Domain-specific Information Browsers for Man Page, File, and Font*

Thomas A. Phelps
Department of Electrical Engineering and Computer Science
Computer Science Division
University of California, Berkeley

Abstract

The task of information browsers is (1) to aid in navigating through a large database to locate the item of interest and (2) to inspect or otherwise manipulate this item once found. This document describes three experiments in constructing tools for information browsing: TkMan for UNIX manual pages, NBT for files in a hierarchical file system, and FoSel for bitmap fonts. Each exploits the large-scale natural organization of data and the fine-grained structure of each datum with a graphical user interface to provide a powerful yet intuitive tool. The lessons learned in implementing the browsers point to general principles that should guide the design of all information browsers.

1 Domain-specific Information Browsing

One need not read Alvin Toffler [Tof90] to see that this is the of Age of Information. Personal computers will become windows into enormous repositories of knowledge. Today, CD ROMs hold encyclopedias replete with full text, color images, sound, music, animation and video. Newspapers and magazines are going online. Books are being digitized. Telephone companies are laying the information infrastruture, and large media companies are conducting high tech experiments to prepare for its completion. Faced with this oncoming flood of information, a major concern is how to determine what is available and how to access it effectively.

On a smaller scale, the individual UNIX computer user faces several types of information stores too large to negotiate effectively without program support. This paper defines information broadly, as any data with interesting internal content and, possibly, external organization. Thus the masses of numbers in scientific visualization are still data (the value of the number completely describes it) but fonts, where the name is merely a handle for a bag of shapes, are information.

Each information domain considered here—man page, file, font—consists of hundreds or thousands of members, each of which has this internal content to examine. On one level these information domains have little in common with each other and thus each requires a specialized tool to effectively survey it. On a deeper level, however, each information domain requires that its browser support certain functionality—functionality ought to be standard in a large range of information browsers.

*. This research is sponsored in part by the Advanced Research Projects Agency under Grant number MDA972-92-J-1028. The content of the information does not necessarily reflect the position or the policy of the Government. This research is also based in part upon work supported by the National Science Foundation under Infrastructure Grant No. CDA-8722788.

2 TkMan for Man Pages

It has long been a UNIX tradition to have online documentation for everything from programs to subroutines to file formats. These manual pages (“man pages” for short) detail the arcana of system use, such as reporting that the `-M` option to the `sort` command collates in month order. These minutiae are essential for any sophisticated use of the system but simply too numerous to memorize all but the most frequently used—indeed, a printout of the online documentation occupies an entire bookshelf. And once committed to paper, it grows stale, liable to be made obsolete with the next revision of the software.

With competent man page readers, however, one need neither memorize or make a hardcopy of the documentation. The information is fresh and available on one’s screen. Still, there is the enormous mass of text to confront, from which to extract the few lines which answer one’s needs. A typical UNIX system has access to over 5000 man pages—15,000,000 bytes of text. And each significant new software system, like Tcl/Tk or InterViews, adds 10s or 100s more, giving some systems access to 15,000 or more. Once one identifies the one out of the thousands, there is still the matter of finding the desired information within the man page, some of which run over a hundred pages.

Clearly, a powerful man page browser is vital tool. The following subsection describes how previous man readers have failed their users in some critical way and how next generation help systems are unsuitable for the task at hand, and the subsection following that outlines TkMan’s elegant solution. TkMan is the first experiment in domain-specific information browsers.

2.1 Related Work

2.1.1 Manual Page Readers: Text-based and Graphical

Text-based: `man`, Perl `man`, Emacs

The original man reader, `man` accepts the name of the manual page to view, finds it (if it exists) and pipes the output to a pager. The sophistication of `man` depends on the pager. With `more`, one may search for a regular expression, but not review any text that has already scrolled past. `less` searches and scrolls both forward and backward. Neither provides any special support for manual pages over that given to any ordinary ASCII document.

Perl man is “the result of a complete rewrite of the man system” with the three goals of “effect[ing] substantial gains in functionality, extensibility, and speed” [Chr87]. It combs through man pages cross checking for misnamed, misorganized and missing man pages. The results of all this labor are stored in a database to provide very rapid location of content location, bypassing a search through the file system. Perl man indexes long man pages to provide direct access to sections and subsections. To take advantage of the feature, though, one must first memorize or instruct Perl man to produce a list of available sections, then invoke it again with a few-letter key to identify the section.

Emacs [Sta87] can bring man pages into its editing environment, where there the user can take advantage of its wealth of capabilities. The user can search for text incrementally or with regular expressions, copy and paste text into another file or to a shell window within Emacs. Extension packages bring hypertext functionality. For those who conduct most of their computing within Emacs, this is a popular option.

Despite their limitations, text-based man readers should continue to be popular for some time to come because of their great portability. Although character-based terminals are antiques in the offices, there are still the mainstay over phone lines; and any windowing system must provide a UNIX command line, from which text-based readers are available. Also text-based man readers are most adaptable to new environments: For instance, in an `xterm`, one can copy text and paste it into another X application, a capability that `xterm` provides for free and that text-based man readers do not subvert with well meaning but unportable ornamentation.

Graphical: `xman` and `hman`

The graphical man reader bundled with the industry-standard X Window System is `xman` [SP88]. One may scroll forward and backward with scrollbars to read the text in a pleasing proportionally spaced font. From a listing of all names in a volume, one may point and click in place of typing in the man page name. And with mandesc files, short declarative files placed at the root of a man tree, one may control the mapping of directories in the file systems to section names. Unfortunately, a directory's mandesc file dictates the mapping for all readers of that directory (see "Improving xman and SGI with tkmandesc" on page 12). Alas, removed from the text-based UNIX pipe world, `xman` lacks copy and paste and searching within the man page. One must scroll page by page, reading the screen for the relevant passage, and then retype it into its destination.

Just as Perl `man` improved upon the earlier `man`, `hman` [Rao91] improves upon `xman`. The manual page of `hman` claims that it "displays a manual page and highlight[s] the references to other UNIX commands. Clicking on the highlighted text will display the new manual page. The Search button searches a regular expression in the text." Unfortunately we must trust the word of its man page as the latest and only version of this software known to `archie` (a program that searches all registered public FTP sites for software with a given name) has not been updated since 1991 and does not even compile with the current version of Motif. An e-mail message to its author received no response.

Cut off from the program composition afforded by UNIX pipes, graphical readers must themselves implement all the desired functionality or do without.

2.1.2 Next-generation Help Systems

"Next-generation help systems" refers to those online documentation systems that recognize the limitations of the manual page scheme, born as it was before windowing systems, and strike out into graphical user interfaces with multiple fonts, multiple colors, mixed media (text and embedded pictures at minimum), hierarchical structuring of information and hypertext links. Space constraints permit mention of only a select few.

TeXinfo format spans the divide between man pages and multimedia formats. It describes a structured, hierarchical format homologous to the essentials of LaTeX's format, with hypertext links between structural entities. Info viewers can build more advanced features on top of this infrastructure, but it remains a text-based format at heart.

XHyper [Hem93] and X.Desktop [Dav93] surpass TeXinfo viewers with multiple fonts and embedded pictures. Navigation in the hypertext web is enhanced by backtracking and bookmark setting. Of the two, XHyper is the more advanced, leveraging code from InterViews' `idraw` graphics editor and `doc` word processor. In addition, one enjoys the convenience and sophistication of FrameMaker in writing his help documents, as FrameMaker's Maker Markup Language is XHyper's storage format.

HyperHelp [Bri], a commercial product, represents the top of the line of this new class of help systems. It sports hierarchical document navigation with hypertext, backtracking and bookmarks. Help pages are rendered in multiple colors and illustrated with embedded pictures. One may attach PostIt-like notes to arbitrary help pages. It boasts a text search facility and a popup glossary and can print out any help page.

No matter how sophisticated, however, each of these new help systems requires its own specialized file format. Although there is a proposal for a X-based online help protocol [SV93], it is difficult to imagine a format as universal as the traditional manual page. And what is to become of the thousands upon thousands of existing man pages? Are these converted into the as-yet undecided standard next-generation format? What if no standard emerges?

2.2 Goals

Previous man page readers support the existing collection of manual pages but lack essential capabilities. Next-generation help systems have these capabilities but do not operate on manual pages effectively. The goal of TkMan, succinctly put, is to give the best of both worlds. It leverages aggressive heuristic parsing to bring as much of the functionality of the next generation as possible to existing man pages. TkMan marshals its efforts to (1) find the single man page from among the thousands and (2) to locate the relevant lines within that man page. TkMan exploits all pre-existing structural signposts to this goal and lets the user construct his own ad hoc guides. TkMan fleshes out this core functionality with a full set of supporting features. Finally, TkMan makes its domain expertise available for use by other programs.

The following sections consider in greater detail the problem domain, TkMan design, TkMan implementation, and the public reception of the software.

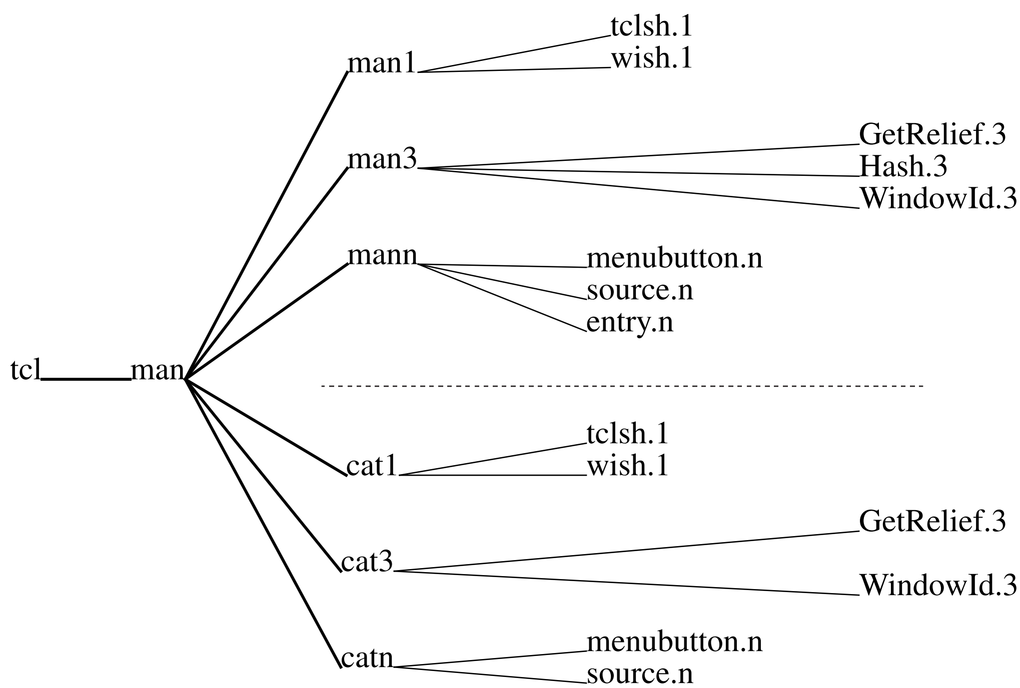
2.3 Raw Materials

This section takes a detailed look at the problem domain and defines terms used to refer to aspects of man pages in precise language. It also considers the tools used to build the tool, namely Tcl and Tk.

2.3.1 man pages

Man pages are online documentation files written in `[tn]roff` source form that more or less have a common internal organization. Man pages must be formatted by `nroff` for viewing on the screen or line printer or by `troff` for typesetters and laser printers. For online use this formatting time would be annoying and so "preformatted" versions of nroff-formatted man pages are cached on disk. It is not necessary to keep a cached version as every man reader will format one on demand from the source if a formatted version is not found. In a man page tree, formatted pages are kept in separate directories from source: formatted pages are stored in “cat” directories, source in “man”. Within the cat and man, man pages are further categorized by function, with a numbering scheme that assigns numbers to various functional groupings, e.g., 5 to file formats. This results in directories named `cat1`, `man1`, `cat2`, `man2` and so on.

The full set of directories and files comprises a man page *hierarchy* or *tree*. A possible man tree is shown in the diagram below. There are three functional groupings or *volumes*, namely



1, 2 and n. The directories `man1`, `man2` and `mann` hold `[tn]roff` source, and the corresponding cat directories hold formatted versions with the same file name. Observe that `Hash.3` and `entry.n` lack corresponding preformatted pages. The `tcl` at the top of the tree suggests that the man pages below all relate to Tcl and Tk. Other substantial software, especially that used for program development, may have man trees devoted to them. General system software is usually placed in `/usr/man` or `/usr/local/man`. With a `MANPATH` environment variable listing several directories, one may instruct man page readers to search in several man trees.

