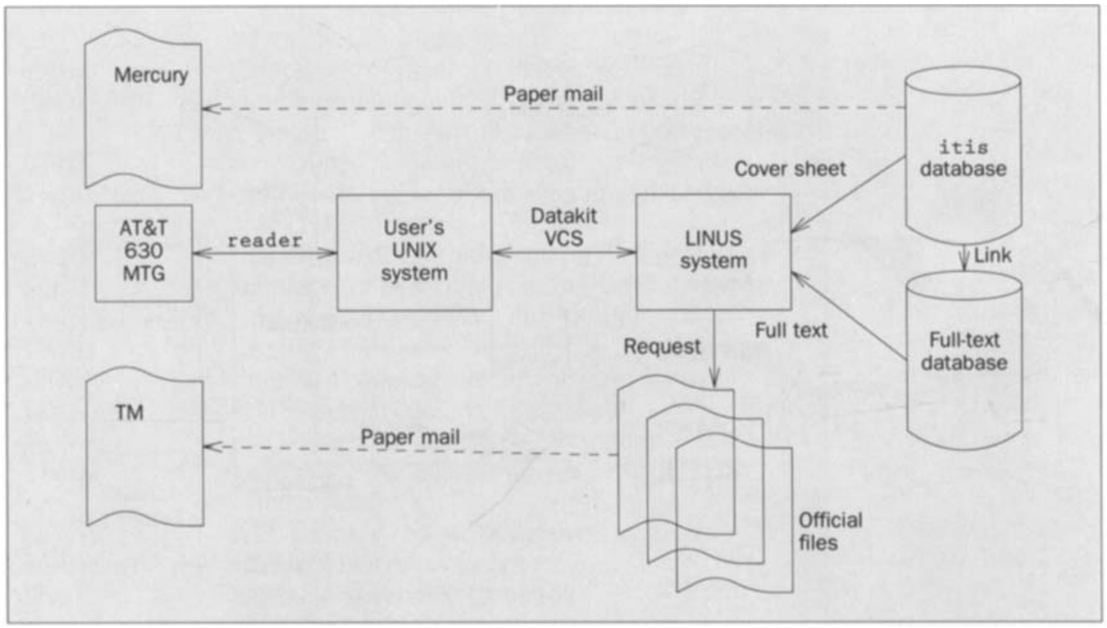


Figure 2. User access to technical papers. Datakit VCS is a virtual-circuit switched network, `itis` is the ITDS bibliographic database, Mercury is a bulletin service that alerts readers about new books and papers, and TM is a technical memorandum. The 630 MTG is a multitasking terminal with graphics.

When ITDS receives the full text, the LINUS system links the full text to its bibliographic record in the `itis` database. If a corresponding bibliographic record cannot be found, then the full text for that document is not made available on-line. Such failures occur if the document number cannot be identified in the full text, or if the document number extracted from the full text does not match the one in the `itis` database.

At this point, ITDS takes primary responsibility for handling the document, which involves two, sometimes contradictory, roles:

- ITDS is responsible for protecting and storing the proprietary technical information produced within AT&T.
- It is responsible for disseminating as much information as possible about technical documents to ensure that all employees are kept informed of pertinent work.

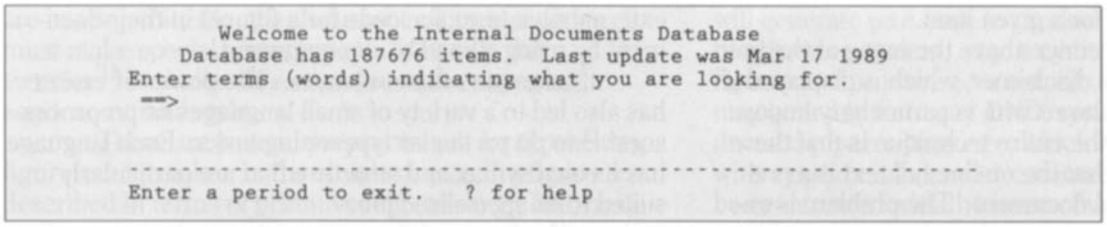
As Figure 2 shows, ITDS has two primary ways of keeping employees informed about technical documents of potential interest:

- An ITDS service, called Mercury, alerts users about new documents that have been filed. This paper bulletin describes the documents, separated by subject category. Mercury bulletins are generated from the `itis` database.
- AT&T employees can search the `itis` database directly using the LINUS service. This mode of access is discussed further in the next section.

LINUS Access to the `itis` Database

In some sense, the way users gain access to the full text is peripheral to this paper. But because the whole point of this effort is to give users access, the way this is achieved within AT&T is of interest. Therefore, this section is a brief overview of how users gain access to the `itis` database and to the full text, when available.

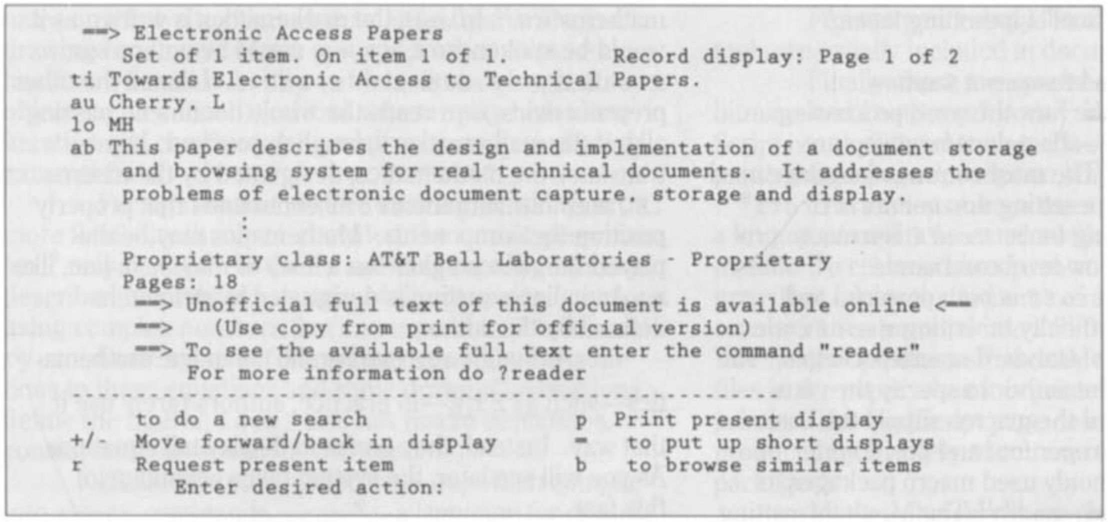
The AT&T Library Network, of which ITDS is a



```
          Welcome to the Internal Documents Database
       Database has 187676 items.  Last update was Mar 17 1989
   Enter terms (words) indicating what you are looking for.
    ==>

   -----------------------------------------------------------------
   Enter a period to exit.   ? for help
```

