OLH:
An On-Line Help Facility for Managing Multiple Document Types in Their Native Formats in a Distributed Environment

Kevin M. Cunningham
Senior Technical Writer/OLH Project Manager
Athena Documentation
MIT Information Systems
kcunning@ATHENA.MIT.EDU

ABSTRACT

In trying to construct and maintain a useful and comprehensive on-line help system that organizes all the material available in a heterogenous distributed UNIX-based environment, our documentation group encountered three fundamental difficulties in dealing with existing online documents:

- an incompatible variety of formats
- no central location
- political constraints

Rather than spend our time converting and relocating perfectly good existing documents, we developed a menu-based help browser, OLH, that allows us to maintain the documents in their native formats and in their original locations in a way transparent to the user. The OLH system:

- makes all our online documents accessible through a common interface
- supports on-the-fly conversion of some document types to support users who can't view the documents in their native formats (e.g., over dialup)
- allows us to maintain the hardcopy and on-line versions of many documents in single-source modules

OLH is built on an operating-system-independent database, and currently has interfaces for devices supporting the X Windows System and plain-text terminals.

BACKGROUND

ABOUT PROJECT ATHENA

Project Athena at MIT was a major research project sponsored by IBM and Digital Equipment Corporation to explore distributed computing on a larger scale than had hitherto been attempted. The goal was to create a system of engineering workstations linked on a vast network to central services such as file servers, printers, and system software servers. The user base for which Athena was designed was the entire academic community at MIT, an estimated 10,000 users.

In 1991, Project Athena ended as a funded research project, but the Athena system continues to serve as the campus-wide academic computing resource for the MIT community. Currently, about 1,300 workstations in more than 40 clusters are connected to a campus-wide network, enabling users to communicate over the system and access their data as well as other resources from file servers and other machines that are network-connected. About 150 courses in more than 20 departments at MIT require students to use Athena.

Athena is structured so that a student may sit down at any Athena workstation on campus, and have access to his or her own customized environment and personal files. These files, as well as the software applications, are located on file servers which are scattered throughout the campus, but appear to be located on the local workstation once the user has logged in and his home directory is automatically mounted.

Athena provides a bridge between the two familiar extremes of stand-alone personal computers and timesharing machines. Each user has a dedicated, powerful multitasking computer at his or her disposal. At the the same time, users access a number of shared services that would normally be available only on a central facility.

DOCUMENTING THE ATHENA SYSTEM

As a research/development project aimed at constructing a vast distributed system, Athena has always been difficult to document. In addition to being a moving target because it was still in development, the distributed nature of the Athena system posed a special problem: where in this vast network of computers were the information resources?

As an additional complication, in the Athena model not all users have the same view of the complete distributed system. Rather than have the entire system directly accessible all the time, each user adds one or more filesystem subsets (called "lockers") to his or her local "world" as desired— so each user has a distinct subset of the entire distributed system "attached" to his or her workstation at any given time. There was no single system to document.

Furthermore, the contents of the various lockers were never centrally registered. So a practically numberless set of new filesystems could become available, each potentially a repository of a significant online information resources, yet no-one was keeping track of their contents from a central point of view. Not even Athena knew what was "out there" on the network. How could even the Documentation group, nevermind the users, be expected to find all the riches available online?

Even when we stumbled across caches of information, they were not in the format we were using for our own documents. Instead, they were in troff format, or were PostScript documents for which the original Scribe or TeX manuscript files were curiously missing. We had no way to revise them, or to regenerate them into another format. And we had no mechanism for including them with the documents we were creating — they were just in a different format, and we had no way to mix apples and oranges. We had not yet figured out that we could build a crate to put both in.

FIRST ATTEMPTS

Our documentation group made what efforts it could toward providing useful free hardcopy documents, but there were only so many topics we could cover given our meager resources. As a first effort at online documentation, we assembled the PostScript versions of our documents into a directory online and included a special document displayer so that users could look at the documents. Users could also print out a complete document if they wanted to. The displayer was limited and the results barely acceptable, but it was a start.

In 1988, the first phase of Project Athena ended, and funds were no longer available to provide the user community with free printing. This meant both that the Documentation group could not give away documents anymore, and that users who printed the online versions of documents would have to pay for printing them (as they now had to pay for any files they printed). Either way, the users lost big.

We realized that, if Athena was to become usable to the community, we would have to find ways to provide users with access to information without requiring them to pay for it. For us, that meant two venues: locked racks of hardcopy documents for each terminal room, and online documentation.

A NEW ONLINE SYSTEM

For online documentation, we realized that we would have to create a better system than the one already in place: the displayer was inadequate, and the information was simply a hardcopy document placed online. We would need to create more modular documentation and organize it better. Modular documents would be easier for users to use online, and smaller documents would cost less for users to print out (users could find the specific piece of information they were looking for and just print out that module rather than a whole document). So we began to think about "online help" rather than simply "online versions of hardcopy documents".