2.3.2 Tcl and Tk

Tcl is a string-based, general-purpose interpreted language influenced by C, UNIX shells, Lisp, and Awk, among other languages [Ous90]. Tk is a windowing toolkit for X Windows that augments Tcl with commands to control most of its functions [Ous91]. The Tcl interface to Tk hides many of the details that plague other X toolkits. In creating widget instances, one specifies as many options as relevant and Tk provides reasonable defaults for the rest. Tcl handlers for various Tk events can accomplish much of the work of an application with short scripts. Where Tcl builtin commands do not suffice, C functions can be added on an equal basis to builtins.

The Tcl interface to Tk resembles pseudocode one might write to outline an interface before coding it. For instance, to specify a button with a red background that reads 'Abort' and exits with error code 1 when invoked, one might scribble: "button: 'abort', background red, callback 'exit 1'". In Tcl/Tk the complete, executable code is "`button .b -text abort -background red -command {exit 1}`". Whereas many Xaw and Motif programmers spend much of their time looking up and reading function templates and carefully managing application contexts and color maps and similar inessential detail, the Tcl/Tk programmer can directly specify the desired result in plain language and Tk hides confounding details.

If it were not for the ease and power of Tcl/Tk, TkMan would not have been written. It grew out of a long-time frustration with `xman`, but the thought of writing a better man page browser with any existing toolkit (with the possible exception of InterViews) did not inspire one to take to the keyboard. To try to quantify the relative ease in programming in Tcl/Tk as compared to traditional C function-oriented toolkits, consider that the current version of TkMan runs 2700 lines of code, well under half that of `xman`; the original version of TkMan, which matched `xman` in features except for mandesc but compensated with searching, shortcuts, and history, ran 1400 lines—75% less code than `xman`.

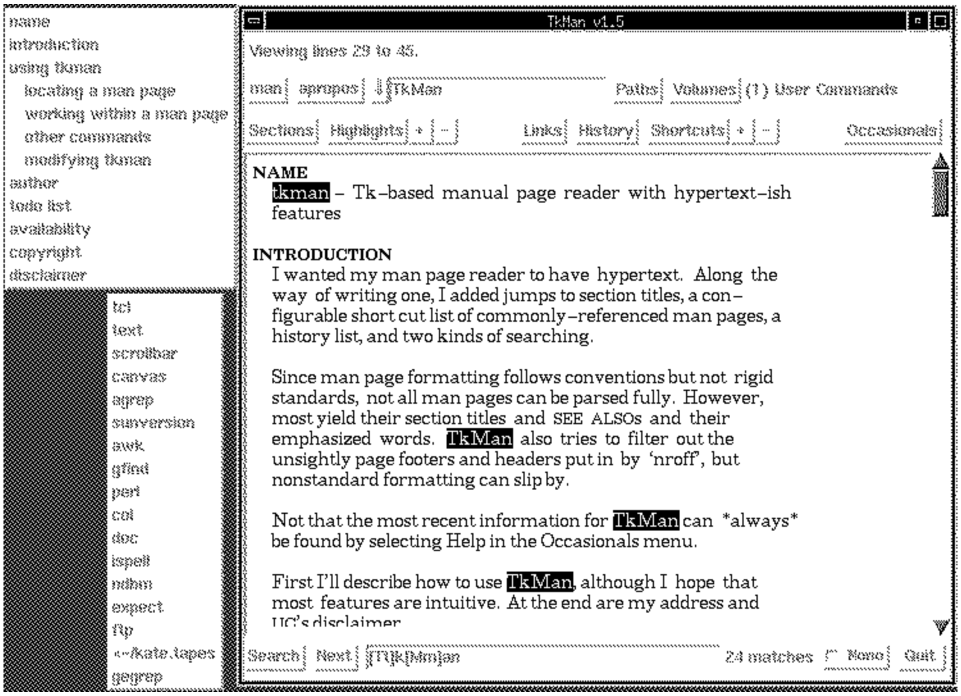
As a side benefit if using Tcl, the executable runs on top of a full interpreter. All of the advantages available to the Lisp programmer working in a development environment are availble to every Tcl executable. In particular, an application-specific scripting language or extension language comes for free—it is exactly the implementation language, Tcl.

2.4 Design

2.4.1 The "Visible Interface"

Typically, a graphical user interface to an application groups its commands in pulldown menus, which in turn are grouped in a menu bar; likewise, some applications group icons representing tools into a tool palette. In either case access to functionality is collected together into one homogenous region; one command is undifferentiated from another in the graphic design. A case in point, `xrn` creates tens of buttons and lets a row-column geometry manager place them one after another on the line, linebreaking the stream when necessary. This results in wide horizontal separation of buttons that are closely related logically, like the `next` and `prev` article buttons, which are separated by six intervening buttons, and a vertical organization of buttons determined by window size. Donald Norman's teaching to "make things visible" and "the principle of mapping" between controls and their effects [Nor88] applies just as strongly to a computer's graphical user interface as to everyday objects. Edward Tufte's principles and illustrations of information organization [Tuf83, Tuf90] are more relevant than any keystroke-level model for user performance time [CMN80]. The user interface elements of TkMan are placed in logical relation to each other so as to suggest relationships between functionality and to suggest the operation thereof. Indeed, the interrelationships of functionality dictate the organization, which in turn points back to the function. Form follows function. Form illuminates function.

The screen dump below displays TkMan's main window at right, with two torn-off pulldown menus at left. High-use functionality is given top-level screen real estate. For instance, the



man button, of obvious importance, is placed at the upper-left corner. Lesser-used functionality or those that would occupy considerable screen space are compressed into pulldown menus. For example, the iconify-on-startup switch and other miscellaneous commands are collected under the 'Occasionals' pulldown heading.

Once the relative importance of functionality is determined, those of comparable worth are arranged into functional groups. The first line of the screen is the status bar. Always visible, it reports a continuous stream of information, anything from the full path name of the displayed man page to the status of an incremental search. As some of the messages report error conditions, the status bar has been lifted above the rest of the graphical user interface so as not to be lost in the crowd.

The second line of the graphical design is devoted to identifying the single desired man page from among the thousands. The man and apropos buttons, which initiate manual page and keyword searches, respectively, announce that the adjacent text entry box is for typing in the man page name or search term. If multiple matches are found in a man page search—several unrelated man pages may have the same name—a downward arrow appears; its appearance suggests a pull-down menu and its position suggests a relation to the man page name, which turns out to be a list of alternatives with the same name. Past the apparatus for searching for man pages, the 'Paths' pulldown controls the directories searched. Lastly, 'Volumes' lists all man pages in the selected volume, from which one may choose the desired one.

The center portion of the third line is devoted to navigating other man pages by selection from various caches. 'Links' gleans the names of semantically related man pages from the current one's SEE ALSO section. Although the user may select these features by scrolling to the SEE ALSO section and using hypertext, it is often more convenient to be able to select from a pulldown menu of choices. 'History' lists the last fifteen or so man pages viewed; it is a temporal cache. 'Shortcuts' caches a user-selected list of man pages. The graphical placement of plus and minus buttons adjacent to ‘Shortcuts’ gives a strong visual cue as to how to add and remove names from the list. Although there is a second set of plus and minus buttons on the same line, the propinquity of each set to its related menu clearly states the intended associations.

The lefthand portion of this line and that far below are concerned with navigation within a single man page. ‘Sections’ and ‘Highlights’ both jump directly to interesting places in the document. ‘Sections’ takes advantage of sophisticated heuristic parsing that distinguishes section titles from page headers and footers in order to present a list of sections, a selection from which displays that section. Sections are structural units that the author of the man page thought important; one may voice his own opinion of interesting locations by highlighting a region of text as though with a yellow marker. A fragment from this region becomes its key in the ‘Highlights’ menu.

At screen bottom, just underneath the man page content, lies the regular expression search interface. The ‘Search’ button both labels the purpose of this region and initiates a search. ‘Next’ obviously searches forward for the next match (due to the technical infeasibility of searching backward for regular expressions, there is no ‘Prev’ button). As with the man name entry box, the position of the entry box here suggests its use in typing in the regular expression for which to search. Also related to the man page just above is the ‘Mono’ button, which toggles to a monospaced font that is needed to align columns in tables.

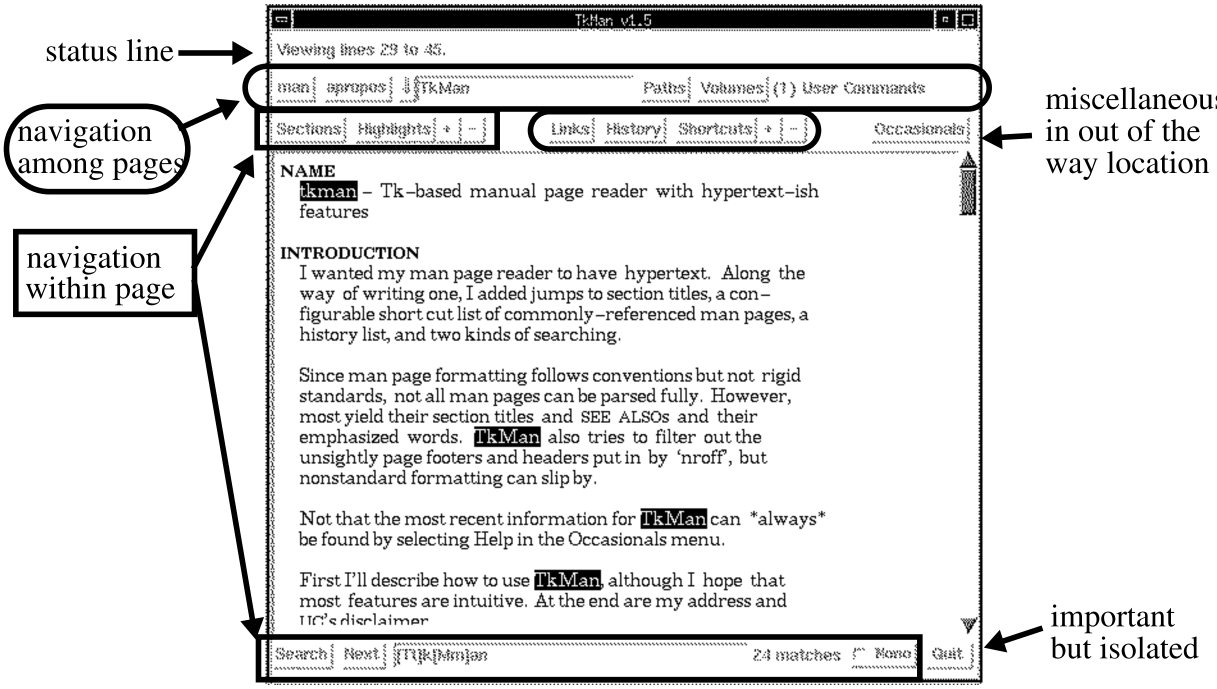
Incremental searching does not have a graphical interface. In truth, one long-time user of TkMan, apparently reluctant to read the help page, did not know it existed. The reason for the lack is that incremental searching is intended for quick, hands-on-keyboard searches, as say to identify the meaning of a mysterious command line option. Giving incremental searching its own real estate as for regular expression searching takes space away from showing the man page content. Given that regular expression searching demands screen space for entry of often-complex regular expressions, perhaps incremental searching could share space with regular expression searching, with a switch between the two.

‘Quit’, in the lower right of the screen, is used at most once during each run, obviously. It earns its conspicuous screen space in its role as a reminder to quit via the ‘Quit’ button in order to save the modified state from the run. Its placement in at the extreme far corner mitigates the chance of accidental use.

‘Occasionals’, a pulldown menu of commands seldom if ever used, gets the area otherwise unused in the interface, which happens to be the extreme right of the third line.

The diagram below illustrates the functional groupings described above.

One can imagine a more fluid interface yet. The widgets provided by Tk and most user interface toolkits come in rectangles of varying sizes that are pasted together in nonoverlapping regions. Given a more powerful rendering mechanism such as Display PostScript [Ado93], one could compose an interface of greater aesthetic harmony and take fuller advantage of graphic design with greater freedom in widget shape and relative placement. Some of this is possible with Tk’s canvas drawing widget, which can draw arbitrary shapes and associate widget-like behaviors with them. Unfortunately, this unorthodox implementation of the interface must dictate widget placement precisely; it loses the support of Tk’s geometry management for window resizes, although Tk supplies enough hooks to reimplement this on an ad hoc basis. Implementing a canvas-based interface for TkMan would be an interesting exercise, but such effort is more profitably expended for more complex applications. Kai’s Power Tools [HSC], a set of image processing fil-



ters for the Macintosh, uses the idea of custom user interface elements to great effect in providing an intuitive interface to controls that otherwise have no clear relationship to the standard Macintosh GUI [App87]. Furthermore, Robertson et al at Xerox PARC [RCM93] exploit 3-D rendering and animation in the creation of novel information browsers.