Figure 3. Main screen of the user mode of `slimsrch`. The terms a user enters are used to locate appropriate database records.



```
   ==> Electronic Access Papers
       Set of 1 item. On item 1 of 1.          Record display: Page 1 of 2
   ti Towards Electronic Access to Technical Papers.
   au Cherry, L
   lo MH
   ab This paper describes the design and implementation of a document storage
       and browsing system for real technical documents.  It addresses the
       problems of electronic document capture, storage and display.
                        :
       Proprietary class: AT&T Bell Laboratories - Proprietary
       Pages: 18
       ==> Unofficial full text of this document is available online <==
       ==>   (Use copy from print for official version)
       ==> To see the available full text enter the command "reader"
            For more information do ?reader
   -------------------------------------------------------------
   .   to do a new search                   p   Print present display
   +/- Move forward/back in display         =   to put up short displays
   r   Request present item                 b   to browse similar items
         Enter desired action:
```

Figure 4. Full display screen of a record. A disclaimer is displayed above the menu at the bottom.

component, is committed to maximizing employee access to a wide range of information by electronic means.[6] This is summarized by the "vision" statement: *The vision of the library network is to provide all professional employees throughout AT&T with an "electronic window" to the vast array of internal and external information services and to assure that the underlying information resources are managed as strategic assets providing a competitive advantage to AT&T.*

A major component of this electronic access is achieved through the LINUS system. LINUS is a library network service that gives users access to a wide variety of AT&T internal databases, including: the AT&T Library Network book and serial catalogs; the personnel, photograph, and standards databases; and the `itis` database of internal documents. (External databases are being considered.) These databases are all maintained under `slimmer`,[7] a database package developed internally to meet the needs of the library network.

When a user has gained access to LINUS, the system first confirms whether a bitmapped terminal is being used with the `reader` program loaded. The system then displays a menu of database options. When the user selects a database from the menu, he or she is presented with an initial screen—something like the one in Figure 3. By entering terms, such as a name or part of a title, the user does a search that retrieves relevant items. Figure 4 shows a typical display of the bibliographic entry retrieved for a given item.

Notice the disclaimer above the menu at the bottom of the display. This disclaimer, which is displayed if full text is available for the record, is particularly important. One drawback of the entire technique is that there is no way to guarantee that the on-line full text is a precise match to the official document. The problem is endemic to full-text systems, even when the capture technique involves getting "final" typesetting tapes.[8]

Word-Processing Tools and Document Capture

Here, we describe how the word-processing tools—`troff` and `tex`—affect document capture.

The `Troff` Tools. The most common language used on UNIX systems for typesetting documents is `troff`[9] and its suite of typesetting tools. `Troff` is a macro processor with a few, very low-level commands.

As a language, `troff` is both powerful and difficult to write. The difficulty in writing `troff` code has led to the wide use of standard macro packages. The macro packages allow the author to specify the components of the paper, and the macros supply the basic `troff` commands for proper font and positioning.

The most-commonly used macro packages in AT&T are `-ms`, `-mm`, and `-me`.[10–12] The Monk formatting language[5] is also used.

Monk is a high-level language in which the author describes the components of the paper with English-language commands like:

```
|author(name "L. L. Cherry",
  initials LLC, location MH,
  department 11271, extension X6067,
  room 2C-516)
```

The `monk` command then produces the proper `troff` commands for typesetting the paper.

Even when they use a macro package, authors can and do intersperse raw `troff` commands in their documents for fine-tuning. They may also include external files (e.g., the code for a figure) in their document by using `troff`'s `.so` command.

Languages and preprocessors. The power of `troff` has also led to a variety of small languages or preprocessors[4,13] to do particular typesetting tasks. Each language has its own syntax and structure that are particularly suited to its specialized job.

The first such language was `eqn` for typesetting mathematics.[14] In `eqn`, the mathematics is written as it would be spoken, so $x_i = y_i + z_i$ would be entered as: `x sub i = y sub i + z sub i`. Like all the other preprocessors, `eqn` reads the whole document, passing all but the mathematics through untouched. `Eqn` translates the mathematics, designated by the macros `.EQ` and `.EN`, into the `troff` commands that properly position the components. Mathematics may be displayed (i.e., placed alone on a line) or may be in-line, like x_i. An in-line equation is designated by delimiters defined by the author.

Although `eqn` can handle tall in-line mathematics—such as $\sum_i^\infty x_i y_i$—in practice, authors rarely use it that way. Instead, they prefer to display tall equations. As you will see later, the system takes advantage of this fact.

The preprocessor that specializes in tables is called `tbl`.[15] The author describes the layout and contents of the table, and `tbl` produces the `troff` commands to format it. Tables are delimited by the macros `.TS` and `.TE` and may contain mathematics. `Tbl` does not need to know how to format equations; it simply passes the mathematics through for `eqn` to process. Both `eqn` and `tbl` are in general use for producing technical documents.

`Refer`[16] is a program for managing bibliographic citations. The author may embed the references in the document; and `refer` can be told to collect them, format them, and place them at the end of the document. `Refer` can also be told—on the command line—to get the references from a separate file. When the references are in a separate file, the document capture program must make special arrangements to handle them. Prefer[17] is similar to refer but with entries in the monk command style.