Over the course of months, we determined how we would convert our existing documents into modules. This included not only segregating sections of documents into separate plain-text files, but also re-thinking how these modules could be assembled in a menu-oriented system rather than a hardcopy-oriented system.

In starting to build a tree of modular documents, we also thought about bringing other online resources into the picture. We knew that there were vast stores of information out there on Athena itself, if we only knew where to look. Here at the outset of our design process we had a great opportunity — to design our new help system to actually incorporate those other caches of information, rather than ignore them.

We conducted a survey of online resources, and found nearly a dozen significant information resources that would be immediately usable. Unsurprisingly, the information was all in different formats, and all in different locations.

THE PROBLEM OF HETEROGENEOUS DISTRIBUTED INFORMATION

Our fundamental problem then was how to make these different resources easily available to users. Three key difficulties, it seemed, stood in our way:

- The existing documents were not all in the same format, but rather existed in an incompatible variety of formats. Each different lode of information had its own eccentric display mechanism and storage format. (Among the formats we have to deal with on a regular basis are plain ASCII, TeX/LaTeX-generated dvi, Scribe-generated PostScript, Andrew EZ, and troff/nroff.)

- The available documents were not centrally located, but were instead found in various often obscure locations all over the network. Furthermore, there was no single access point for finding information (no single command or exhaustive system map).

- Even if we had proposals for overcoming the first two difficulties, the political climate was completely against a standard solution. Not only were we a meager documentation group hoping to influence "the powers that be" (a dire proposition under the best of conditions), but the culture forbade a standard solution. This was a development project after all — people wanted to continue to explore *new* ways of doing things, not squeeze everything into the tried and true (or even the untried but well-designed).

The result of the problem was palpable — Athena resources were not being used efficiently:

- Knowledge was not easily shareable among users.
- The time it took to learn how to use system was very long (how could you learn about the system when it was so difficult to even know where to look?)
- Users couldn't easily generalize their knowledge of the system (they knew what they had learned could just be a nook of the system, not necessarily something fundamental).
- Even developers frequently ended up re-inventing the wheel because they did not share information with each other.
- There was a general lack of awareness of many available system features (even among the Athena staff).

For all these reasons, it was difficult for community-wide "folk wisdom" to evolve, except for technically-minded hackers. Athena was universally perceived, correctly, as an "insider's system." And the continued proliferation of isolated caches of information created an extraordinary drain on the user support services, including documentation — how could you possibly support a system whose dimensions bordered on the infinite?

In addressing the problem of how to deal with distributed information that exists in a variety of formats, we asked many questions: How are documents in such different formats to be accessed by users without utter confusion? Do we have to instruct users in all the various commands used to display the documents, or should we select one format and convert all the documents to that format, if that's even possible? Is it feasible or desirable to retain all the documents in their native formats? Should we pick a limited number of supported formats and ignore documents outside that format, or is that not comprehensive enough? Can we provide a unified approach to the various documents, even if they are in different formats and spread across the network?

CHARACTERISTICS OF A SOLUTION

We did not have the programming resources or political clout to solve these problems in the "obvious" way (i.e., convert the documents to a common format, and move them to a common location), nor, after some thought, did we even desire this: we did not want to spend our time converting and maintaining perfectly good existing documents — we just needed to help users get to them.

As the main (but not sole) channel for all Athena-generated documentation, we knew our system would have to:

- provide a single point at which a user can always go to start a search for information (this did not imply single locus of information, or inaccessibility of info by any other means)
- help users find and access the information they need (display the information or at the very least identify where the information can be found)
- be useful to all levels of users (not just technical information)
- be easy to use for all levels of users (self-evident interface, reasonable response/access time)
- be easily extensible (allowing new information to be added)
- interact smoothly with existing methods for storing and delivering information (i.e., not require major revisions to already existing methods)
- employ a consistent, easy-to-maintain, and easy-to-learn support methodology (the maintainer's side of the system could not be prohibitively complex)
- be portable enough that the information could be easily transferred to an alternative system in the future, if appropriate
- be funny

CREATING OLH: THE ON-LINE HELP FACILITY

With these ideas in mind, we designed a menu-based help browser and fought for the development resources to implement the system. As always in such situations, the resources came to us the "wrong" way and for the "wrong" reasons, and with appalling slowness — but they did come. And so, over the course of two years, the part-time contributions of half a dozen programmers helped create OLH, Athena's On-Line Help facility.