The other aspect of the "visible interface" is its dynamic reconfiguration to different document types. Four kinds of document are distinguished: manual page, TkMan's help page, raw ASCII text files, and apropos/volume listings. If a feature is not applicable to particular document type, it is grayed out. This immediately announces its unavailability, as opposed to an interface that allows the user to attempt the operation, only then to be told in an error message that the function is unavailable. Grayed out user interface elements also help distinguish between the case of features being unavailable and the case of features being available but without effect. For instance, often one will read a man page without any ad hoc highlighting. The user can look under 'Highlights' to find that there is none. Compare this with viewing a volume listing, during which time 'Highlighting' and its add and delete buttons are grayed out, indicating that highlighting is not possible.

2.4.2 Navigating among Man Pages

Once one has need to consult a manual page, the first order or business is to find the desired one from among the thousands. TkMan provides many aids to do this quickly and efficiently.

If the name of the man page is known, one simply types it in and presses the 'man' button. If the page's volume is also known, one may append this information to the name and expedite searching the database. If one knows only an approximate name or wishes to scan the database for certain patterns—all man pages with "mail" in their names, say—one may search with a pattern like those available in UNIX shells. Sometimes when searching for a name and often when searching with a pattern, multiple matches are found. Other man pages variously display the first one only with no mention of the existence of others, or force the reader to wade through the lot to find the one of interest. TkMan improves upon this by showing the most likely match and giving a pulldown menu listing all other matches by full path name, so that one may directly pick the man page of choice. Rather than suffer false positives gladly, one may elect to mask out certain directories of the `MANPATH` known not to be relevant. The 'Paths' menu gives piecewise control to include or exclude individual paths. Man page searches, as well as the apropos and volume listing commands described below, recognize the new `MANPATH`. Once a man page is on the screen, one may invoke a hypertext search for a man page name by a double clicking with the mouse over a reference in the current text.

For those occasions when one has no knowledge of the name of the desired man page or simply would like to pick from a list, there are apropos searches and volume listings. Apropos searches the one-line description lines from all man pages for the given keyword. All matches are listed to the screen, where one may be picked with the hypertext facility. One may further process the result of apropos through arbitrary pipes. To show only those commands that begin with the letter "g", for instance, append the pipe specification '` | grep ^g`' after the search key. 'Volumes' lists all and only those man pages in a single volume, from which, again, one may be chosen via hypertext. Application of tkmandesc commands rearranges the man page-to-volume mapping and can even create "pseudo volumes" not grounded in traditional volume names.

Whereas the functions described above search large portions of the database, three features present short lists of likely candidates. 'Links' takes its list from man pages that the current one lists as semantically related. This feature lets one find the general area of interest and branch from there to specifics. 'History' is a temporal cache of recently viewed man pages. If one is comparing man pages or constantly switching between two perhaps it would be more efficient to instantiate a new viewer, but for those cases when there is a small "working set" of pages, this function is handy. 'History' is better than the back command found in many hypertext systems because 'History' gives direct access to any of the last several pages, whereas with back one must step through recent history in strict reverse chronological order. Finally, the user may store his own selection of frequently consulted man pages on the 'Shortcuts' menu.

2.4.3 Inspecting Individual Man Pages

Once the correct man page is found, it remains to locate the information within the man page—a "page" that can run tens of printed pages. Here too TkMan provides assistance in several ways.

Traditionally, man pages are organized into the following sections: Name, Synopsis, Description, Options, Files, See Also, Diagnostics, Bugs. Longer man pages often add more sections and introduce subsections. Clearly, this structure is invaluable to identifying the relevant portion of a document. Whereas other man page viewers force the user to scroll screen by screen keeping a lookout for a section title of likely relevance, TkMan collects section and subsection titles into a pulldown list; selecting from the list scrolls directly to that section.

However, these author-imposed sections may not suit the individual's needs. Perhaps one section (say Options) is so long that the structural organization aids little. In fact, often the desired information is often a specific fact with little relation to structural groupings. As an illustration, consider Tk's arrowhead shape option, which uses a list of three elements in the following way:

> The first element of the list gives the distance along the line from the neck of the arrowhead to its tip. The second element gives the distance along the line from the trailing points of the arrowhead to the tip, and the third element gives the distance from the outside edge of the line to the trailing points.

Even if this option is used frequently, the ordering of these elements is likely to be difficult to recall exactly. With the 'Highlights' features one may mark such easily forgotten passage as if with a yellow highlighter (this can be changed to another color, underling, a font change or a 3-D effect). Not only is the text visually marked but the first thirty characters (also adjustable) of the passage become its identifier in a pulldown list that collects all highlighted passages from the man page. Highlights are not stored with the man page text itself—an important consideration as this text is often not writable, and it is likely to be shared among many users whose would not care to see everyone else's annotations—but rather are stored in the individual's startup file, along with all other customization. Highlights are tied to specific line and character positions in the line and rely on infrequent updates to man pages keep the annotations accurate over a reasonably lengthy lifespan. (One could imagine extending the matching algorithms found in `diff` to try to recover accurate highlight positions after small changes to the man page.)

To scan the entire document for the desired information, one may employ two kinds of searching, incremental and regular expression. Incremental searching is intended for quick, exact-match searching. It searches only as far as the next match. Regular expression, on the other hand, provides more powerful searching but takes slightly more effort to specify. Regular expression searching scans the entire document for matches, highlights them, and then lets the user navigate from one to the next. Switches under the 'Occasionals' menu toggle the cases sensitivity of the search mechanisms.

Finally, Emacs-like [Sta87] bindings control screen scrolling from the keyboard. One may scroll by lines or pages, jump directly to the head or tail of the document, or set a mark and return to that position later.

2.4.4 Customization

TkMan may be customized on three levels of increasing difficulty. On the most transparent level, TkMan records various aspects of a user's interaction and restores this state at next execution. One would expect that the short cuts and highlights are persistent, but TkMan also remembers the window size and position. Given color palettes, font browsers and other system support, it would be trivial to record these cosmetic changes transparently, thus relieving the user of wrestling with X resources. While the same executable runs, a re-examined man page automatically scrolls down to the last screen viewed; this information could easily be made transparently persistent, but—assuming a decay of temporal locality between process executions—this would cause more confusion than good.

All persistent information is stored in an individual's `.tkman` ("dot t-k-man") startup file. In contrast to X resources which provide complete access to public widget properties at the cost of learning how to navigate general widget hierarchies and the application's in particular, TkMan stores all "interesting" parameters as suggestively-named variables, which can be edited in the obvious way. In most cases the widget hierarchy only obscures the path to the desired behavior modification. Installation with TkMan's method is also easier because many X applications require a baseline set of application defaults be in place, whereas TkMan creates its own if none exists. Customizing TkMan as this level does not require knowledge of the Tcl language as long as one can recognize and follow model forms. For instance, '`set man(aproposfilter) {| sort | uniq}`' defines the default filter for apropos. One needs to be UNIX literate enough to recognize that, semantically, this is a pipe ('|') through two UNIX commands, and that, syntactically, the pieces are surrounded by whitespace. With just a little knowledge and common sense, one can modify all the aspects of TkMan that its author has deemed interesting.

Those not satisfied with restricted access can always modify the source code, of course. But since Tcl is interpreted, one can dynamically replace any component of TkMan, including procedures, just as in Lisp—and without modifying a common source. This is made possible by the structure of the startup file. The head of the file consists of those "interesting" variables described above; it is managed by TkMan internals, which will discard any unrecognized invading code. The tail of the file, however, is maintained verbatim; it is here that one may place arbitrary code to execute at startup time. Since this code, what is called extension language code in other systems, is Tcl code identical to that which implements TkMan, the customizer commands great power. To see how such power can be useful, consider IBM's AIX operating system. Since IBM has chosen to disregard all tradition in man page organization, storing them its own unique format, AIX users are currently denied the benefits of TkMan. A Tcl programmer may decide to let AIX's `man` command retrieve the text from storage and then hand off to TkMan at that point. In his startup file he could replace the procedure `manShowMan`, which accepts the name of the man page to read, with a version that spawns a `man` process with the name as its argument. The barrier to exploiting this level of customization is that one must learn the Tcl language, obviously.

2.4.5 Improving xman and SGI with tkmandesc

One may have a number of software systems substantial enough to have their own set of man pages. X Windows, for instance, lists approximately 1000 "System Calls" and many more "User Commands", and Tcl/Tk has almost 200 manual pages in various volume categories. As described in subsection "man pages" on page 4, man pages are grouped into a number of volumes, typically eleven or twelve, with such titles as "User Commands" and "System Calls". All manual page names are combined together in a global namespace subdivided only by volume. With 5000 or 15,000 or more entries, it becomes less and less manageable to pick out the name from a list of all entries in a volume. Moreover, it becomes more and more likely that names will conflict, especially considering the UNIX predilection for short, generic names. Although TkMan gives one a menu of all duplicate names so that the correct one may selected, it would be more convenient to be able to specify the desired page directly. Finally, it would be convenient to view a listing of all documentation related to a particular software system, say TeX and LaTeX, without regard to volume categorizations.

A partial solution to these problems is provided by `xman`. By writing a *mandesc file* for the man directory hierarchy for a particular software system, one may specify, on a volume-by-volume basis, the mapping from "physical" volume to `xman` volume. For instance, sections 2 and 3, usually called System Calls and Subroutines, may be combined together and called "Programmer Subroutines”. A number of repetitive lines of specification can remap all volumes in a TeX/-LaTeX man hierarchy into a new volume called “TeX/LaTeX”. SGI’s IRIX operating system extends `xman` by giving every subdirectory found in the manual page hierarchy its own section: make a subdirectory in man/man3 called “TeX-LaTeX” and all filenames in that directory appear in a new TeX-LaTeX menu entry.

This solution does not come without a price, however. The principal disadvantage is the lack of customization by the individual user. Although one may specify which hierarchies are to be searched with the `MANPATH` environment variable, the groupings within every hierarchy is specified by its single mandesc file, which must be observed. This may be acceptable to the isolated user, but the situation is less tolerable to anyone sharing man pages with others. Given that man pages are (to a first approximation) simply text, they can even be shared across machine architectures. Since customization for one dictates “customization” for all, the system administrator may simply decided not to customize at all. SGI’s extensions further this loss of control. By placing directories for specific systems in the core hierarchy, it forces one to include these entries, stripping one of the ability to decide this with `MANPATH` control.

With a mandesc file it is possible to create directories that are not searched by the (ASCII-based) `man` command or by `catman`, which creates the indexes for apropos. Moreover, any man pages installed via SGI’s extension are not available to ordinary man readers. SGI ships man readers suitably modified for IRIX, but this excludes sharing these man pages across operating systems in a heterogeneous environment. If for no other reason, `man` remains popular for copying and pasting text with the X selection, a function that `xman` does not support, and therefore its support should remain an active concern.

Modeled on `xman`’s mandesc, TkMan’s tkmandesc commands solve all of these problems. Given the coarse-grained volume groupings by default, one may employ a concise, pattern-matching command language to remap physical volumes to volume names in TkMan. This remapping is expressed in the individual’s own `.tkman` startup file so that his preferences do not affect those of others.

By default TkMan creates the cross product of the directories in the `MANPATH` environment variable with the common volume titles to create a complete matrix of directory names to be searched. For example, given a `MANPATH` of `/usr/man` and `/usr/local/man`, volume 1 is set to search directories `/usr/man/man1` and `/usr/local/man/man1`, and volumes 2 3 4 5 6 7 8 o l n are constructed similarly. tkmandesc commands manipulate this matrix by adding pseudo volumes, adding or deleting directories to or from a volume, and moving or copying directories between volumes. Any command may use UNIX shell-style glob patterns to reduce redundancy.

Whereas `xman`’s mandesc is tied to a particular hierarchy, tkmandesc patterns span them, so that a single command (namely, `manDescDelete * {*[2348]}`) can, for instance, collect all man pages from every hierarchy (*) in the System Calls, Subroutines, Devices and System Administration volumes ({*[2348]}) and remove them from the database of names. This particular command would be useful for a UNIX novice who wishes to banish all programming-related documentation.

No tkmandesc commands affect `man` in any way. (Although TkMan does support cut and paste which mitigates the need for `man`.)

Finally, this remapping is recognized by the name search engine so that one may directly specify, say, Tcl's `gets` function (with the search specification `gets.t`), bypassing the one in the C standard I/O library.