There are several preprocessors for typesetting figures and graphs. In the simplest, pic,[18] a figure is described in terms of primitive objects—e.g., line, box, circle, and arrow—and positions, either as exact coordinates or relative to other objects. Pic figures are line drawings and may contain text and mathematics. Pic has its own macro facility (which allows complex objects to be defined), control structures (which allow iteration and conditionals), and provisions for including external files.

For figures that require shading, masking, or more flexibility than pic provides, the ideal preprocessor[19] may be used. In the ideal language, figures are described in terms of a system of simultaneous equations using complex numbers that correspond to points in an *x-y* coordinate system. The points defined by the solutions to these equations and some drawing instructions define the figure. Ideal also has macro definitions, control structures, and file inclusion.

Unlike the other preprocessors, which compile into troff commands, grap[20]—a language for describing and plotting graphs—compiles into pic code. Graphs are described in terms of primitives such as the type of frame that surrounds the graph, labels, line-drawing style, and points. Grap has macro processing, control flow, and file inclusion. Points to be plotted are often stored in external data files.

Another preprocessor that compiles into pic code is chem,[21] a language for describing chemical structure diagrams. It describes the diagram in terms of primitives—such as atom, ring, bond—and their connections. Chem figures may include pic macro definitions, text, and eqn expressions.

Besides writing in a preprocessor language to make figures, authors may use interactive programs to generate them. For example, the S statistical package will generate pic code to produce graphs. The interactive drawing system, cip, may be used to construct figures and produce pic output. Figures and graphs may be made from any UNIX system program that produces the plot language, which is then preprocessed with trplot into troff commands. Figures can also be constructed with the interactive graphics editor, ped,[22] and preprocessed into troff with tped.

Figures generated interactively with any of these tools are usually included in documents as external files.

Finally, there are special macros for including bitmaps and PostScript® code[23] in documents. (PostScript is a registered trademark of Adobe Systems Incorporated for a page-description language.)

The bitmap macros run a program that converts a bitmap, stored in an external file, into troff commands. To create a bitmap, an author usually runs a program that copies the contents of a window or the whole screen from a terminal into a file on the host computer. The PostScript macros[24] include the external PostScript files in the paper when it is printed on a PostScript printer. The PostScript page-description language is produced by a variety of commercially available graphic packages.

Many technical papers use several of the preprocessors described above; and in this computing environment, each preprocessor must read the whole document. As a result, authors often go to great lengths to speed draft production. They store and process figures separately and build clever commands to avoid having the preprocessors see the entire document. The multifile nature of troff documents complicates document capture.

The Tex Formatter. The other document-description language used to typeset documents is tex.[6] Unlike troff, the tex compiler knows about page layout, mathematics, tables, and graphics. Like troff, the tex language is difficult to program, and authors usually use a standard set of macro commands called the LATEX system.[25] In latex, the author identifies the components of the paper and the tex commands are generated for font

Table I. Ways to Submit a Paper

Sample command	Action
`docsubmit -C p?`	paper is stored in files `p?` and includes the coversheet
`docsubmit -c cover p?`	cover sheet is stored in file `cover`, paper is in files `p?`
`docsubmit -C -r refer.d p?`	paper with cover sheet in `p?` uses `refer` file `refer.d`
`docsubmit -C -t paper`	paper is `tex` input stored in `paper.tex` (used with `tex` or `latex` macros)
`docsubmit -N 11271-890228-01TMS p?`	document number is not in files `p?`

change and positioning. Although the tex language does have commands to include external files, generally tex documents are stored in one file. One exception to this is bibliographic files, which are processed by the bibtex program. Another is that authors who write with tex macros often use the more powerful troff graphics preprocessors to include figures in their documents, storing them in external files. PostScript figures may also be included in tex documents.

Document Capture—docsubmit

Submitting the full text of a technical document is an additional step in the publication process. To gain the author's cooperation, this step needs to be as simple and unobtrusive as possible. The docsubmit command was created to simplify document capture by the ITDS and has been distributed throughout AT&T. By using this command, authors can easily send their full text to the ITDS database, without any special manipulation of the files used to produce the document.

As we mentioned earlier, authors often store their papers in many files and automate the process of typesetting the paper. The goal of docsubmit was to make it easy to send a paper to the database. For example, docsubmit looks inside the paper to find components that a preprocessor would include automatically. The author does not need to specify these external files on the command line.

Docsubmit packages the paper into one cpio file and sends that file to the database spool directory. To link the paper in the ITDS full-text database to the entry in the itis bibliographic database, the system must know the document number. So, docsubmit must be told either where to find the cover sheet, or what the document number is and what filenames a user would type on the command line with troff or tex when typesetting the paper. Table I gives examples of docsubmit commands.

To process the full text, docsubmit first creates a temporary directory and a temporary file. In this file, it stores the date, the user's electronic mail address, and the document's number (if it was included on the command line). It also notes if the command line specified an external refer or prefer file. Then, the files that make up the paper are copied into the temporary file.

Capture of Troff Files. When copying troff files, if docsubmit encounters .so commands, it selectively copies the external user files that are included in the paper, replacing the commands. If a .so command is meant to include a macro package, docsubmit simply copies the command.

Other commands are examined to make sure they are not preprocessor output. Preprocessor output usually begins with a troff command that contains the name of the input file. If docsubmit finds such a command in the file, it looks for the preprocessor input file and uses that instead. But if it can't find the input file or detects preprocessor output (i.e., it finds a pattern of commands that are characteristic to a preprocessor), then `docsubmit` issues an error message to the author and refuses the paper.

While it is copying either the files that make up the paper or external files, `docsubmit` must look for other files that may be needed to capture the complete paper. If it encounters a macro to include bitmaps or PostScript files, it copies the bitmap or PostScript file to the temporary directory. If the filename specified contains the path to that file, it also rewrites the command without the path. So, for example,

```
        .BP figures/figure1.ps
```

would be rewritten as:

```
            .BP figure1.ps
```

The path must be deleted because, on the database machine, the external file will be in the same directory as the paper.

`Docsubmit` also looks inside macro definitions for the bitmap or PostScript macros and remembers the names of these user-defined macros. If it encounters the macros in the paper, then the files that would be used are copied to the temporary directory.