One would expect that such haphazard contributions would produce a mess, and typically that is true. But in the case of OLH, the documentation group maintained a very strong presence at every stage in the development process. The key was evolving the specifications for the software within the documentation group, taking lots of feedback on the proposal (technical expertise from the developers was very important here), and making sure that the final say for the development of the system was with the "buyer" (i.e., the documentation group), not the "seller" (the development group).

This product orientation in the development process was anomolous and troublesome even by Athena standards. OLH was one of the very few projects for which the Project Manager was not either in upper management or in the development group itself. Having the managerial center in an "outside" group made it difficult to get strong support for the project at certain critical points. Not only were we contradicting the cultural idea that "development knew best", but, more importantly, we were facing the tremendous inertia of the idea "nobody cares about documentation — we have bigger technical problems to tackle". Furthermore, this was a research project — the idea that development should produce a program according to design was at odds with the most fundamental approaches of the hacker mentality ("built it first, play around, then see what you've got").

But through persistence and clarity of vision, we eventually obtained the resources we needed, and the expertise of several excellent programmers. And OLH, though still incomplete relative to its complete design, was finally created.

THE OLH SYSTEM

ABSTRACTION OF THE DOCUMENT HIERARCHY

The key to OLH's success in dealing with the problem of distributed information is that it is built on an abstract operating-system-independent database. The hierarchy of menus and documents is not based on the actual file locations, but is instead abstracted. Each online document is given a unique identification in the database, and it is these ID's that are organized into a hierarchy — the files themselves can be anywhere on the system. For example, here is the database information for one document:

type	DOCUMENT
node-id	dialup_step1
label	Step 1 -- Setting Up Your Terminal and Modem
keywords	terminal:modem:dialup:communications software
primary-parent	account:dialup_menu
file-location	/mit/logos/olh/Account/dialup/step1.olh
filesystem	logos
file-format	ez
author	Kevin Cunningham
maintainer	kcunning

This module is known to OLH as dialup_step1, even though the operating system knows it as /mit/logos/olh/Account/dialup/step1.olh.

In fact, the most difficult tasks for the maintainers of OLH were gathering the information they needed to load the database information, and designing an intelligent hierarchy of all the Athena information. In other words, the hard part was finding everything and making sense of it all, which are intellectual problems, not software issues. Once those tasks are addressed, the maintenance of OLH is, in principle, extremely simple.

(Unfortunately, one of OLH's biggest weaknesses at this time is the lack of an easy-to-use interface for loading database information. Currently the maintainers must edit database files by hand, and perform all the integrity checking mentally. The work is all extremely simple, but it is time-consuming. Because this is not a user-visible problem, this limitation of OLH has not received the development attention it needs — and will have to get before OLH becomes a viable program for exporting to other sites — but it is the highest development priority for OLH when resources become available.)

OLH DOCUMENT DISPLAY CAPABILITIES

OLH's database abstraction addresses the problem of distributed documents. The other key element of OLH is how it addresses the problem of heterogeneous document types.

Rather than try to force all documents into a standard type, which would involve resources we did not have, we decided to show documents in their native formats. We were able to do this by identifying each document in terms of what type of viewer is appropriate for it. One of the pieces of information gathered into the database about each document is its file format. The list of recognized file formats is also abstracted elsewhere in the database (it is trivial to add a new file type).

For example, one document style we recognize is latex, that is, a document generated using the LaTeX macro package of the TeX formatter. Such files usually have the file extension ".dvi". In our database of file-formats, we have the following entries for latex:

type	FILE-FORMAT
label	Files in LaTeX .dvi Format
node-id	latex
tty	dvi2tty $file | more
X	xdvi -geometry -0+0 $file
lptln	dvi2ps -r $file | lpr
PostScript	dvi2ps -r $file | lpr
edit	emacs $file

When the user selects a document that happens to be in the latex format, OLH checks the file-format database and determines what command it needs to use to startup the appropriate displayer for the file. If OLH were running X windows, OLH would call the xdvi program. If it were on a terminal-like (tty) display, OLH would use the dvi2tty program to convert the file to a plain-text form and then display it using a plain-text scrolling program like more.

LEAVING DOCUMENTS IN THEIR ORIGINAL FORMATS

When we originally considered showing documents in their native formats, our collective aesthetic sense rejected the idea instantly -- "What an ugly thing, to show a plain-text file, a PostScript file, and a LaTeX file on the same screen! a brutal clash of styles presented by the Documentation group itself! No, no, we would have to convince all the people writing online documents to move toward a common format, *then* we could show all the documents...."

But we quickly realized that this idea involved a mistaken orientation toward Athena and distributed computing. We were still hoping that Athena would turn into some homogeneous complete computer system that we could document, something like the computer systems we used to work on or that existed elsewhere (like the Macintosh). But Athena was anything but this wished-for unified system, and was unlikely to suddenly standardize itself before our eyes in the time it would take us to create our help system.