Whereas one may create volume listings specific to a particular software system in `xman`, it takes one line per volume—potentially eleven or twelve nearly identical lines. With tkmandesc patterns, this is a line or two. For instance, to create a volume for Tcl/Tk man pages and remap all man pages in the `/usr/sww/tcl-7.0/man` directory to this new volume, one merely invokes `manDescAddSects {{"t Tcl/Tk"}}` followed by `manDescMove * t *tcl-7.0*`.

2.5 Implementation Notes

2.5.1 Filtering with RosettaMan

Once the name of a man page is identified, TkMan obtains its content from the same location as the `man` command: from a cache of preformatted files or, failing that, from its `[tn]roff` source, which is dynamically formatted with `nroff`. `nroff` produces text meant to be read on character-based terminal screens or printed to line printers. It exploits the limited features of these output media to simulate some text formatting. For example, `nroff` emits *<character>-backspace-<character>* to overstrike a character with itself and obtain boldfacing, and *<character>-backspace-underscore* to obtain underlining, a poor man's italics. Unfortunately, this is more burden than blessing for modern computers with bitmapped displays that can compose multiple fonts.

Further complicating the nroff-to-TkMan conversion process is the great variety of formatting macros found among the flavors of UNIX. What could possibly be so deviant about the simple man page? The following table gives a taste of the significant variations to be suffered.

OS	nonstandard feature(s)
SunOS	best behaved
Sun Solaris	bold italics (!)
DEC Ultrix	agressively interspersed tabs, lower case in section headers
SGI IRIX	packed (i.e., compressed), two-line headers and footers
AT&T System V	global indentation
H-P HP-UX	compressed files—directories (but not files) marked as compressed
IBM AIX (not supported)	complete set stored as single file, not in man/cat hierarchy; section headers indistinguishable from body text; no bold face or italics; no `[tn]roff` source

The obvious question is, Why not use a custom set of formatting macros and always reformat directly into TkMan format? Alas, the original and sole reason for the preformatted man page is that it takes too long to format anew for every request and this is still the case even with today's hardware, orders of magnitude faster though it is. In addition, some flavors of UNIX are shipped without man page source; they only have preformatted pages.

The result of much empirical research, TkMan's companion man page filter, RosettaMan, reads this babel, all dialects simultaneously, and writes an international Esperanto. It is especially important to handle multiple man page styles at sites with heterogeneous machinery, as man pages do not depend on any machine's instruction set and thus are supremely sharable. On the output side, RosettaMan takes advantage of centralized analysis services to talk to not only TkMan, but a number of output formats including Ensemble and HTML.

It is the unprecedented sophistication of RosettaMan that is the key to much of TkMan's functionality. It is critical to separate out the boldfacing and underlining to permit searching. We need high quality discrimination of page headers and footers from section titles in order to offer direct jumps to sections via a pulldown menu. Out of the list of sections we need to comb the SEE ALSO section in search of references to other man pages for the Links menu.

Typographical Filtering

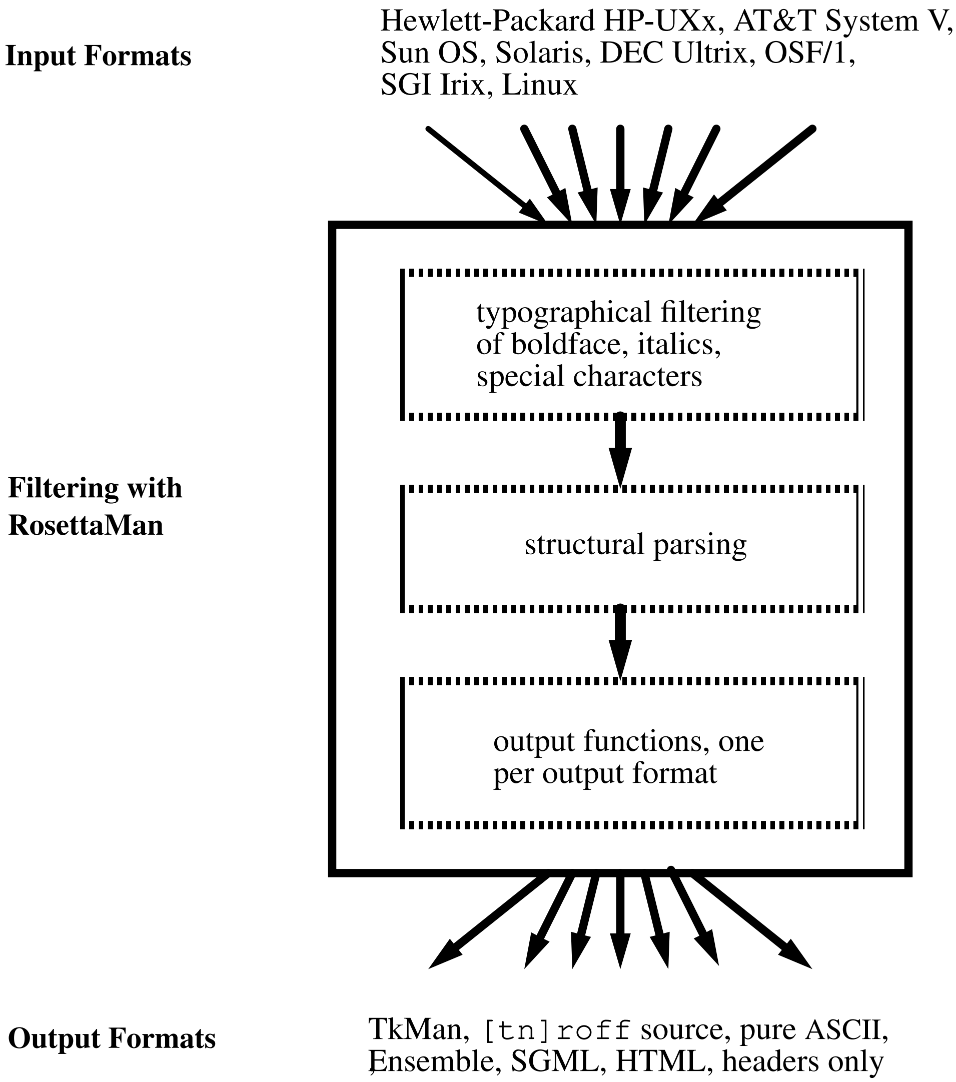
As depicted in the diagram below, the filtering done by RosettaMan is a three-stage process. First a character-level formatting stage separates out boldface and italics ranges, changebars, and indentation, leaving pure ASCII. The structural analysis phases identifies headers, footers, section heads, itemized lists and more, calling one of a number of output functions with a corresponding signal. An output function centralizes all knowledge of a specific output format, of which there are currently seven.

Knowledge of man page varieties is localized in the typographical filtering stage. It converts tabs to spaces. It knows that sometimes underlining begins *<character>-backspace-underscore* and at other times *underscore-backspace-<character>*. It knows that '`o`'*-backspace-*'`+`' (or '`+`'*-backspace-*'`o`') yields a bullet.

Once boldfacing and underlining information is separated from the text, it can be reapplied in a form more appropriate to the output format. And there is no reason to map boldface to boldface, underlining to underlining; one may choose more aesthetically pleasing presentations. Typically underlining is replaced by italics (which is unavailable on line printers), but one may also choose to remap the bold italics of Solaris to simple bold. It is more pleasant to read man pages in a proportionally-spaced font, but this misaligns tables, so a monospaced font is kept available at the click of a button. In a proportionally-spaced font, strings of capital letters (as in 'MANPATH') tend to overwhelm the line, so these are presented in small caps ('`MANPATH`'), which is to say the capital letters in a smaller font.

Structural Parsing

Structural analysis attempts to identify the extent of various structural units, including page headers and footers, section and subsection titles, and itemized lists. Operating from the formatted man page, there is no entirely accurate method to achieve this. Both page headers and



footers and section titles start in the first column and may have any number of words of mixed case.

Certain heuristics, however, have proven remarkably effective. Page headers and footers, for example, typically repeat elements from one to another, so one can check all lines that begin in column one (which are potentially either a section title or a page header or footer) against portions of headers and footers seen before. Often the page number is included in the footer, and sometimes the macros format odd and even pages differently, always placing the date near the binding and the page number toward the outer edge, say. This variation is effectively strained out by grabbing the first "phrase" (all characters up to a lengthy space) from the first header and footer and looking for a match at *both* the start and end of the line in question. Effectively distinguishing pages headers and footers from section titles is critical to providing noise-free list of sections to jump to and to identify the SEE ALSO section from which are gleaned references to other man pages.

All structural parsing is centralized in this phase. It sends signals to whatever output function is active notifying it of interesting structural units. This phase also provides common services for converting multiple lines into paragraphs and compressing the multiple spaces used in fill justification into single spaces. By centralizing structural parsing, any heuristic analysis algorithms are immediately made available to all output formats, which can either act on this information or simply let it be handled by the default case.

Output Functions

Given all the analysis already done, it is a simple matter generate output suitable for a variety of uses. By little more than introducing section titles with '`.SS`' and surrounding italics passages with '`\fI`' and '`\fR`', one writes a reverse compiler which generates `[tn]roff` source. This is useful for obtaining high-quality printouts on laser printers in those occasions when only formatted version is available, as is the case with some vendors' flavors of UNIX. Moreover, one can write man page in ASCII without bothering with macros, then reverse compile and work from there.

A number of output formats have been implemented besides those for TkMan and `[tn]roff`. The ASCII format produces pure ASCII shorn of boldfacing and italics for use by indexing programs. With all of the structural signals that output functions receive, it is trivial to write SGML [Gol90] and HTML (which is simply SGML with a specific document type definition) output formats simply by emitting an appropriate start tag or end tag when called with the corresponding signal. The most highly developed output form is that for Ensemble [GHM92]. With its companion structure and presentation schemas, the appearance of a man page in Ensemble can be radically transformed. Once Ensemble gives access to elision from its user interface, one will be able to bring up a man page in collapsed form, only the section titles showing, and view selected components as though one were using an outline editor.

The set of output formats is very extensible. One merely adds another C function that responds to whatever structural signals it is interested in. Output functions implemented thus far average 75 lines of C code in length.

2.5.2 Database & Searching

At startup TkMan constructs a database of all man pages available to it. This database serves two purposes. First, it is a cache for locating a man page name within a directory. Reading directories every time to find the requested name results in a noticeable wait, especially if some directories are mounted across a network. The user may restrict searching to a single volume (e.g., only search volume 1, User Commands) to speed searches, but often *all* man page names are searched for a match, as the UNIX tradition of using terse, English-like names leads to multiple commands with the same name. '`tar`', for instance, is both a command and a file format, and a command named '`format`' is used by system administrators to format disks and by Tcl programmers to emit a formatted string.

The second purpose of the database is to obtain different organizations of man pages from that given by their storage in the file system. The file system groups man pages by software package and within that by functional class. With the ordinary man page reader one may restrict searches to specific functional groups or man page directories, but this is tedious to type out and remember, and the organization is not customizable. Using tkmandesc commands, the user may add, delete, copy or move any of a software package's functional classes, and even add ad hoc "pseudovolumes", i.e., volumes not found in the file system.

The database is built by first establishing the mapping of volumes to directories of man pages and then fleshing out this structure by reading the man page names from disk. Lists of man pages are keyed by their directory path, a fact which is exploited when a user registers temporary disinterest in a software package's man pages, as the searches may disregard entire hierarchies of man pages with a single check. By default, TkMan behaves as most other man page readers, taking the simple cross product of `MANPATH` directory by volume names. In preparation for rearrangement by tkmandesc commands, each volume name has a list of full directory names. In the example default matrix shown below, the `MANPATH` consists of the directories `/usr/phelps/man` (abbreviated "(a)"), `/usr/local/man` (b), `/usr/sww/man` (c), `/usr/sww/X11/man` (d), and `/usr/sww/tcl/man` (e).

Volume name	Constituent directories
User Commands (1)	(a)/1, (b)/1, (c)/1, (d)/1, (e)/1
System Commands (2)	(a)/2, (b)/2, (c)/2, (d)/2, (e)/2
Subroutines (3)	(a)/3, (b)/3, (c)/3, (d)/3, (e)/3
Devices (4)	(a)/4, (b)/4, (c)/4, (d)/4, (e)/4
File Formats (5)	(a)/5, (b)/5, (c)/5, (d)/5, (e)/5
Games (6)	(a)/6, (b)/6, (c)/6, (d)/6, (e)/7
Miscellaneous (7)	(a)/7, (b)/7, (c)/7, (d)/7, (e)/7
System Administration (8)	(a)/8, (b)/8, (c)/8, (d)/8, (e)/8
Old (o)	(a)/o, (b)/o, (c)/o, (d)/o, (e)/o
Local (l)	(a)/l, (b)/l, (c)/l, (d)/l, (e)/l
New (n)	(a)/n, (b)/n, (c)/n, (d)/n, (e)/n

tkmandesc commands give full control over this volume-to-directories mapping, even to the point of creating new volumes (pseudovolumes) and adding directories not found in the `MANPATH`. To see how this such control can be useful, consider a Tcl/Tk programmer. He does not care about X11 subroutines since he can accomplish the same more easily with Tk; however, he is still interested in X-based programs, as not all of them have been reimplemented in Tcl/Tk. Moreover, he would like to group all Tcl/Tk-related man pages into their own volume so he can pick from a complete menu in a volume listing.