The graphics preprocessors—`pic`, `grap`, `ideal`, and `tped`—all may include external program or data files. Such files are copied into the text file in place of the macro command to include them, or are copied to the temporary directory. For example, if `docsubmit` finds `pic`'s macro of the form:

```
            .PS <figure.pic
```

it copies the file `figure.pic` in place of the command. It may also find `pic`'s `include` or `copy thru` command either in the file that it is copying or in the body of the paper. If so, it copies the referenced file to the temporary directory and rewrites the command to contain a simple filename rather than a path.

Finally, external `refer` or `prefer` files are placed at the end of the file, marked with comments.

Capture of `Tex` Files. For `tex` papers, `docsubmit` also copies the paper to a temporary file, selectively resolving the `tex` commands `\input` and `\include`. Any such commands are replaced by a copy of the referenced file. When `docsubmit` encounters the `tex` commands `\bibliography` (identifies a bibliographic file), `\special` (identifies a PostScript file), and `\includepicture` (identifies a `troff` file), it copies the referenced file to the temporary directory. `Docsubmit` also looks inside user-defined macros, specified by `\newcommand` or `\def`, for commands to include other `tex`, PostScript, or `troff` files.

Transmitting the Full Text. When this capture is completed for either a `troff` or `tex` paper, a `cpio` file is made from files in the temporary directory. The combined file is then sent with the `uucp` command to the spool directory on the library's UNIX system.

Structure of the Full-Text Database

Each paper is stored in a separate directory on the database machine. The papers are processed and stored in their `troff` or `tex` input form, along with some data files and preprocessor input files that are computed when the paper is installed in the database. These additional files are needed for `linus` and `reader`.

All `troff` preprocessor input or `tex` commands that would generate complicated motion or graphics are culled from the body of the paper and stored in separate files. Each of these files is also stored in its resulting bitmap form, as well for use by `reader`.

Installing a Paper—`attextract`

The task of installing a paper in the ITDS database involves several steps. A directory is created for the paper and the cpio file is expanded into individual files that docsubmit created. Next, the title of the paper, authors' names, and document number are extracted and put in a file for LINUS.

For the full text to be accessible, it must be linked to the corresponding bibliographic record in the itis database. This task is easier and most reliable if the document number is an easily identifiable part of the text or was specified on the command line when the paper was submitted. Usually the document number can be found, although where it is found and in what form varies considerably among different AT&T organizations. Also, a document commonly obtains or changes its document number after the full text is considered stable. So, if a paper's document number fails to match any itis entry, the paper will be linked to its bibliographic record, based on exact matches of the title and authors extracted from the full text.

Next, any figures, tables, graphics and complicated equations are preprocessed into bitmaps and replaced in the paper by pointers to both their source file and the resulting bitmap file. Finally, the section headings and their locations are written to a file for reader.

The procedure for installing either troff or tex papers is the same, although the details differ considerably.

Troff Papers. Generally, attextract copies the file that contains the paper. But as it does so, it writes each preprocessor input to a separate file, replacing that input with a .so command (so that the file can be included if paper copy is desired) and a .bm command (so that reader can include the bitmap created from the file). Because the .bm command is not a valid troff command, troff will ignore it.

The sequential nature of information in a document complicates processing each equation, table, or figure separately. For example, at some point in the paper, the eqn delimiters may be changed or turned off. Several of the preprocessors have their own *define* mechanism that must be in the proper state when the figure is processed. Troff string definitions must be available in the figures. The rest of this section discusses how the information for each preprocessor is handled.

Because equations can appear in tables and graphic figures, eqn definitions and troff string definitions are saved in a file that is included with each equation or preprocessor file when the bitmap is made. Displayed equations—such as fractions, matrices, and summations—that produce complicated troff motions are copied to a separate file; contiguous displayed equations are placed in a single file. If an eqn definition contains a construct that would cause motion, the definition is remembered and equations that contain this definition are also copied to a file. So, if a paper contained the definition:

```
define sumof # sum from i=0 to infinity #
```

then the following equation would be copied:

```
.EQ
zeta(x) = sumof x sub i
.EN
```

because it contains sumof.

Because neither tbl nor tped has any memory from one table or figure to the next, processing the files separately is straightforward. For each table or figure, the current value of the eqn delimiters, eqn definitions, and troff string definitions are included to make the bitmap.

Pic, grap, and ideal have macro definitions and parameters that may be remembered, redefined, or deleted from one figure to the next. These definitions and parameter assignments are saved separately for each figure. As the bitmaps are computed, the definitions are accumulated and included with the code for the figure, along with the state of the eqn delimiters and definitions.

Any commands to include PostScript files are written to a file and converted to bitmaps with a PostScript interpreter.

Tex Papers. Because `tex` commands can appear anywhere on a line, `attextract` first rewrites the file to place all the `tex` commands of interest at the beginning of a line. This step greatly simplifies further processing. Although the `tex` program is monolithic, conceptually its math mode is similar to `troff`'s `eqn` preprocessor. In-line `tex` math mode is entered with the character `$`. For displayed equations, math mode may be entered with either the characters `$$` or the commands `\[`, `\begin{equation}`, `\begin{displaymath}`, or `\begin{eqnarray}`. In `tex` papers, as it did for their `troff` counterparts, `attextract` writes tables, figures, and any equations that have complicated motions to separate files that are processed into bitmaps. So, if `attextract` encounters:

```
$$
a \over b
$$
```

it writes that code to a file (e.g., `eqn1`) and replaces it with:

```
\input{eqn1}
\bitmap{eqn1.bm}
```

The `tex` command `\input` causes the equation to be included if paper copy of the full text is made, and the `\bitmap` command signals `reader` to display the bitmap file. `Attextract` collects all text it sees before either a `latex` `\begin{document}` command or a `tex` `\centerline` command, placing it into a file that will be used to make the bitmaps. It similarly collects any macro definitions found in the body of the paper.

Both the `tex` and `latex` macro definitions are recognized. If the macro contains mathematics that produces motions, the name of the macro and the number of arguments it takes are remembered. So, if the paper contained the definition:

```
\newcommand{\aoverb}{a \over b}
```

any equation that contains `\aoverb` is saved for processing.