And as we thought about it, we realized that it made sense for the document delivery system to show different documents as they had been created. This would reflect the way Athena really was (which in itself is an important lesson for users), and showing the document as written perhaps best reflected the intentions of each document's authors (who had presumably selected the particular format for their own aesthetic reasons).

This was not to say we should not organize the material. The OLH menus *should* be standardized, but that had nothing to do with the display types. This important distinction between the browser (the map to the information) and the displayers (the actual presentation of the information) was rigidly maintained throughout the design process, and is a fundamental principle of the resulting OLH.

One other notion also eased our minds: certainly we would like to drive people to start using the same formatting tools, for unity of presentation, but how could we achieve this goal? Our own aes-

thetic convictions were not going to be enough to convince some system programmer elsewhere on the network to use our formatter of choice. But, if we presented all the documents in one place, perhaps this vivid presentation of clashing styles would touch the aesthetic sense of these other writers and they would naturally want to move toward standardization.

And if they didn't, maybe standardization of presentation was not that important anyway.

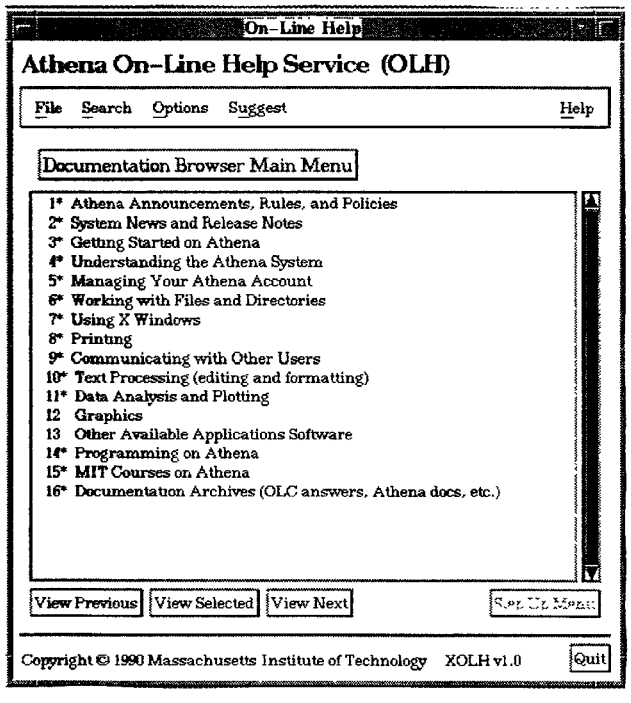
CALLING OLH

To start OLH, an Athena user can simply type the following at the system prompt:

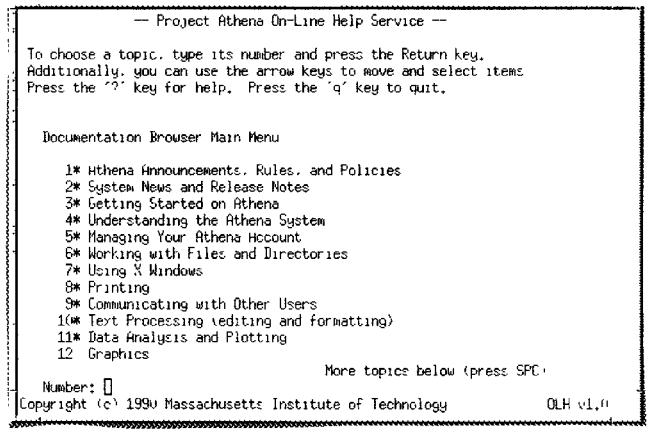
help

Called without arguments, the help command brings up the initial OLH menu. OLH currently has interfaces for devices supporting the X Windows System and plain-text terminals.

If the user is on a workstation running X Windows, a version of OLH based in the Motif toolkit appears as a separate window:



If the user is not on a workstation running X Windows (e.g., if the user has remotely logged into Athena via the network, or if the user has dialed in to one of Athena's dialup servers), a version of OLH based in the curses terminal-oriented interface appears:



A user can also force the terminal-oriented version of OLH to appear on a workstation that is running X Windows by the following command:

help -tty

For the X-based version of OLH, users can also use the **-internal** or **-external** flags to select whether OLH should use the built-in information viewer (a plain text viewer) or to call external viewers (e.g., EZ for EZ-based documents, xdvi for LaTeX documents, etc.) External viewers display the help information in its original format, but they have the disadvantage of taking up more memory and being slow to start up. By default, OLH uses external viewers if they are available (and if the processor is sufficiently powerful to run them without trouble). On smaller workstations, OLH uses the internal viewers by default.