Unlike `xman`, which imposes any customization on all users of the same set of man pages, TkMan allows this Tcl/Tk programmer to customize it to suit his tastes perfectly without interfering with any other users. First he creates two new volumes, one for X commands and another for Tcl/Tk commands, and—knowing that the order of the volumes list is reflected in the ordering in the volumes pulldown menu—carefully inserts them in the volumes list. These new volumes are empty, of course, as they do not correspond to the file system in any way.

User Commands (1)	(a)/1, (b)/1, (c)/1, (d)/1, (e)/1
X11 Commands (x)	
System Commands (2)	(a)/2, (b)/2, (c)/2, (d)/2, (e)/2
Subroutines (3)	(a)/3, (b)/3, (c)/3, (d)/3, (e)/3
Devices (4)	(a)/4, (b)/4, (c)/4, (d)/4, (e)/4
File Formats (5)	(a)/5, (b)/5, (c)/5, (d)/5, (e)/5
Games (6)	(a)/6, (b)/6, (c)/6, (d)/6, (e)/7

Miscellaneous (7)	(a)/7, (b)/7, (c)/7, (d)/7, (e)/7
System Administration (8)	(a)/8, (b)/8, (c)/8, (d)/8, (e)/8
Old (o)	(a)/o, (b)/o, (c)/o, (d)/o, (e)/o
Local (l)	(a)/l, (b)/l, (c)/l, (d)/l, (e)/l
New (n)	(a)/n, (b)/n, (c)/n, (d)/n, (e)/n
Tcl (t)	

Various tkmandesc commands extract X11 commands from other sections and place them in the X11-specific volume (he wants games to show in the new X11 and the old Games sections, so this is copied); X11 subroutines and other technical information (volumes 2, 3, 4, 5, 7, 8) are removed from the database. All man pages related to Tcl are moved into that volume. Lastly, a directory that does not follow the convention of ending in "man" with subdirectories "man*n*" and "cat*n*" is added to the Local. The final database configuration is shown below.

User Commands (1)	(a)/1, (b)/1, (c)/1, (e)/1
X11 Commands (x)	(d)/1, (d)/6, (d)/o, (d)/l, (d)/n
System Commands (2)	(a)/2, (b)/2, (c)/2, (e)/2
Subroutines (3)	(a)/3, (b)/3, (c)/3, (e)/3
Devices (4)	(a)/4, (b)/4, (c)/4, (e)/4
File Formats (5)	(a)/5, (b)/5, (c)/5, (e)/5
Games (6)	(a)/6, (b)/6, (c)/6, (d)/6, (e)/6
Miscellaneous (7)	(a)/7, (b)/7, (c)/7, (e)/7
System Administration (8)	(a)/8, (b)/8, (c)/8, (e)/8
Old (o)	(a)/o, (b)/o, (c)/o, (e)/o
Local (l)	(a)/l, (b)/l, (c)/l, (e)/l, /usr/local
New (n)	(a)/n, (b)/n, (c)/n, (e)/n
Tcl (t)	(e)/1, (e)/2, (e)/3, (e)/4, (e)/5, (e)/6, (e)/7, (e)/8, (e)/o, (e)/l, (e)/n

The same pattern matching found in UNIX shells is employed here, resulting in a conciseness of expression, as commands can operate on multiple directories at once. The entire transformation just described is a case in point; the transformation from default matrix to custom organization requires less than 10 lines of tkmandesc code.

The capabilities made possible by the main memory database come at a price, however. A typical set of man pages numbers in the thousands or more, and it takes time to read the names at startup (especially if reading over a network) and main memory space to store them (paging is counterproductive as many searches touch all names). Accommodating 5000 man pages takes 15 seconds to read in and 40 KB to store. It is expected that one brings up TkMan once and uses it for the entire programming session, thus amortizing the time cost. Perl man chooses a different tradeoff, storing its (shared) database on disk, choosing to trade disk space for speed and correctness. Perl man needs to rebuild its databases after man pages are updated, though, whereas TkMan guarantees the correctness of its database as of startup time and if the TkMan's database is known to become out of date, it can be refreshed without re-executing the binary.

2.5.3 Persistency

The user of TkMan may make several customizations during its operation: changing window size and position, adding man page names to his 'Shortcuts' list, adding highlights to individual man pages, and toggling the various flags under 'Occasionals'. Following UNIX convention, TkMan saves the result of all this effort in a file named `.tkman` in the user's home directory.

Nothing is stored in any man page or man page tree as, in the first place, these directories may be mounted read only, and, in the second, these directories are already large enough without the addition of save files.

TkMan is comprised of several modules, each of which is responsible for saving whatever state it is interested in. In order to simulate in Tcl the separate name spaces of object-oriented languages, each module names all of its procedures and variables with a unique prefix: `man` for TkMan, `sb` for searchbox, `high` for highlights. If a module has any state to save it defines a procedure named <*module prefix*>`SaveConfig`; Tcl has the ability to query for procedure names, which makes shutdown simply a matter of calling all procedure names matching the pattern `*SaveConfig`. Typically each module stores all the variables that are to be persistent into a single array. Thus adding a new persistent variable is simply a matter of naming it to be part of the array, and saving persistent variables is simply a matter of iterating through this array.

The save file is stored as Tcl source code, which has the advantages of being human readable and being trivial to load back into memory with a single Tcl command. Most interesting features of TkMan are configurable simply by editing the appropriate line in the save file in the obvious way. For example, to change the means of marking highlighted regions from the default yellow marker simulation, one changes '`set man(show,highlight) {-background #ffd8ffffb332}`' to '... `{-underline yes}`' for underlining, '... `{-relief raised}`' for a 3-D effect, et cetera using any and any combination of the options supported by Tk's text widget. TkMan supports paging with the Emacs keybindings C-v and M-v, but by mimicing the lines '`set sb(key,C-v) pagedown`' and '`set sb(key,M-v) pageup`' lines one may add vi's keybindings with the lines '`set sb(key,C-b) pageup`' and '`set sb(key,C-f) pagedown`'. Other X applications use X resources for customization, but these unfortunately are tied to the widget hierarchy tree—a hierarchy which is often irrelevant to the desired change. Instead of trying to determine which branch of the tree to specify, users of TkMan identify a suggestively-named variable from the complete list given in the save file.

Another benefit of the fact that the save file is stored as Tcl source is that all Tcl commands are available, not just simple variable setting. This permits much more substantial customizations without modifying a shared executable. One can invoke tkmandesc commands, take advantage of iteration constructs, execute other binaries, even replace procedures! With the design decision to store the save file as Tcl source code, we get an extension language for free, one which is identical with and therefore just as powerful as the implementation language.

2.6 User Testing

2.6.1 Procedure

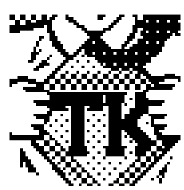
The first public version of TkMan was posted to the newsgroup `comp.lang.tcl` on April 1, 1993, initiating a lengthy informal reviewing process as users reported problems due to UNIX variations, pointed out that some features were not obvious in their use, and suggested new features. After many months, all variations of all major flavors of UNIX had been accommodated. TkMan was submitted for anonymous peer review via the `comp.sources.testers` newsgroup, where it appeared on September 28. This newsgroup posts an announcement of software available for testing purposes, those interested send to the mail server for the sources, and some smaller number of those submit reviews. Appendix A is the set of guidelines for reviewers.

Historically a low-volume newsgroup, `comp.sources.testers` had received at most 30 source requests for any given piece of software, of which an unknown percentage were followed up with reviews. Working to limit the number of reviews of TkMan was the fact that it required one to install Tcl and Tk, in fact the very latest, days-old version. Nonetheless, by the end of the two week review period, 143 people had requested the source and a remarkably high number of those (78) submitted reviews—a far greater number than the author could have collected without using resources of the Internet. Some of this popularity can be attributed to the fact that TkMan had already built a loyal following among readers of the newsgroup `comp.lang.tcl` and that the author personally solicited reviews from 75 known users of TkMan.

2.6.2 Reception

In three weeks of general release, over 600 people have FTP'ed the source from its home site at Berkeley and unknown numbers have taken it from the main Tcl/Tk archive (`harbor.ecn.purdue.edu`) and from distributed sites in Europe and Japan. TkMan is widely used in Internet-worked countries (Sweden, France, U.K., Australia, Canada, Germany, Japan), universities major (U.C. Berkeley, University of Illinois at Urbana-Champaign, M.I.T., CMU, Cal Tech, many in Canada) and minor (Stanford), and research laboratories (AT&T Bell Labs, Xerox, Sun). Feedback mail indicates that many source recipients are system administrators, making for a much larger user population. This statistical success is confirmed by the author's personal experience. His demonstration at Berkeley Tcl/Tk users group meeting was standing room only, and TkMan was mentioned in the User Interfaces class at Berkeley as an example of a user interface whose use is obvious even though its is more powerful than other manual page browsers.

Another indication of a program's popularity is the degree to which others build on it. Emacs boasts scores of elisp packages, and TeX and LaTeX have style sheets for a wide variety of document types and even a users group. On a more modest scale, someone has linked the Help key on keyboard to look up selection in TkMan. tkinfo [Whi93] incorporates TkMan's intra-page searching engine, and a future version will send man page requests to TkMan. Finally, several of the more artistically inclined have submitted icons, of which the best is reproduced below [Shi93], one which nicely picks up on the double joke "A bird? A place? TkMan! (TkPerson?)".



2.6.3 Problems

From the 78 reviews emerged a number of commonly cited problems. Foremost among these were (1) slow startup speed and (2) large memory image. In comparing startup times for TkMan on both the older Tcl 6.7/Tk 3.2 and the new Tcl 7.0/Tk 3.3 beta it was discovered that TkMan started up considerably slower with the latter. This prompted a bug report to Tcl/Tk's author, who fixed the bug in the release version of Tcl 7.0/Tk 3.3. One reviewer reported that his startup time shrank from over two minutes to under 40 seconds. Still, 40 seconds or the 15 that the author experiences is long enough for one to panic if there were no visible signs of life, which is the reason for the status announcement of each database volume as it is read in, information which serves the same function as the Macintosh progress "thermometer". In an attempt to expedite startup, an experiment in caching the database to disk between executions was conducted. Its failure (and the fact that `xman`, which builds a like database, starts up in a few seconds) speaks to the fact that 80% of startup time was spend not in communication with the file system but in Tcl string operations, a situation with no evident relief. This makes TkMan unsuitable for one-shot use, a situations most users cope with by executing TkMan at login time and leaving it running always. The string manipulation overhead also noticeably taxes the load time of lengthy man pages. Some relief is obtained here by very quickly showing the first screenful of the man page, which often answers the question, while waiting for the remaining lines.

Memory use is the other major complaint. The Tcl/Tk libraries compile to over 600 K bytes on a SparcStation and over 1.2 MB on a DEC Alpha, the TkMan script runs almost 100 K, and the name database contributes a total memory image of approximately 1 MB on a Sparc and 3 MB on an Alpha. Compiling Tcl and Tk as shared libraries, which may or may not be possible at present, might mitigate this situation.

In fact, both of these problems could be eliminated by caching the database on disk and accessing it in though an intermediary written in a language more efficient than the interpreted Tcl. This solutions has its own tradeoffs, however, namely that it requires disk space, but this is cheap, especially if the database can be shared. More significantly, it raises the problem of keeping the database up to date as any man page changed anywhere invalidates the cache.

Other dislikes were minor. The remainder of this subsection outlines dislikes voiced by multiple reviewers and the action take for each, if any.

Some accustomed to `xman` wanted volume listings set into columns, as opposed to setting the names as ordinary spaced text. Technically, Tk's weak support of tabs make it impossible to align proportionally-spaced text into columns, and a multiple listbox implementation ran sluggishly, with one full column scrolling then the next and so on in a ripple effect.

Some wanted to interactively edit `MANPATH`s or `.tkman` and have these changes take effect immediately, without quitting and restarting. However, the startup sequence is so delicate in its setting of variables and reading of the database (especially when tkmandesc commands rearrange default mappings) that this is not a trivial matter. One must quit and wait the 15 or 40 seconds to restart.