Bitmaps are a little harder to make in `tex` than in `troff`, because `tex` knows about page layout and insists on a strict structure. To subvert such requirements, `attextract` must create a skeleton file that conforms to `tex`'s structural demands and contains a `\input` command in which the name is changed for each file to be processed into a bitmap.

In addition to displayed equations, `attextract` collects graphics (marked by `\begin{picture}`), tables (marked by `\begin{tabular}` or `\begin{tabbing}`), and figures (marked by `\begin{figure}`). The `\includepicture` command is used to include a `troff` input file in a `tex` paper; the `\special` command is used to include a PostScript file.

Making Bitmaps. The bitmaps are made with a program that reads the `troff` output produced by the `troff` preprocessors and `troff` or by `tex` and `dvit`. The bitmap program runs on the library's mainframe computer but creates a compressed bitmap file for display on the bitmapped terminal. Bitmaps for PostScript files are produced by a PostScript interpreter that also runs on the mainframe computer.

Data Files. For both `troff` and `tex` papers, `attextract` creates several files that are used by LINUS and `reader`. It puts the path, title, author, and date of the paper in a file. This is easy if the author used a standard `troff` macro package or `latex` macros. If not, then centered or displayed text at the beginning of the document is captured as probably containing the title and authors' names.

`Attextract` also creates a file of section headings and their location; `reader` will use this information for a menu that allows a user to move around in the paper by heading.

Finally, `attextract` computes a file of phrases from the paper that are likely to be useful for searching the database by subject. Included are the title, nongeneric section headings, author-specified keywords, and

Table II. Distribution of Papers

	Preprocessor use (%)	
	With seed papers	Without seed papers
No preprocessors	40	44
1 preprocessor	36	37
2 preprocessors	19	16
More than 2 preprocessors	5	3

frequent noun phrases found by a version of the `topic` program.[26]

Sizes and Statistics. As of March 8, 1989, the ITDS database contained 1071 papers from 126 different UNIX systems. Of these, 195 were papers from the AT&T Bell Laboratories Center 1127 (Computing Research) database used to seed the ITDS database. About 60 percent of the papers have computed bitmaps. This number drops by only 4 percent if the seed papers are excluded.

Table II shows the distribution of papers by number of preprocessors used. In the table, use of `eqn` really means use of complicated mathematics; the measure is *were* `eqn` *bitmaps created*, not *was* `eqn` *used for simple mathematics*. The most popular preprocessor combinations are `tbl+pic` and `eqn+tbl`.

For papers that had bitmaps, the average number of bitmap files is 11, while the median is 5. What these statistics reflect is that the database has several very large papers with many bitmaps. Figure 5 shows the distribution of bitmaps per paper.

Although the average bitmap file has 4727 bytes, the size of the bitmaps varies among the preprocessors. For example, the average `eqn` bitmap file is 1123 bytes; for `pic` and `tbl`, the averages are 6995 and 6700 bytes, respectively.

Table III gives sizes and number of files for six

Figure 5. Distribution of bitmaps in the ITDS database.

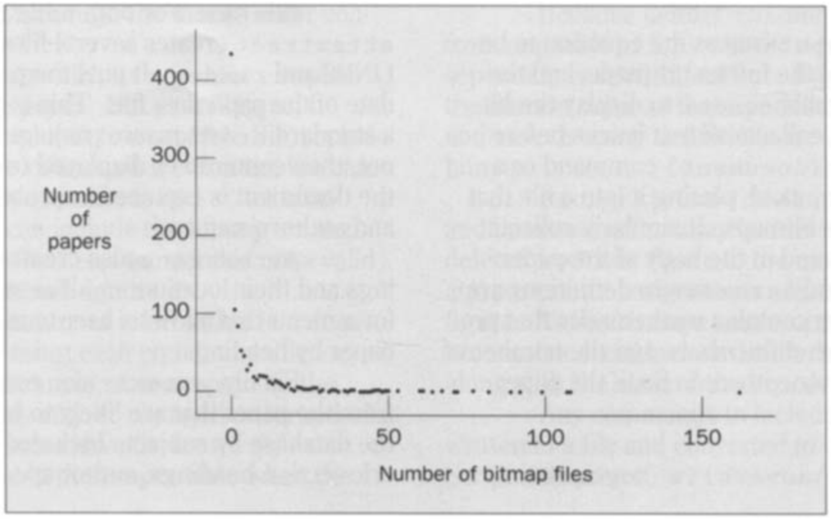


Table III. Size Comparisons in Bytes

	Input				Output		
Paper	troff/tex input	Bitmap files	Computed bitmaps	Total size	troff/tex output	PostScript input	Encoded scanned image
1	22,829	4	4,537	27,365	70,257	75,195	437,500
2	27,198	11	29,523	56,721	82,631	78,767	750,000
3	22,975	9	30,547	53,522	96,508	83,147	562,500
4	58,438	7	51,332	109,770	179,303	147,227	1,000,000
5	91,745	13	48,076	139,821	70,832	155,638	1,062,500
6	88,784	34	147,972	236,756	103,532	598,745	1,875,000

papers in the database. This is what the papers represent:
1. `Troff` paper—low end. Has two small `eqn` bitmaps and two small `tbl` bitmaps.
2. `Troff` paper—medium end. Has six medium-size `eqn` bitmaps and five average-size figure bitmaps.
3. `Troff` paper. Has eight tables and one figure.
4. `Troff` paper. Has six tables, one of which is very large.
5. `Tex` paper. Has two small equations, two small tables, and nine figures.
6. `Tex` paper. Has 24 small equations and 10 PostScript figures.

The sizes listed for "encoded scanned image" assume a resolution of 400 dots per inch with the CCITT-facsimile encoding discussed in reference 18. (CCITT is the International Telegraph and Telephone Consultative Committee.) A paper stored in `troff` input form with bitmaps consumes about 8 to 16 times less space than the encoded image of the paper. If we look at this another way, 45 percent of the documents stored in `troff` input form take up less space than one scanned page; 81 percent take up less space than three scanned pages. We excluded cover sheets in all these calculations.