Users can also specify a keyword or module-identification string to have OLH go directly to a specific topic. For example, the following command brings the user to the OLH entry that has been mapped to the keyword printers (or to a menu of choices if there are multiple modules mapped to this keyword):

help printers

The following command brings the user directly the the module that OLH knows as the sending module in the group of modules concerning electronic mail:

help @email:sending

The help command relies on several other commands, configuration files, and database files, but these are transparent to the user and so will not be described here.

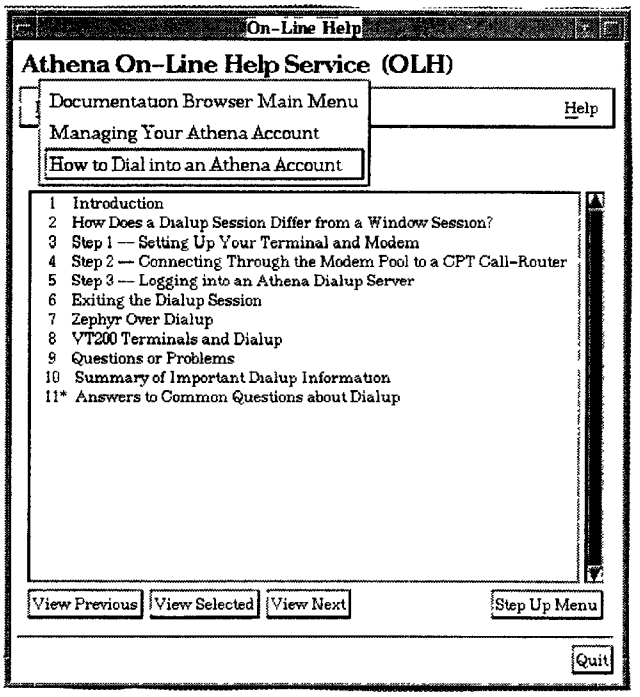
OLH FEATURES

OLH has many features to help users navigate through the document hierarchy. (The following examples are taken from the Motif version of OLH, but all of the features listed that are not directly related to window capabilities have equivalents in the terminal-oriented version of OLH as well.)

The main browser window of OLH lists the current menu of topics. Above the menu is a label indicating where you are in the help tree. The topics themselves are numbered, and any items that are actually submenus are indicated with asterisks.

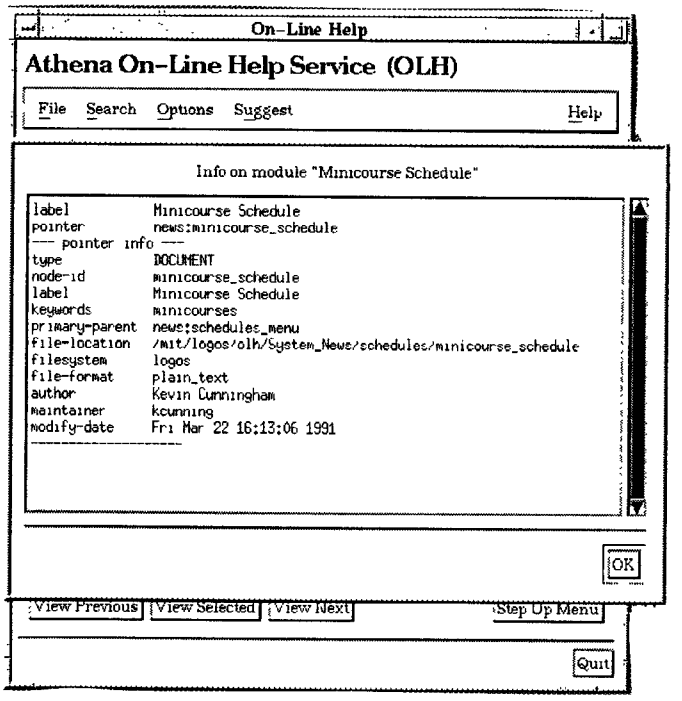
To select an item, you can simply double-click on it, or you can single-click on it then click on the View Selected button. To automatically go through topics in the order listed (i.e., to read them as if they were sections in a document), you can use the View Next button repeatedly. View Previous provides the reverse order.

If you enter a submenu, the Step Up Menu option allows you to return to the menu from which you got to the current menu. You can determine where you are in the hierarchy by clicking on the menu label itself; a list of the menus you have traversed downward from the main menu is displayed. You can return to any of the intervening menus directly by moving the cursor to that menu:

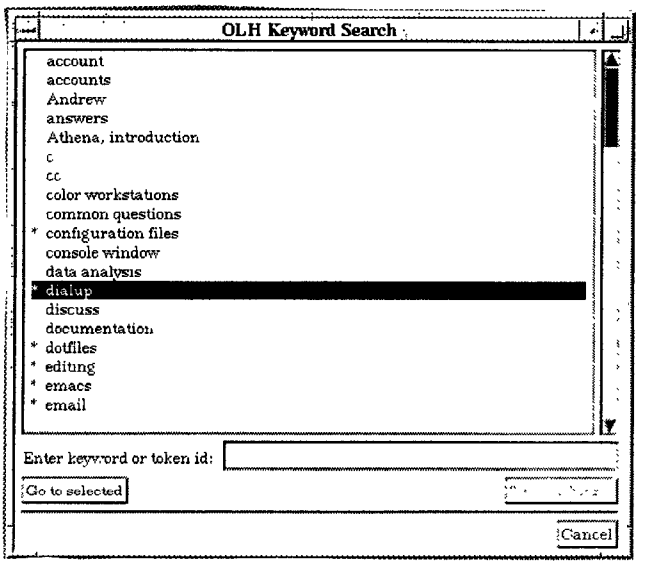


Apart from these navigational tools, OLH offers other tools to help users to get around the menus and make use of the documents.

The File menu offers options for working with modules apart from viewing them. You can send the currently selected module to the printer of your choice, or copy the module to a file, with a name that you choose. You can also show database information about the currently selected module, such as the date the module was last modified, who wrote it, and where it is located on the system:



The Search menu currently offers only one option, Keyword Search. This presents a menu of currently-recognized keywords, and a box to enter a keyword:



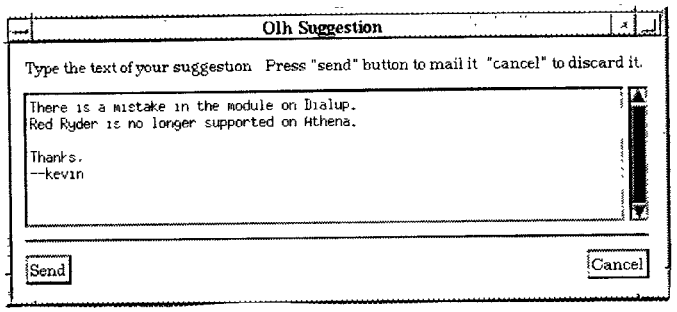
You can either scroll through the menus and select a keyword using the mouse, or you can enter a keyword in the dialogue box. As you type the letters in the box, the menu of keywords will scroll to the keyword that matches the letters you are entering. (You can also enter a module-identification string at the keyword box.)

When you select the keyword of interest, the keyword menu disappears and the OLH browser switches to the menu containing the topic associated with the keyword you chose. If you chose a keyword associated with a text module, that module is automatically displayed.

The keyword menu is like any other OLH menu — keywords that have several associated topics are indicated with an asterisk. If you select such a keyword, an intermediate keyword menu is displayed.

The Options menu offers you miscellaneous options. The External Viewers option toggles the behaviour of OLH between the -internal and -external flag behaviour noted earlier. If you find that your machine doesn't have enough memory to support the full format-oriented viewers, you can still have a window version of OLH running, but the documents (where possible) will be converted to plain text and shown in the built-in OLH plain-text viewer.

The Suggest menu lets you give your feedback about OLH or some document that you have displayed via OLH, so that problems can be corrected or new features added:



Both options bring up a text-editing box so that you can create a message and send it to the OLH maintainers. When the message is sent, the location of the user in the help tree is automatically added to the message, to help the maintainers in case the user neglects to put detailed information about what module is affected.

The Help options give information about how to use the help window and the other options.