The tkmandesc commands follow the Tcl model of just adding key primitives to the language, using Tcl itself for common language constructs like iteration, procedures and argument passing, and data structures (especially lists). As TkMan has grown popular far beyond its original Tcl/Tk savvy user population, knowledge of the Tcl language has dropped precipitously. Thus the call for more examples of how to use tkmandesc commands and how to customize TkMan generally with other Tk commands like `option` and `bind`.

Ordinarily page headers and footers are artifacts from hardcopy-oriented formatting, lacking any information useful in reading them on screen. As one reviewer pointed out, however, the program version or last revision date contained within is occasionally invaluable. This observation spurred the addition of an option to display the (canonical) header and footer at the end of the page, with an entry in 'Sections' for direct access.

The attempt to support multiple paradigms of user interaction led to conflicts. Specifically, most X Windows applications bind mouse button 2 to pasting text into a text widget, whereas Tk binds it to "scanning" (i.e., scrolling without the use of scrollbars) in the same context. Cleverly, TkMan extends Tk to make button 2 serve two masters. When used for scanning, button 2 is pressed somewhere in the text, held down while moving the mouse to trigger moves several times greater in the text, then released. When used for pasting, button 2 is given a quick click. By measuring the time between button press and release and measuring the extent of the mouse motion, we can distinguish between the two: short clicks with little or no mouse movement implies pasting, longer clicks and/or greater mouse movement implies scanning. Considering both time and distance guards against cases of fast scans or scans that scan back to the same location. The same technique multiplexes button 1 between highlighting text and marking the target of a potential hypertext search.

Some long-time users obtained the new sources, typed '`make`' and received an error when executing TkMan. They were picking up their old version of `bs2tk` (the previous name for the companion man page filter), which did not support certain new options. Although the instructions read "type '`make install`'", this is such an easy error to make that now '`make`' alone writes out a message instructing the user of the proper installation command.

Highlights on text may be incrementally adjusted; the addition or deletion at the ends updates the highlights. A number of reviewers expected more rigid highlights, which must created precisely and thereafter live as an immutable unit. Thus to remove a highlight, they expected to click anywhere within it and click the delete button, whereas one must sweep out an overlapping region and remove the highlight(s) within. Since that style of indicating highlights has no obvious advantages over the current method, the current operation of highlights was retained.

Some users wanted a non-ANSI version of `bs2tk`, arguing that since Tcl/Tk does compile under old-style C, so should TkMan. A focus on research gives very low priority to such a port.

As mentioned earlier, tables in man pages are formatted with the assumption of a mono-spaced font, and the button 'Mono' changes the display's usual proportionally-spaced font to a mono-spaced one for this occasion. In the graphical design, this button is placed in equal importance as the regular expression search facilities, suggesting that it is not a one-time configuration option (these are found in the Makefile or under the Occasionals pulldown menu) but rather a frequently used function. A full description of the 'Mono' button is found in the help page. If nothing else, one can click on it and immediately see its obvious effect. Despite this, multiple people have mistaken 'Mono' to mean monochrome! In the release version the font of this button was changed to monospace, in contrast to the proportionally spaced font employed everywhere else in the GUI. No further confusion on this point has been reported.

2.7 Future Work

At nearly eight months old, TkMan is mature, robust, portable, full featured.

Nonetheless, two additional features are possible, given sufficient support from other software. In the first case, it would be a straightforward matter of providing alternative implementations of a few procedures for TkMan to use a database server, thus speeding startup and mitigating memory use. In the second case, if the next version of Tk supports "hidden" text, it could be exploited with an ability to view man pages as outlines, with sections expanded and collapsed.

In addition, there are plans to integrate TkMan with other tools. The Ensemble multimedia editor could use the features of TkMan with its formatting and display engine and other extensions to make for "active documentation" of programs.

2.8 Acknowledgements

Foremost I must thank Paul Raines. Three days after I first posted TkMan he had ported it to run on SGI's nonstandard man tree, and when a later version of TkMan featured a means of supporting the SGI without modifying the source code, it was Paul who wrote the tkmandesc extension file for SGIs. Paul was gracious enough to let me impose on him to try out new features. I knew that if a Smart Guy couldn't figure it out, I needed to invest more effort.

I also acknowledge the legions from the net who consistently peppered my mailbox with bug reports and requests for new features. At times it seemed like death by a thousand cuts, but everyone is richer for the result.

3 NBT for Files

Heretofore, file selection boxes—which an application uses to request a file name—have had only limited navigation capabilities and almost no file manipulation or inspection capabilities. This section argues for the need of *file browsers*, for use in conjunction with and apart from an application. NBT extends the design of NEXTSTEP's file selection box [NeX92] and sets a new standard for file browsers in the areas of directory navigation and file inspection and manipulation. NBT is the second experiment in domain-specific information browsers.

With hundreds or thousands of megabytes of personal disk space augmented by much larger central file servers, the task of locating specific information can be laborious. Once one moves from editing a small number of files in a handful of directories and attempts to incorporate images from clip art or data libraries, reference archived documents, or tap centralized resources, one faces the problem of managing information. In the case of a file system, the information is typically organized into a hierarchy, but lacks any further annotation of or links between files, as contrasted to hypertext systems.

Current file selection boxes fail their users in each of the scenarios mentioned above, for at the point where the user must make a decision about whether to select a particular file or not, he does not have sufficient information. Forcing the user to choose a particular file, which subsequently can be deleted from the workspace and replaced, is obviously an inelegant, second-best solution, if indeed even a recoverable action. When referring to libraries of information, file names may be too similar to make the correct choice; one would like a general preview mechanism. When referencing archived files that may have been stored in a compressed format, which must be uncompressed before use, one would like to apply directly whatever massaging is necessary to make the file acceptable to the application. And when searching large repositories of information, as on a central server, one would like an efficient means of navigating the extensive directory hierarchy, annotating selected locations for future reference.

In essence, the problem of information management as applied to file browsing is that of (1) establishing location quickly and efficiently, and (2) working there (e.g., viewing, searching, printing, unarchiving), for after having invested the effort of establishing a context one should not be forced to a separate "work shell" to accomplish the task.

A component of the Ensemble Multimedia Editor of structured documents [GHM92], NBT addresses these concerns by bringing together various means of navigation, providing an extensible means of viewing and manipulating files, and learning from user interaction to customize itself. NBT is useful for navigating and inspecting files apart from or in conjunction with selecting them for use in an application. NBT is the next, best thing in file browsers.

3.1 Related Work

File selection boxes for graphical user interfaces have existed for at least a decade. Yet even today, many X Windows and even Sun's OpenWindows applications merely present a line to the user on which he is expected to type the full, often lengthy path name of the desired file [NeX92]. Motif [Ope91] and some X and Tcl/Tk programmers have improved on this by offering a point-and-click (no typing) mechanism. After considering a representative set of file selection boxes/browers, we examine NBT's solution.

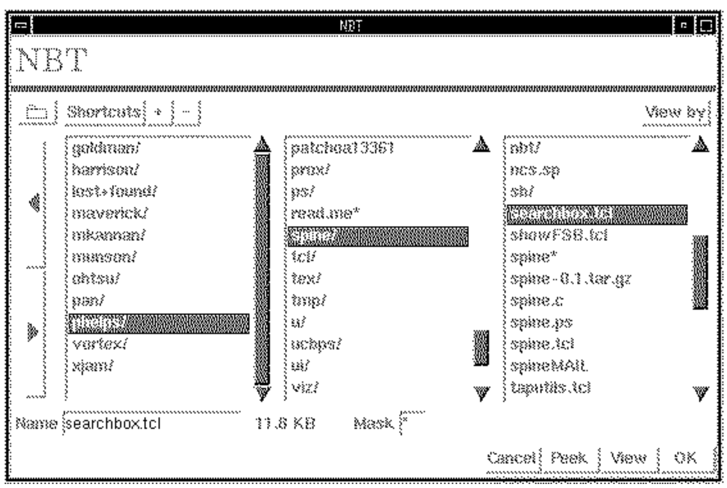
Before graphical user interfaces most programs and shells permitted the UNIX user to abbreviate the full path name with a set of meta characters indicating relative position to the current directory (‘.’, ‘..’) or relative position to one’s home directory (~). Sadly, all too many graphical applications fail to provide even this level of support. X Windows, the de facto windowing standard for UNIX workstations, does not ship a file selection widget, and file selection often written as an afterthought.

The Motif widget set provides a file selector with which one can descend into and ascend out of directories and select files that match a given mask, all with the mouse alone. Although it suffers from a weak set of keybindings as well as a directory and file display that unduly truncates most names, this file selector is usable. Another X file selection box [vdP89] gives a contextual view of three directories and, with a scrollbar, one can scroll up the entire directory path. Although not well publicized, it is a high quality, freely usuable widget. NEXTSTEP [NeX92] improves upon this contextual view with an iconic representation of files and a “shelf” upon which one can cache directories and files for quick access later.

3.2 Design

3.2.1 Navigation of Directory Hierarchy

NBT consolidates and extends the best features of file selection boxes from the Macintosh’s and NEXTSTEP’s, which it resembles, as shown below. Like NEXTSTEP’s, NBT presents a

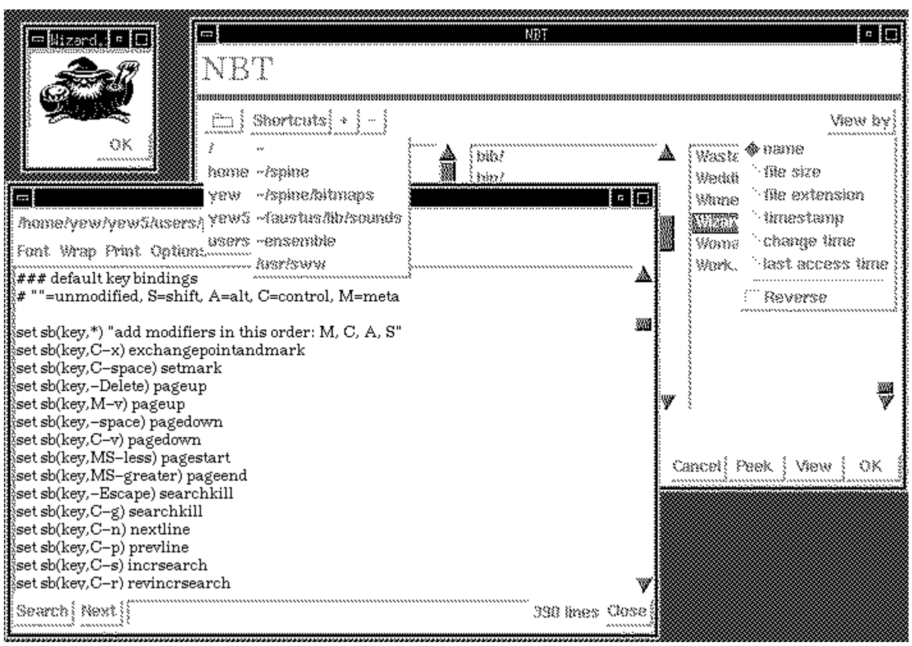


contextual view of the current directory and the *n* previous ones in the current path. One may navigate to nearby directories by clicking on directory names to descend into that directory, or by clicking on the left arrow at the left-most edge to move up. As on the Macintosh, NBT’s folder icon in the upper-left lists all the directories in the current path in a pulldown menu; releasing the mouse button over any component jumps to that location. Furthermore, the user may save commonly accessed directories on the Shortcuts menu. Clicking ‘+’ adds the current directory to the list and ‘−’ removes it; subsequently selecting this entry from the Shortcuts pulldown jumps to that location. Thus, NBT brings together and gives direct access to three modes of navigation: local (a single directory level up or down), global linear (any number of directories up, but only along the current path), and global (to any location in the full hierarchy, so long as it has been previously saved).

Finally, consider the following scenario. In putting together a newsletter, a user must repeatedly switch between a photos directory and a story text directory. Most present file selection boxes present the user with the previously-selected directory the next time it is invoked, which in this case maximizes inconvenience. With NBT, one could save the two directories as Shortcuts and switch between them quickly. However, NBT takes a further step by keying the default directory to the application-supplied title, so that a request from the main application to NBT to, say, ‘‘Import Bitmap’’ presents the last directory accessed with this message, which may be very different than the ‘‘Import Text’’ directory.

3.2.2 Inspection and Manipulation of Files

Having located a particular file with the navigation facilities described above, one must decide what to do with it. One may (from a ‘Utilities’ palette, not illustrated) execute simple commands that apply to a single file, for example deleting, unarchiving and uncompressing. For file inspection, NBT gives both ‘‘raw’’ and ‘‘semantic’’ views, both of which are illustrated in the screen dump below.