`Reader` and LINUS

A user may access the ITDS database and read full text either from a bitmapped terminal that is running `reader` or from a dumb terminal. Before a user connects to the LINUS system from a bitmapped terminal, `reader` must have been downloaded into a window on the terminal. This is necessary because programs cannot currently be downloaded over the network.

When a user logs in as `linus`, the LINUS system asks the terminal if it is talking to a `reader` window. If it is, then LINUS will invoke `reader` when the user asks to read full text. If it is not a `reader` window, then `areader` will be invoked instead. `Areader` is a program that translates `troff` or `tex` code to ASCII form and presents the full text sequentially.

The rest of this section describes `reader.`

`Reader` consists of a *host program* that runs on the mainframe computer, and a *terminal program* that runs in either a Teletype 5620 or AT&T 630 terminal.

The host program recognizes the structural components of the paper and basic commands that cause white space and line breaks. It sends commands and the text to the terminal program. The terminal program decides how to display the paper's components in the window. The terminal program is the same for both the `troff` and `tex` versions of `reader`, although there is an extra font for `tex`'s special characters.

Some commands sent from the host to the terminal are: title, heading, paragraph, word, tab, display, in eqn, sub, sup, center, bitmap. So, for example, when the terminal program sees a title command, it changes the font to bold and collects text to be centered. When it sees a paragraph command, it indents.

Fonts are compiled into the terminal program and not downloaded, as they are in `proof` (a UNIX system program that simulates `troff` output for viewing on a 5620 terminal). A 12-point Palatino®-Roman font is the terminal program's regular font; but where appropriate, the 12-point Palatino-Bold and -Italic fonts, 9-point Palatino-Roman font, and a 12-point special font are also used. (Palatino is a registered trademark of Linotype Company.) Although subscripts and superscripts are traditionally set in italics, `reader` uses the small, regular font, which is much more readable at the terminal's resolution.

The terminal's mouse enables the user to control how the display is updated. The button-3 menu permits a user to move forward, `more`, or backward, `less`, or quit reading the paper, `done`. If possible, the terminal program updates the display on paragraph boundaries. It moves the beginning of the last uncompleted paragraph to the top of the window. To mark the line of text that was on the bottom of the previous screen, it puts a line in the bar at the left edge of the window. If a bitmap was at the bottom of the window, the top of the bitmap is moved to the top of the window. This keeps the bitmap on the screen with the caption or descriptive text that follows it.

The mouse's button-1 menu is loaded with the section headings in the paper, and the user may select a heading at any time. Button 2 is used to exit from `reader`.

Because the terminal program maintains a two-window buffer, the display update is almost instantaneous if the user is moving sequentially through the paper. However, the `more` selection on the button-3 menu is active only when the second buffer is full and a "cherries" icon is displayed. (The cherries indicate that the program is waiting for the user to do something.) An inactive menu item is enclosed in parenthesis. The other button-3 selections are always active.

A `The End` icon is displayed at the end of a paper. The user may quit with the `done` selection on button 3, or go back into the paper by choosing a section heading on button 1 or `less` on button 3.

When the user selects `done`, the host program exits, and the terminal program changes to the default font and becomes a terminal emulator. The terminal program remains active in the window until `exit` is selected on button 2. Thus, the program does not have to be downloaded when `reader` is used later. Button 2 should not be selected in the middle of a LINUS session.

Although most communications between `reader`'s two parts are from the host to the terminal, button selection requires messages from the terminal to the host. To minimize delay in its response to button selection, the host program in the original version of `reader` did very little buffering of its output; instead, it queried the terminal frequently for status. The speed of this design was satisfactory when all communications were done on one machine, but proved too slow when done over the network. Buffering on the host side improved the speed over the network but slowed the program's responsiveness a bit to button selection. On a button selection, the terminal program must throw away characters from the host until it sees a status query.

`Reader` and Psychology—Design Issues

`Reader` displays the text and figures of a paper in a window. Unlike `proof`, which displays a "window" into the bitmap image of a page, `reader` displays the text in whatever size window was used to load it, at a resolution appropriate for the terminal.

We expended every effort to make the text as readable as possible. By *readable,* we mean the visual ease of reading the text. Many factors contribute to making text readable, including: font design; letter and interword spacing; point size, line spacing, and line length; paragraph indicators, and general layout. Psychologists have long studied how these factors affect the ease and speed of reading text on paper and, more recently, of reading text on screens. Where possible, we used the results of these studies to decide how `reader` would present the text to users.

`Reader` uses a few fonts from the Palatino font family and the terminal's default Pellucinda font. We selected the Palatino family for several reasons.

First, it is a proportionally spaced font. This means that the characters are different widths, unlike typewriter fonts where the characters all have the same width. Beldie, Pastoor, and Schwarz found[27] that reading speeds were higher with proportionally spaced fonts than with constant-width fonts. Also, if `reader` uses a proportionally spaced font for the text, it can also display programs and variables in the constant-width font, as they would appear on paper.

Second, the Palatino family is a book font, not a newspaper font. Book fonts are designed to be very readable and are intended for use in long lines. The characters are wide and the ascenders and descenders are long. Newspaper fonts, like the Times® Roman font, are designed to conserve space and are intended for use in short lines. (Times is a registered trademark of Linotype Company.) Generally, their characters are narrow and the ascenders and descenders are short. Although Treurniet found[28] in a searching test that subjects missed fewer targets and performed faster when fonts had 1- or 2-pixel descenders than for fonts with 0-pixel descenders, the resolution of the characters was not large enough to conclude that longer is better. (A *pixel* is the smallest display unit on a video screen.) However, longer descenders accentuate both word and letter shape, which many believe is why lower-case text is read faster than upper-case text.

Finally, we selected the Palatino family because it has serifs. Bigelow points out[29] that serifs act as flags on the character shapes, thus enhancing letter discrimination and improving text readability.

We selected 12 point as the standard font size for `reader`, with subscripts and superscripts set in 9 point. Palatino 12-point type has an x-height (the distance from the baseline to the top of a lower-case "x") of 8 pixels. This is in the middle of the range Bigelow suggested for characters read at the typical viewing distance of screens. If the user is reading in a full-screen-width window, a typical line contains about 75 characters. Duchnicky and Kolers have shown[30] that reading is faster and more efficient for text lines of 80 characters than for text lines of 40 characters.