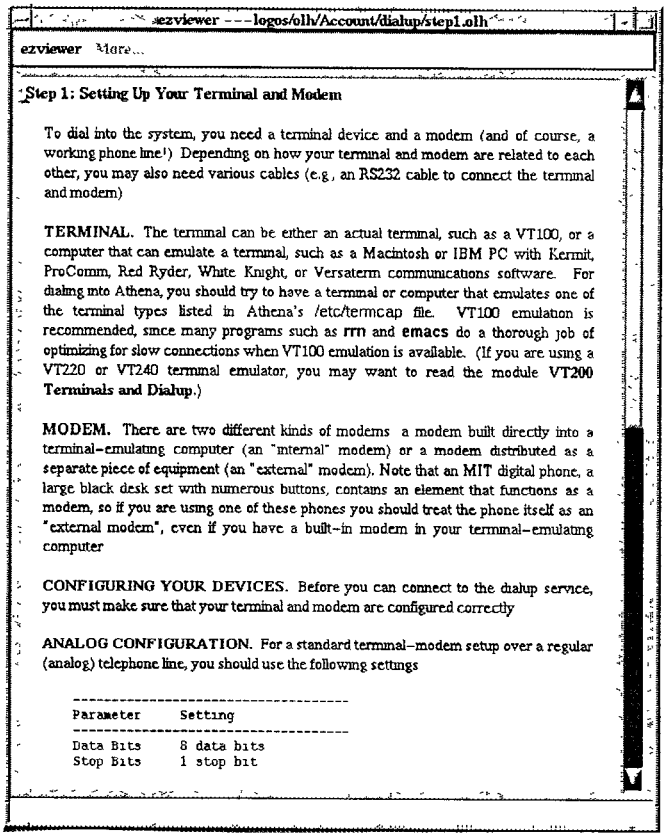
OLH DOCUMENT DISPLAY CAPABILITIES

As explained earlier, when the user selects a document that happens to be in a particular format, OLH checks the file-format database and determines what command it needs to use to startup the appropriate displayer for the file. If the workstation were running X windows, OLH would call a window-oriented displayer; if the user were at a terminal-like display (e.g., dialup), OLH would convert the file to a plain-text form (if possible) and then display it using a plain-text scrolling program.

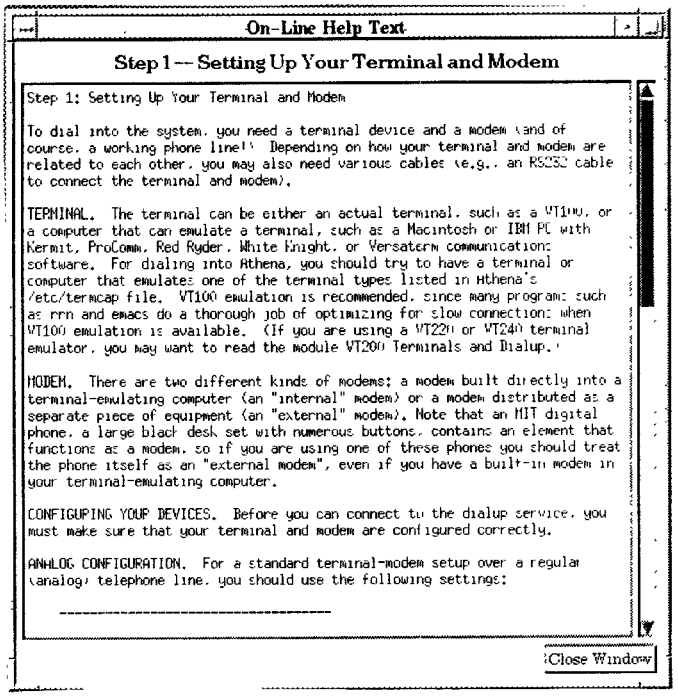
If the user had selected the -internal flag in calling OLH, the program would likewise convert the file to plain-text, but would send it to the built-in text viewer that the X Version of OLH has.

For some document types, there is no program on the system to convert the document to plain-text (e.g., Athena has no program to convert PostScript documents to plain text). For documents of this type, if the user is on a terminal-oriented screen, OLH simply puts up a message saying it can't display the document; if the user is in an X session but has selected -internal, OLH still calls the PostScript previewer (-internal means "show documents as plain text *if possible*"). In other words, OLH does what it can.

Documents are shown in their native format by running separate programs that produce separate windows appropriate to that format:



Plain-text documents are shown in a separate window created by the main program (i.e., not spawned separately). For example, here is the same document as shown above, as it would be displayed if the user had selected the -internal flag (this document type, ez, can be converted to plain text):



In addition to the Close Window button and scrollbar, the built-in plain-text OLH window supports many Emacs-style key bindings for moving around the window.

SINGLE-SOURCE MAINTENANCE

For users, OLH is a piece of software. We in Athena documentation tend to think of OLH as a complete system for efficiently creating, maintaining, and delivering online and hardcopy documentation. OLH was not just a solution to an intellectual problem about collecting distributed information — it was a very practical approach toward streamlining our work.

In implementing our design goals for OLH, we were able to include certain features that make the OLH system particularly easy to maintain and useful to our writers:

- OLH allows us to generate the hardcopy and on-line versions of many documents from a single source.
- OLH supports on-the-fly conversion of some document types to support users who can't view the documents in their native formats. This effectively allows single-source control of formatted and plain-text versions of documents.

For many documents, therefore, OLH allows for single-source generation of three delivery types: hardcopy, formatted online, and plain-text online.

One of the keys to this was the evolution of an effectively WYSIWYG (What You See Is What You Get) text formatting program named EZ, originally developed at Carnegie Mellon and championed by an Athena developer.

Unlike other WYSIWYG-like programs, EZ not only shows formatted text on the screen, it also automatically refreshes the text layout of a document to match the dimensions of a window. This makes EZ useful for arbitrary placement and sizing on a workstation screen. And EZ documents can also be converted to plain text on the fly, making it exceptionally useful for the tty version of OLH (e.g., dialup).

Because we in the Documentation group create most of our documents using EZ, all Athena users can have access to our official documents no matter what kind of system they are using. And our documents, in any case, retain a stylistic consistency.