Upon selecting a file, users may ‘Peek’ at it or ‘View’ it. ‘Peek’ing shows the raw ASCII contents of a file, along with file meta information, e.g., creation time and file size. As in Emacs [Sta87], one may search the contents incrementally, character by character, either forward or backward. Regular expression searching is available in a text entry line at the bottom of the window. One may print out this information through a paper-conserving filter (`enscript`) or raw (`lpr`, useful for PostScript files), or select a region and make it the X selection. ‘Peek’ is useful for lengthy files in which only the header contains any human-decodable information (e.g., PostScript sources). Both ‘Peek’ and ‘View’ (described below) automatically decode common (e.g., compressed) formats for saved files.

'View'ing tries to determine the file's type by consulting a user-extensible suffix-to-type association list. If nothing matches, its full ASCII contents are shown. If a match is detected, the file is viewed in its "semantic" form, for instance a PostScript file is shown in a WYSIWYG view. Semantic matches may either (1) start up a process which is given the full path name of the file to view (e.g., `.ps` files start up a PostScript previewer such as Ghostscript and `.au` sound files are played on the speaker, if any), (2) use Tcl's `send` command to communicate this information to a running Tcl interpreter (e.g., man pages call TkMan), or (3) invoke custom Tcl (and hence C) code (e.g., bitmaps are shown on a Tk canvas widget). The most complex example of a custom viewer is that for `tar` archives. The semantic view shows that `tar`'s table of contents; selecting an entry (perhaps `README`) and clicking 'View' yet again retrieves the file from the (possibly compressed) `tar` archive and shows it, all without extracting it to disk, although that is also an option. Thus, viewing a PostScript file buried deep within a compressed tape archive file is only a few clicks away.

4 FoSel for Fonts

Not as technically deep as TkMan or NBT, FoSel* is here described to illustrate information browsing principals gainfully applied to a class of information that one might at first glance consider too simplistic to benefit. FoSel is the third and final experiment in domain-specific information browsers.

Tk applications invoke FoSel to identify a font. A typical X Windows installation has hundreds or thousands fonts, each descriptively named as follows:

`-adobe-new century schoolbook-bold-i-normal--0-0-75-75-p-0-iso8859-1`

This format compounds the font's "foundry", family, weight, style, spacing, size, and character set encoding. Clearly, it would be tedious to find such a specification from among the hundreds and then error prone to type it out by hand. FoSel is one of several good solutions to browsing X's large store of bitmap fonts.

4.1 Related Work

4.1.1 X Windows Font Selectors

The X11R5 software distribution supplies two good, complementary font examination tools. `xfontsel` mirrors the technical specification of a font with a row of pulldown menus, one for each attribute. By picking from the pulldown menu, the user locks that attribute into place, and the other menus disable options that no longer match an existing font. Attributes not yet determined match any value. The first matching font matching the specification is displayed in a window below the pulldown menus. One simply specifies attributes until the desired font appears. `xfontsel` is a good solution. For the novice, though, it is perhaps too directly tied to the technical specification of a font. Also, the introdution of new fonts into the system can change the font matched by the partial specifications.

Whereas `xfontsel` selects a font from the hundreds available and displays a representative sample of the characters, `xfd` examines one font in great detail, showing every character, the character codes and metrics. It solves the second half of the pick-and-show problem.

Neither `xfontsel` nor `xfd` is integrated tightly with X applications. `xfontsel` can pass the name of chosen font though the X selection, not the friendliest interface.

4.1.2 Macintosh Font Specification

One of the earliest popular machines with WYSIWYG page display, the Macintosh suffers under a second-rate font specification system. Perhaps because the small size of early Macintosh screens discriminated against large panels or dialog boxes working in close, extended connection with a main window, traditional font attribute specification takes place through various pulldown menus. Early word processors like MacWrite could devote an entire menu to font family, style and size, or combine two of these attributes in one menu. This situation provided quick access to the attributes, even if one had no preview of the font determined by the chosen attributes. Indeed,

*. FoSel is pronounced like "fossil" in hopes that this mystical chant will make the need for a bitmap font browser disappear and that outline fonts become ubiquitous.

third-party system code extensions that displayed each font name in that font are quite popular. However, the increasing sophistication of Macintosh applications claimed more and more menu space, and where fonts found a place, their explosive growth in numbers made selection unwieldly, as the menu of choices conceptually extended beyond the bottom of the screen, making for a noticeable wait for those fonts to scroll into view.

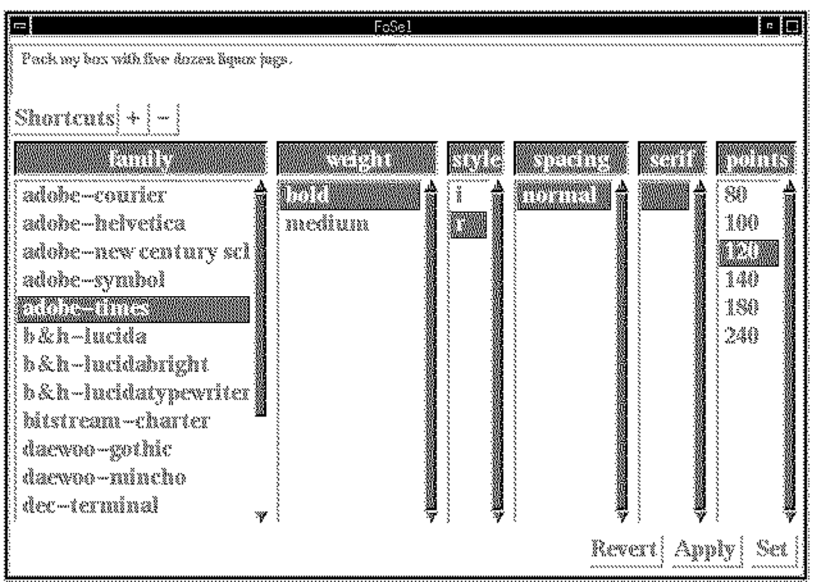
4.1.3 NEXTSTEP's Font Chooser

All NeXT machines boast large monitors, which NEXTSTEP populates with a variety of system-supported controls panels that work with applications without cramping or necessarily overlapping their windows. Compared to the Macintosh, there is less of a urgency to conserve screen real estate by placing lists of options in pulldown or popup menus. Given this luxury, control panels can group together logically related controls and make them available for quick change value-see effect loops, which is especially important for making small, precise adjustments. In the case of fonts, the font panel can both give attribute information about the current font, and let the user change any of its attributes, all of which are displayed simultaneously.

Each font attribute in NEXTSTEP's font panel is given its own list of available choices. A few mouse clicks quickly specifies the change from the current font. A preview image of the current font is displayed giving one the opportunity to decide on the appropriateness of the choice before committing it to the document. Font panels are tightly integrated with applications within NEXTSTEP's object-oriented framework. As with so much else in NEXTSTEP, its font panel is a model of user interface design.

4.2 Design

A casual comparison with NEXTSTEP's font panel makes clear that it was the inspiration for FoSel, which is shown below.



Like NEXTSTEP's font panel, FoSel gives each font attribute its own list of available values and previews the specified font. NEXTSTEP has the luxury of operating with high quality, well-behaved PostScript fonts that have a standard set of attribute values and an infinite choice of point sizes. FoSel must deal with X Windows' incomplete set of bitmapped fonts. Thus FoSel cannot hardcode attribute values. Instead it compiles a database of the fonts available to the system into a database. Using this database it is able to present only valid choices of attributes: the user cannot select a combination of attributes that does not correspond to an existing font. Lastly, FoSel slightly extends NEXTSTEP's model with the addition of a 'Shortcuts' menu, a user-configurable list of frequently used fonts. Like NEXTSTEP's, FoSel integrates easily into applications.

4.3 Implementation Notes

Attribute validity checking is FoSel's main technical features of interest. It works as follows. At initialization FoSel reads the list of system fonts and creates a set of lists—in effect a nested set of hash tables, but the Tcl's dearth of data structures forces certain compromises. The first list consists of all the font families. From each family name is created a list named *<family name>*`-weight` to hold a list of valid weights for each family. In turn, a variable called *<family name>*-*<weight>*`-style` is created to hold a list of valid styles for each valid family name-weight combination. This process continues for all attributes.

<u>*list name*</u>	<u>*values*</u>	
family	courier, helvetica, new century schoolbook, symbol, times, ludica, charter	
	choose times from listbox	
	extend to make key into second list	
times-weight	bold, medium	
times-medium-style	o, r	*(oblique and roman)*
times-medium-r-spacing	normal, compressed	
times-medium-r-normal-serif	(empty)	*(only serifed available)*
times-medium-r-normal- -size	80, 100, 120, 140, 180, 240	

final font specification: times-medium-r-normal--120-*

As diagrammed above, it is now a trivial matter to determine the valid attributes for given prefix of attributes. For any given attribute change, those attributes to the left remain valid. These are concatenated with the new attribute to give the key to a list of valid values for the next attribute. If the value of the next attribute remains valid in the next list, that value is retained; if not, heuristcs try to make a close match. In either case, a valid value is obtained. This propagation of valid values propagates left to right until all attributes are guaranteed valid values. A final step composes these attribute values into an X font specification string.

FoSel is configurable in the same ways as TkMan and NBT. In particular, by editing a single variable, the application implementor can change the list of attributes which the user must specify. For instance, since ‘serif’ usually has only one choice and spacing is usually ‘normal’ (as opposed to ‘compressed’ or ‘expanded’), the applications programmer may decide to supress these. Other applications may need to control attributes with their own mechanisms. Ensemble, for example, sets boldface and italics attributes to ‘true’ or ‘false’. The variable change drops ‘weight’ and ‘style’ from the cascaded lists and a small amount of Tcl/Tk coding changes control of these two attributes to toggle buttons. This demonstrates that applications with varying needs are accommodated in a first class way with little effort.

5 Conclusions

For a domain-specific information browser to be useful, the infomation set that it supports must be large enough to warrant a supporting tool and must exhibit some interesting organization of which general-purpose tools are ignorant. Together, the large size and organization of the data set make for a de facto database. The existing organization of the database may not facilitate browsing, and therefore its browser may need to internalize and reorganize it.

To provide usefulness over general-purpose tools, a domain-specific information browser needs to exploit its knowledge about the domain. Furthermore, since the organizer of the information could not hope to match the desires of all those who will use it, the information browser should support user annotations. These facts dictate that the browser should exploit the natural organization of the database and permit the user to mark interesting locations therein. Any interesting structure within each element of an information set can be seen as the same problem at a finer level of magnification and should be supported similarly, which means that the browser should exploit the natural fine-grain structure of each datum and permit user annotations on it.

When a domain-specific information browser meets these criteria as much as is possible, it becomes a domain expert. Now other applications that deal with this domain could usefully profit from the acumen embedded in the browser, which requires the browser to interoperate with them. If the domain-specific information browser is to integrate seemlessly, however, it may be necessary to make certain modifications, some cosmetic, others more significant, all of which should be reasonablely easy to make.

Despite the disparity of data types that the three browsers examined in this report promote—man pages, files in a hierarchical file system, and fonts—they fulfill to a significant degree the criteria outlined above. The table below shows the extent of compliance.

	TkMan	NBT	FoSel
database internalized	✓	(would be too large)	✓
hierarchical organization of information	✓ (customizable)	✓	✓
individualized user annotations (shortcuts)	✓	✓	✓
inspection of datum	✓	✓	✓
user annotations within datum	✓		(n/a—data type too simple)
interoperable hypertool	✓	✓	✓
deep, noncosmetic customization	✓	✓	✓

TkMan and FoSel both build an internal database of the large scale organization of the data in order to possibly rearrange it (TkMan's tkmandesc commands) or to build a more efficient access method (FoSel's nested list data structure). It is key to obtaining functionality and performance.

TkMan, NBT and FoSel all exploit the existing organization of the information; the hierarchical nature—either natural (man page, file system) or imposed (fonts)—gives rise to more powerful means of navigation. Yet for all the planning on the part of the information providers, inevitably users will want a different organization. By providing shortcuts (all) to various points in information space and in some cases highlighting places within that point (TkMan), the user can transform the information space into a more comfortable working environment.

It is not enough merely to find the point in information space; one must also have the power to examine it. This is most obviously necessary for TkMan, which would be useless without it. TkMan provides numerous means of working within a man page (see "Inspecting Individual Man Pages" on page 10). NBT lets the user examine a data file in its natural medium—text, bitmap, TeX dvi file, PostScript, sound, et cetera—and extend this repertoire in a straightforward manner. FoSel's display is uncomplicated but complete: it shows the representative sample of text in the chosen font.