Studies that compared reading speed for justified text on paper and for unjustified text have shown little difference between the two, except possibly for poor readers. (In *justified* text, the interword space varies because extra space may be added between some words to produce straight left and right margins.) There is evidence, however, that regular word spacing increases reading speed.

`Reader` uses a 1-en (the width of a lower-case "n") interword space and an unjustified margin.

Cost and Benefit of Structural Components

There are four possible levels of abstraction at which `reader` could operate (*level of abstraction* means the type of information received and used to produce the display):

- Macros (the highest level)
- Raw `troff` input language
- `Troff` output
- Bitmaps or facsimile.

The last level cannot be searched by other programs, consumes vast amounts of space, and does not allow the program to control the display.

`Troff` output is also the wrong level of abstraction if the terminal is to control where text is placed in the window. Ignoring graphic commands, basic `troff` output selects a font, selects a size, and positions a character on the page. Most placement information is useless to a terminal program that wants to display the text in a window regardless of the window's size.

Although the raw `troff` input level is possible, the papers would have to be stored as preprocessor output or `reader` would have to run the preprocessors. Preprocessor output is huge compared to its input. For example, the three-character equation $a=b$ expands to

141 characters of `troff` output. If `reader` had to run the preprocessors before it could display a paper, it would be unusably slow. At the level of raw `troff` input, `reader` would also have to read the macro packages before it could begin to display the paper.

At the highest level, the macros, less information needs to be sent to the terminal, and the terminal program decides about fonts and positions for the paper's components. However, if authors also used raw `troff` commands in their paper, then this level will miss those commands and display the paper incorrectly. So, `reader` uses a hybrid of the macros and `troff` commands that cause breaks.

Because we chose a high level of abstraction, we could add `tex` papers to the system with no significant changes to the terminal program. (Other page-description languages could be added easily as well.) Both a cost and a benefit is that neither the `troff` nor `tex` version of `reader` needs to be a full implementation of the language to display accurate renditions of most papers. Let us ignore tables for the moment. `Reader` is only 20 percent of the size of the combined `troff` and `eqn` programs, and only 10 percent as large as the `tex` program. As a result, some features of the language are omitted. A paper that uses `troff` number registers to label figures or has private macros is not always displayed correctly. `Tex` automatic labels also are not done correctly. But in practice, little information seems to be lost by these omissions.

Images versus ASCII Representations

Two competing technologies can be used to capture and give users electronic access to full text. One consists of scanning the physical document and capturing the images of each page. The other involves capturing the document in character form during the production process (one variant of this method was described in this paper). Both techniques have advantages and disadvantages.

Scanned Images. In this approach, every page of every document is passed under a scanner. For internal documents, this is less onerous than it sounds because, for many years, all these documents have been scanned to create microfiche. In addition, the library network is now scanning all documents and storing the images on optical disk for request fulfillment purposes. Because these images are in machine-readable form, it becomes possible to use them to provide electronic access to full text.

This technique has several major advantages:

- All internal documents must be filed with ITDS. Therefore, this method's capture rate is perfect—100 percent of all documents will have their full text scanned.
- The scanned full text is an exact image of the document, which eliminates any questions about that image being the official full text.

There are also major disadvantages to using scanned images:

- The image's size is fixed. Thus, the user cannot be presented with a "page" that is optimal for the window he or she is using to read the text.
- The document's contents are not available for indexing and similar functions. These tasks would require high-quality optical-character recognition (OCR), a technology that is not yet available. Even an OCR unit with 99.9-percent accuracy would yield about one wrong character per page. This error rate probably would not be acceptable.
- The "reader" would be simply an image presenter. It offers nothing to help a user decide where to look in a document, except in terms of images from the beginning.
- A page's size (in terms of bits) is large, even if the image is compressed. Over today's networks, this would cause an unacceptable delay in presenting a page to a user.

ASCII Text. We have already described a method of presenting full text that involves capturing the text during the publication process. This technique has several important advantages:

- The captured full text is truly text (i.e., it is in ASCII form). Therefore, it can be used to enhance the retrieval process through indexing and related techniques.
- Because the image is constructed as it is being presented to the user, it can be optimized for the window being used.
- The reading program can use its knowledge of the document to improve the user's ability to read or browse the document quickly.
- Only a small number of bits need to be transmitted to present a page to the user. This is what makes `reader` acceptable at today's network rates of 19,200 baud and lower.

Unfortunately, this method of capturing full text also has major difficulties:

- Because the capture occurs during the publishing process, this adds a task for the author. Even the easiest and smoothest technique will be unacceptable to some authors.
- We cannot guarantee that the full text we captured corresponds exactly to the paper copy filed with ITDS. Because authors often use "cut and paste" methods and correction fluid to make last-minute changes, the full text most likely will not be an exact match.
- The full text captured by this technique will never be complete. Some document production methods prevent capturing the full text easily during the publication process.

In many ways, the advantages and disadvantages we described complement each other. Often, a weakness of one approach complements a feature of the other method. Because they complement each other, it is reasonable to consider using both techniques to give users the best access to documents.

Possible Future Services

The technique we have described for giving users electronic access to AT&T internal documents can be applied to any full text. It is already being used to provide access to internal publications, such as the *AT&T Technical Journal.*

A peripheral application arose once we had a user population that used the downloaded `reader` program. Because `reader` can function as an image presenter, these users were already set up to receive images on their terminals. This creates several opportunities. Among those being investigated are: augment the current database of photographs by making the image available, and add people's pictures to the AT&T personnel database. Both efforts are moving slowly because, unlike the electronic full text, we do not have machine-readable copies of these images to capture.

Conclusion

This paper described a method for giving users electronic access to the full text of internal documents. It addressed two needs:

- Capture the full text during the production process—which, at AT&T, involves author participation.
- Give the user an interface that allows him or her to peruse a document randomly, without unacceptable wait times.