ASSOCIATED HARDCOPY TOOLS

Along with the OLH browser program, we have also created several tools to help us automatically take OLH modules and assemble them into full documents. These tools are not yet fully integrated into the OLH system, but are worth noting to show how OLH can serve as the basis for a complete document-generation system, even though it is not itself a text-processing tool.

First of all, we maintain cover pages and copyright pages, as well as separate style template files, as separate modules that are not shown in the menu browser. When we need to create a document, we use special concatenation programs to collect a specified set of modules into one large file (we use the UNIX Make utility and another EZ program named datacat). We can then preview the resulting file online and note any small formatting problems (bad page breaks, etc.) We fix these problems in the large file, review it, then send it to a printer. And that's it: camera-ready copy.

If we find any factual errors, we fix the initial modules and remake the master file (the Make program assures that the latest version of the files are used; we don't have to check on this, we simply issue a command such as "make all").

THE FUTURE OF OLH

CURRENT PROBLEMS

The current version of OLH has one problem that needs to be fixed immediately: the way Keywords are implemented in the database makes it extraordinary slow to bring up the keyword list if there are many (> 100) keywords in the database. At Athena, we have simply not added new keywords, and know a solution, but we are waiting for development resources.

NEW FEATURES

OLH as it is currently implemented is fairly robust, but it does not yet include all of the features that were part of its original design. Some of the features were prototyped but left out of the current release for technical reasons. Others were never coded.

Among the features we plan to add when development resources become available are:

- a fully-featured interface to help automate the task of adding new modules/menus to the system
- automatic database generation/integrity-checking programs
- a "bookmark" capability, whereby users can "mark" modules they refer to frequently and jump directly to those modules
- a history capability, to keep track of where a user has visited in this OLH session to make it easy to return to a module already looked at
- a search capability in the built-in plain text displayer (some of the displayers already have this feature, OLH's built-in displayer does not yet)
- improved conversion scripts to support more documents in plain-text style

IMPROVEMENTS

In addition to adding new features to OLH, we would also like to improve the way OLH is currently implemented. Our original implementation included some concessions and inefficient code that could stand to be re-examined:

- The database routines were coded quickly and inefficiently. We already know many of the weak points and would like to rewrite these programs.
- The database routines were developed before a standard protocol for information retrieval calls had been developed. Such a protocol, Z39.50, is now emerging for library reference coding, and we would like to explore using that standard.
- MIT has separately developed a plain-text-only information program called TechInfo. This program offers interesting alternative ways of implementing our solution (e.g., a server-client model for the database). We would like to explore possible coordination with that program.

OLH OUTSIDE MIT

We first presented OLH to the world at large at an Athena Technical Conference in the Spring of 1991. At that time, we discovered that other sites than Athena were facing similar problems to ours as far as online documentation was concerned, and were interested in exploring OLH as a possible solution.

Because Project Athena ended as a funded research project in the Summer of 1991, resources are scarce at Athena right now to help make OLH ready for general distribution. Nevertheless, we are interested in sharing OLH with the world and hearing what other sites might like to see in OLH. OLH development has not been terminated, just slowed.

To make OLH available to other sites who have the technical know-how to adapt it to their systems, we have prepared a rough alpha-test version of the code. This version is available via anonymous ftp from a server at MIT. This version is not a generally usable version, but a test version.

The chief missing links in getting OLH to run at other sites are the Athena-specific assumptions built into parts of OLH (it only runs on UNIX systems, for example), and the lack of a interface for the document database maintainers.

We are hoping to evolve a more shippable version, and are glad to get input and resources toward that end.

CONCLUSION

As a documentation group trying to serve the user community of a large computer system that is both distributed and heterogeneous, we found that, although there was plenty of information on line, that information was not in a common format or standard location, nor was it like to be. Also, changes in policy were pushing us to move our own document set online, although we did not yet have a resource to handle this.

We decided that making online information available in an organized way was more important than assuring common formats or central locations. We also knew we needed a coherent way to deliver our own documents as small modules in an ordered hierarchy. Consequently, we designed and implemented a menu-based help browser, OLH, that allowed us to bring a wealth of online material to users through a single program while leaving the actual format and location of each document untouched.

In implementing this document browsing system, we were also able to incorporate special features such as support for plain text devices, and means for maintaining different versions of documents in a single source.

OLH is a powerful solution to an increasingly prevalent problem.

ACKNOWLEDGEMENTS

OLH owes its existence to lots of people. Too many people contributed to the design of OLH to be mentioned here. Among the programmers who contributed to OLH with actual coding were John Elsbree, Ian Boardman, Lucien Van Elsen, Chris VanHaren, Bill Cattey, Bruce Lewis, and Ezra Peisach.