Finally, all three have well defined interfaces to communicate with other applications. For a file browser and font brower this is a necessity. For TkMan, too, it is useful. Rather than trying to build viewers for all types of information, a generalized information browser could profit by calling upon specialized expert browsers, especially if they could interact coherently. It is certain, however, that no two users will have the same conception of how to customize the tool, and therefore personal customizations must remain personal—outside of the information space and outside of the source code (all, a lesson not learned by `xman`'s mandesc). To be truly useful, this customization must extend beyond mere cosmetics; it must embrace the transformation of information space, at the least.

Certainly not all information browsers will share all of these features. However, before designing a browser for a new class of information, one would do well to consider the lessons learned in the design and implementation of TkMan, NBT and FoSel.

6 Availability

TkMan, NBT and FoSel are available for anonymous FTP at `ftp.cs.Berkeley.EDU` in the `/ucb/mhgs` directory. This software is free for noncommercial use and distribution; permission to incorporate this software into commercial products may be obtained from the Office of Technology Licensing, 2150 Shattuck Avenue, Suite 510, Berkeley, CA 94704. All three require Tcl 7.0/Tk 3.3 or later versions, also available on `ftp.cs.berkeley.edu`.

7 References

[Ado93] Adobe Systems Incorporated. *Programming the Display PostScript System with X*. Addison-Wesley, 1993.

[App87] Apple Computer, Inc. *Human Interface Guidelines: The Apple Desktop Interface*. Addison-Wesley, Reading, Massachusetts, 1987.

[Bri] Bristol Technology Inc. Hyper help 3.0. Commercial software.

[Chr87] Tom Christiansen. The answer to all man's problems. Technical report, CONVEX Computer Corporation, 1987.

[CMN80] Stuart K. Card, Thomas P. Moran, and Allen Newell. The keystroke-level model for user performance time with interactive systems. *Communications of the Association for Computing Machinery*, pages 396–410, 1980.

[Dav93] Paul Davey. Three help systems: xhelp, X.desktop, and UNIXhelp. *The X Resource*, (6):141–151, 1993.

[GHM92] Susan L. Graham, Michael A. Harrison, and Ethan V. Munson. The Proteus presentation system. *Proceedings of the Fifth ACM SIGSOFT Symposium on Software Development Environments*, pages 130–138, 1992.

[Gol90] Charles F. Goldfarb. *The SGML Handbook*. Clarendon Press, 1990.

[Hem93] James J. Hemmer. Xhyper: A hypertext online help system. *The X Resource*, (6):153–164, 1993.

[HSC] HSC Software. Kai's power tools. Commercial software.

[NeX92] NeXT Computer, Inc. NeXT versus Sun: a comparison of development tools. Technical Report Seven in the NeXT Computer, Inc. White Paper Library, NeXT Computer, Inc., Redwood City, CA, January 1992.

[Nor88] Donald A. Norman. *The Design of Everyday Things*. Doubleday, 1988.

[Ope91] Open Software Foundation. *OSF/Motif Programmers Reference*. Prentice Hall, Englewood Cliffs, New Jersey, 1991.

[Ous90] John Ousterhout. Tcl: An embeddable command language. In *1990 Winter USENIX Conference Proceedings*, 1990.

[Ous91] John Ousterhout. An X11 toolkit based on the Tcl language. In *1991 Winter USENIX Conference Proceedings*, pages 105–115, 1991.

[Rao91] Baudouin Raoult. hman. Free software, October 1991.

[RCM93] George G. Robertson, Stuart K. Card, and Jock D. Mackinlay. Information visualization using 3D interactive animation. *Communications of the Association for Computing Machinery*, pages 57–71, April 1993.

[Shi93] Rei Shinozuka. Personal communication, November 1993.

[SP88] Barry Shein and Chris Peterson. xman. Part of the X11 software distribution, 1988.

[Sta87] Richard M. Stallman. *GNU Emacs Manual, Sixth Edition, Version 18*. Free Software Foundation, Cambridge, MA, March 1987.

[SV93] Kent J. Summers and Jeffrey L. Vogel. Proposal for an X-based online help protocol. *The X Resource*, (6):7–24, 1993.

[Tof90] Alvin Toffler. *Powershift: Knowledge, Wealth, and Violence at the Edge of the 21st Century*. Bantam, 1990.

[Tuf83] Edward R. Tufte. *The Visual Display of Quantative Information*. Graphics Press, Cheshire, Connecticut, 1983.

[Tuf90] Edward R. Tufte. *Envisioning Information*. Graphics Press, Cheshire, Connecticut, 1990.

[vdP89] Erik M. van der Poel. Select file widget. Free software, 1989.

[Whi93] Kennard White. tkinfo. Free software, 1993.

A Reviewer Guidelines for TkMan

`Date: Sat, 21 Aug 93 22:40:34 EST From: csr@staff.cc.purdue.edu (Comp-sources Reviewed) To: phelps@ginkgo.CS.Berkeley.EDU Subject: guide-2 part 00/03`

`>From csr Fri Dec 11 14:00:08 EST 1992 Submit guide-2 00/03`

`This 'product' is the guide for reviews and guide for submissions to comp.-sources.reviewed. If you are going to be submitting anything you should really read these! {So you won't be rejected for a missing 'patchlevel.h' or something else simple to fix.}`

`You will also want the help file from the mail-server (send a subject of 'help' ot this address.`

`Thanks! ksb`

Date: Sat, 21 Aug 93 22:40:45 EST From: csr@staff.cc.purdue.edu (Comp-sources Reviewed) To: phelps@-ginkgo.CS.Berkeley.EDU Subject: guide-2 part 01/03

>From csr Fri Dec 11 14:00:08 EST 1992 Submit guide-2 01/03

Comp.Sources.Reviewed Guidelines for Reviewers (Version 2.0)

This document is designed to provide guidelines and suggestions for use while reviewing software for the news group "comp.sources.reviewed". It also provides potential submitters with information about what the reviewers will be looking for, and may be useful to others who are asked to evaluate software.

These guidelines may change as we gain more experience in reviewing software, and comments are welcome.

Getting Submissions to Review

Comp.sources.reviewed (CSR) uses a volunteer system for assigning sources to reviewers. When submissions come in, a Call For Reviewers (CFR) is prepared describing the software and the kinds of equipment and expertise that is needed to test it. This CFR is posted to comp.sources.testers. Potential reviewers who are interested request the entire submission via a mail-server. Thus, everyone has complete control over what they review. You are also able to volunteer for reviews at the times when you are not too busy with other things (just to check packing and 'style').

Given this procedure, it is expected that you return your reviews in a reasonable period of time. If you find that you cannot provide a review of a submission in reasonable amount of time (say 1-2 weeks), then please inform the moderator so he does not waste time waiting for you. Also, if you find that once you get the package you cannot review it because of missing equipment or expertise, please inform the mail-server as soon as possible.

To request an index of the packages up for review send a message with the subject line like 'Subject: send index' to csr@cc.purdue.edu.

Record Your Work

A crucial part of the review process is recording what you do. You must make complete and accurate notes of your time spent working on the package. Don't rely on your memory to record the problems or suggestions you come across, because you will forget quickly as you move on to other things. So, the first thing to do when you get a submission is to start documenting everything that happens. Since you will probably be working on several machines, you may have to work with pencil and paper (gasp).

Evaluating the Format of the Submission (Do this first.)

Was it in the form of a "shar" file, or similar packaging appropriate for the architecture?

Did in unpack correctly?

Did it unpack in the places you expected it to?

Did it contain a MANIFEST file listing all the parts of the submission?

If it passes these checks send a 'got package' note to the mail-server. This will clue the moderator in that the package might be worth extra time and effort. If the package doesn't pass these checks send a 'nix package' to the mail-server so others will not waste time on the package! {The moderator will remove it from the archive.}

Evaluating the Description and Purpose (Take the time to do it right.)

Is there a README file that contains:

- the purpose and value of the software (give details here) - the types of systems it is intended for (e.g., BSD only, Unix and DOS, etc.) - any dependencies in the system (e.g., must have perl to run) - known limitations of the software - the authors of the software (with e-mail addresses) - the "patchlevel" of the software (see below) - any copying or distribution conditions

Is there a "patchlevel.h" file (or equivalent) indicating the version number of the software?

Building the Software

Is there an Installation document explaining how to build the software?

Are the options and conditional parts of the package (e.g., do this for DOS, this for System V, etc.) clearly documented?

Is there a proper Makefile? (Imakefile for X software?)

Did the make run correctly?

Are building and installation two separate steps?

Did the software build on each of the architectures you were able to test?

NOTE: It is important that you give details about the environments you used for testing in each and every review that you do. Make sure you describe the hardware, OS, compilers, etc. that you used, and any other information that is relevant. This information will be used to evaluate the portability of the package, and will be posted along with the sources to let the readers know if the package is appropriate for them.

At this point you *may* send in a 'preliminary review' to the mail-server. Mark it as 'review package' (multiple reviews are OK). Multiple reviews for multiple platforms you might have are also fine.

Testing the Software

Did the software run on each of the systems you tested?

Did the program perform the functions it was supposed to perform?

It is not your place to fix the software you are reviewing. However, if you find a problem with an obvious solution, make sure you document it so the authors can make use of it.

The Documentation

The documentation is a very important part of a submission, so it should be reviewed carefully. The documentation may be built-in to the program and be in the form of document files and/or man pages. It is difficult to give simple rules for documentation, and not all programs require the same amount of documentation, but here are some things to consider.

General Guidelines

Is the documentation written in a form that is clear, precise, and easy to understand?

Is each feature or function of the program documented?

Is the documentation organized? Sections and sub-sections often make documents easier to read and understand.

Documentation should be provided in "text only" form. An author may choose to also provide formatted manuals that use PostScript or TeX, but a readable form should be provided.

Guidelines for Unix Documentation

Is there a man page? Man pages should contain the following sections (at least):

NAME required SYNOPSIS required AVAILABILITY optional DESCRIPTION required OPTIONS required if there are any command-line options ENVIRONMENT required if there is provision for environment variables FILES required if use is made of other files SEE ALSO optional DIAGNOSTICS required if the program can produce debugging output NOTES optional BUGS required if any are known AUTHOR optional

These sections should be complete (e.g., all the OPTIONS must be described).

Guidelines for VMS Documentation

Is there a VMS HELP file? HELP files should contain the following sub-topics (at least):

Parameters if there are any Command_Qualifiers if there are any Examples if appropriate

Is there additional documentation to supplement the VMS HELP file?

Guidelines for DOS Documentation

There are no standards for DOS documentation, and this makes the review process difficult. We suggest that the information and features described above should also be present in DOS documentation, although they may not be in the form described above.

Functionality and Features

Does the software perform some function that is valuable?

Are there obvious features or additions that would improve the package?

Overall Evaluation

What is your overall evaluation -- would you recommend it to a friend?

Would you suggest that this submission:

- be accepted for posting as is - be accepted after minor revisions (that you have detailed) - be returned to the author with a recommendation to make major changes and re-submit - be rejected

Short Summary of Review

Regardless of your recommendation for the submission, you should provide a short summary (one or two paragraphs) of your review. If the package is posted, this summary will be posted with it. Otherwise, it will be used by the moderator to determine the appropriate action to be taken with a submission. This summary might mention why you found the package useful, what you liked about it, what machines you tested it on, and what limitations you did find. Your name will NOT be attached to this review summary when it is returned to the authors or posted.

Submitting Your Review

The final step is to return your review to the mail-server. Please make sure you put the correct package name in the subject line.

To test the review facility review the test product: To: csr@cc.purdue.edu Subject: review test

Test.

DO NOT send your review to the author. The moderator will collect all the reviews, prepare a grand summary, and forward it to the author. You will get a copy of this grand summary to allow you to compare your work with other people's reviews.

Your review should provide as much detail as possible. Make sure you give details on the kinds of machines you tested on. You may choose to use this file as a template to record your review.

Contacting the Authors

Your identity will NOT be revealed to the authors unless you choose to do so. If you feel that you would like to contact the authors for clarification, you may contact them directly or ask the moderator to forward your question. You should only contact the author for clarification or additional information needed to complete your review. You should not comment on the submission at this time, but instead save your comments for the report you send to the moderator. You should also ensure that your interactions with the author are conducted in a courteous and professional manner.

In addition, you should ensure that the other reviewers working on that submission also receive the information you get back from the authors. Further, you must do so in a manner that does not interfere with their wishes not to identify themselves to the authors.

It is not a good idea to suggest that authors make patches to software during the review process. If you find a problem that the authors are able to fix easily, document the problem and suggest that minor revisions are needed before the submission is posted. It is important that you are not evaluating software that has been patched, while others are working with the original submission.

You should not send the author any comments about, or evaluation of, the submission. That information should be sent to the moderator in your final summary, and he will collect the comments from all the reviewers and forward them to the author if appropriate..

Thanks for your time! ksb