In general, the method described is successful. The added step in the publication process was acceptable to most authors, so we are capturing the full text of a high percentage of documents. The reader program works well over many of the networks in use at AT&T. Even at 1200 baud, it gives acceptable response rates for reading or browsing.

References

1. E. Sloan, "The library of the future," *BACSpace*, March 1988, pp. 2-3.
2. W. D. Penniman, D. T. Hawkins, and E. Mount, "The Library Network at AT&T," *Science & Technology Libraries* Vol. 8, No. 2, Winter 1987/1988, pp. 3-24.
3. B. W. Kernighan, "The UNIX® System Document Preparation Tools: A Retrospective," *AT&T Technical Journal*, Vol. 68, No. 4, July/August 1989, pp. 5-20.
4. S. L. Murrel and T. J. Kowalski, "Monk: A High-Level Text Compiler," *AT&T Technical Journal*, Vol. 68, No. 4, July/August 1989, pp. 45-60.

5. D. E. Knuth, *TEX and METAFONT, New Directions in Typesetting*, Digital Press, Bedford, Massachusetts, 1979.
6. AT&T Library Network, "AT&T Library Network Annual Report," AT&T Bell Laboratories, Murray Hill, New Jersey, 1988.
7. R. K. Waldstein, "SLIMMER—a UNIX™ system based information retrieval system," *Reference Services Review*, Vol. 16, No. 1-2, 1988, pp. 69-76.
8. R. Pagell, "Searching full-text periodicals: how full is full?," *Database*, Vol. 10, No. 5, October 1987, pp. 33-38.
9. J. F. Ossanna, "NROFF/TROFF User's Manual," *UNIX™ Time-Sharing System: UNIX Programmer's Manual*, Seventh Edition, Volume 2, Holt, Rinehart and Winston, New York, January 1979, Section 12, pp. 196-229.
10. M. E. Lesk, "Typing documents on the UNIX system: using the –ms macros with troff and nroff," *UNIX™ Time-Sharing System: UNIX Programmer's Manual*, Seventh Edition, Volume 2, Bell Laboratories, Murray Hill, New Jersey, Holt, Rinehart and Winston, New York, January 1979, Section 8, pp. 125-145.
11. AT&T, *UNIX™ System V DOCUMENTER'S WORKBENCH™ Software, Technical Discussion and Reference Manual*, CIC No. 310-005, Issue 1, AT&T Customer Information Center, Indianapolis, Indiana, 1986.
12. E. P. Allman, "–ME Reference Manual," *UNIX Programmer's Manual 4.1 BSD User Document*, Seventh Edition, Vol. 2c, Computer Science Division, Department of Electrical Engineering and Computer Science, University of California, Berkeley, California, November 1980.
13. J. L. Bentley, "Little Languages for Pictures in AWK," *AT&T Technical Journal*, Vol. 68, No. 4, July/August 1989, pp. 100-110.
14. B. W. Kernighan and L. L. Cherry, "A System for Typesetting Mathematics," *Communications of the ACM*, Vol. 18, No. 3, March 1975, pp. 151-157.
15. M. E. Lesk, "TBL—A Program to Format Tables," *UNIX™ Time-Sharing System: UNIX Programmer's Manual*, Seventh Edition, Volume 2, Holt, Rinehart and Winston, New York, January 1979, Section 10, pp. 157-174.
16. M. E. Lesk, "Some Applications of Inverted Indexes on the UNIX System," *UNIX™ Time-Sharing System: UNIX Programmer's Manual*, Seventh Edition, Volume 2, Holt, Rinehart and Winston, New York, January 1979, Section 11, pp. 175-195.
17. *UNIX Programmer's Manual*, A. G. Hume (ed.), Tenth Edition, Vol. 2, AT&T Bell Laboratories, Murray Hill, New Jersey, to be published, 1989.
18. B. W. Kernighan, "PIC—A Language for Typesetting Graphics," *Software Practice & Experience*, Vol. 12, January 1982, pp. 1-20.
19. C. J. Van Wyk, "A High-Level Language for Specifying Pictures," *ACM Transactions on Graphics*, Vol. 1, No. 2, 1982, pp. 163-182.
20. J. L. Bentley and B. W. Kernighan, "GRAP—A Language for Typesetting Graphs," *Communications of the ACM*, Vol. 29, No. 8, August 1986, pp. 782-792.
21. J. L. Bentley, L. W. Jelinski, and B. W. Kernighan, "CHEM—A Program for Phototypesetting Chemical Diagrams," *Computers and Chemistry*, Vol. 11, No. 4, 1987, pp. 281-297.
22. T. Pavlidis, "PED Users Manual," Computing Science Technical Report No. 110, AT&T Bell Laboratories, Murray Hill, New Jersey, 1984.
23. Adobe Systems, *PostScript Language Reference Manual*, Addison-Wesley Publishing Company, Reading, Massachusetts, 1985.
24. N. P. Nelson and C. L'Hommedieu, "An Open Architecture Corporate Electronic Publishing Platform," *AT&T Technical Journal*, Vol. 68, No. 4, July/August 1989, pp. 100-110.
25. L. Lamport, "LATEX—A Document Preparation System," Addison-Wesley Publishing Company, Reading, Massachusetts, 1986.
26. L. Cherry, "Writing Tools," *IEEE Transactions on Communications*, Vol. Com-30, 1982, pp. 100-105.
27. I. P. Beldie, S. Pastoor, and E. Schwarz, "Fixed versus variable letter width for televised text," *Human Factors*, Vol. 25, 1983, pp. 273-277.
28. C. B. Mills and L. J. Weldon, "Reading Text from Computer Screens," University of Maryland, College Park, Maryland, 1986, pp. 15-17.
29. C. Bigelow, "Principles of Font Design for the Personal Workstation," Stanford University Stanford, California, 1984.
30. R. L. Duchnicky and P. A. Kolers, "Readability of text scrolled on visual display terminals as a function of window size," *Human Factors*, Vol. 25, pp. 683-692.

Biographies (continued)

University of Maryland, an M.S. in computer science from Cornell University, and both a M.L.S. and a Ph.D. in information studies from Syracuse University.

(Manuscript received May 1, 1989)