33

Zephyr Extensibility in Small Workstation Oriented Computer Networks

by

Jason T. Hunter

Submitted to the Department of Electrical Engineering and Computer Science

in Partial Fulfillment of the Requirements for the Degrees of

Bachelor of Science in Computer Science and Engineering

and Master of Engineering in Electrical Engineering and Computer Science

at the Massachusetts Institute of Technology

May 27, 1997

Author [illegible]
Department of Electrical Engineering and Computer Science
May 27, 1997

Certified by [illegible]
Barbara Liskov
Supervisor

Accepted by [illegible]
Arthur C. Smith
Chairman, Department Committee on Graduate Theses

Zephyr Extensibility in Small Workstation
Oriented Computer Networks
by
Jason T. Hunter

Submitted to the
Department of Electrical Engineering and Computer Science

May 27, 1997

In Partial Fulfillment of the Requirements for the Degree of
Bachelor of Science in Computer Science and Engineering
and Master of Engineering in Electrical Engineering and Computer Science

ABSTRACT

ZephyrNT is an implementation of the Zephyr Notification Service for use on Windows NT and Windows 95. ZephyrNT provides a complete set of client-side applications for communication with existing zephyr servers. The software has been completely redesigned to take advantage of the specific features offered by the Windows operating systems. While providing a sophisticated interface, the software has been enhanced to support the transmission of multimedia messages including the display of messages written in RTF and HTML. The software illustrates the differences in interface design paradigms and control flow techniques used in developing Windows versus UNIX applications.

Thesis Supervisor: Barbara Liskov
Title: NEC Professor Of Software Science And Engineering

Contents

List of Figures

1. Introduction

One of the main benefits of the rapid proliferation of computers is the increased ability for groups of people to communicate, work, and interact in an efficient and organized manner. As networks grow larger and computers become a more fundamental part of everyday activities it is increasingly paramount that clear, easy to use tools for computer interaction be developed. Project Athena was designed to investigate how computer networks and distributed systems could be used effectively within a medium-to-large sized organization. In 1987 the Zephyr Notification Service (Zephyr) was designed to help meet these goals[1]. Zephyr was designed to be an instant, intelligent, notification routing system. The principal use of Zephyr is to send private or semi-private notices between individuals and groups of people.

The original design was well suited for the Athena operating environment of 1987 and, as any Athena user will attest, Zephyr has become a valued and useful component of the Project Athena services. However, the Internet needs a more flexible service. A service that supports enhanced content types and global scalability. The intent of this thesis is to describe an effort to expand Zephyr to achieve these goals. This project included the transfer and reorganization of the software to run on the Windows family of operating systems as well as an attempt to expand the protocol to handle a wider variety of content types.

This document is organized chronologically to allow the reader to understand the motivations that led to the end result. This document describes the work completed and provides a reference manual for users of the system. The rest of this chapter explains the conventions used in the rest of the document. Chapter 2 describes the state of the Zephyr system prior to this thesis work. Chapter 3 details the modifications to the client side interface made during the porting phase. Chapter 4 describes some initial client porting efforts to illustrate why the environment described in Chapter 5 was created. Chapter 5 gives a broad overview of the features and interface of the ZephyrNT environment. The end of Chapter 5 includes a discussion of the architecture underling the environment. Chapter 6 summarizes the efforts made to expand the protocol for greater flexibility in formatted content delivery. Finally, Chapter 7 provides some brief suggestions for future work.

1 DellaFera, C. Anthony, et. al. "Zephyr Notification Service," Project Athena Technical Plan, Section E.4.1. Massachusetts Institute of Technology, 1987.

1.1 Document Conventions

This section lists a number of terms and their definitions. The purpose of this section is to disambiguate several terms whose names are similar. In particular, the name "Windows" is used in many different contexts in this thesis. Furthermore, the name "Zephyr" is used in conjunction with almost every element of the system described here. The following list describes each such use along with the information necessary to distinguish them.

Windows	*This designation alone refers to the entire Microsoft Windows™ family of operating systems including Windows 3.1, Windows 95, and Windows NT. When a specific member of this family is intended its full name is used explicitly.*
X Windows	*Also known as the X Windows System. A network-based graphics window system that was developed at MIT in 1984.*[2] *Though this system can be run on Microsoft Windows™ they should not be confused. X Windows is primarily used on UNIX systems.*
GDI	*Graphics Device Interface is the window system for Microsoft Windows™*[3]*.*
Zephyr	*This designation alone or in conjunction with System refers to the entire Zephyr package including client side operations, server side operations, and second tier library functions that provide the communication between the other two layers. Work for this thesis concentrated on client side operations. No re-implementation of the server was attempted. Most zephyr components contain the word "zephyr" in their names and they should not be confused with the system as whole.*
Zephyr Server (zephyrd)	*The Zephyr Server or "the server" is the centralized component of the Zephyr System responsible for directing communications. All references to this component, except for those is Chapter 7 where differentiation is made explicit, refer to the UNIX based program provided by the original MIT implementation.*
Zephyr Host Manager (zhm)	*A client-side program responsible for concentrating server-bound communications and coordinating efficient retries.*
Zephyr Write (zwrite)	*The client-side program responsible for authoring messages.*
Zephyr Windowgram Client (zwgc)	*The client-side program responsible for receiving messages.*
Zephyr Library (libzephpyr)	*Second tier library that provides the API to the Zephyr System and hides the network layer communications. This library is used both by server-side and client-side operations. This library as implemented by MIT is a static library (libzephyr.a) and as implemented in this thesis is a dynamic-load library (libzephyr.dll). All references to the zephyr library, except where explicitly stated, refer to the DLL developed for use on Microsoft Windows as part of this thesis work.*

[2] For a more complete discussion of X Windows see *X Window System User's Guide*, O'Reilly & Associates Inc., 1993.

[3] For a complete discussion of GDI see *The Windows Programmer's Reference* and *The Windows Software Development Kit*, both by Microsoft Press.

ZephyrNT	*An integrated environment for Microsoft Windows NT and Windows 95 providing a complete set of client-side operations for the Zephyr System. ZephyrNT, in addition to the user-interface described in Chapter 5, refers to the Zephyr Library DLL and all other support libraries necessary for providing communication services through the Zephyr System from a computer running Windows NT or Windows 95.*

2. The Zephyr Notification Service

This chapter describes the state of the Zephyr Notification Service prior to this thesis work. Section 2.1 gives an overview of the basic zephyr components and their interactions. Section 2.2 describes the additional set of services provided by the MIT implementation. All of these additional services are built on top of the basic components given in 2.1. Their description is provided here to give the reader a familiarity with how Zephyr is currently being used.

2.1 The Basic Components

The Zephyr Notification Service was originally designed in 1987[4]. This service offers the end user the ability to send short text messages to other users who may be logged into the system.

A simple message notice is created by the end user through one of several different programs designed for this purpose. At the time of writing, four such programs exist (other than ZephyrNT):

- Zwrite is the original UNIX command line utility. This program takes a number of command line arguments indicating sending options and the intended recipient. The program then allows the user to type the message in successive lines of text.
- Xzwrite is a windowed version of the program mentioned above written for the X Windows system. This program also combines the abilities of the Zephyr Location Client (zlocate). This client and its features will be discussed the next section.
- Xzewd is a rewrite of xzwrite based on the Andrew Toolkit. Xzewd offers a slightly different keyboard interface from xzwrite as well as a more aesthetically pleasing visual appearance.
- MacZephyr: a combined port of several zephyr client programs for use on the Macintosh operating system.

4 DellaFera

Each program has approximately the same functionality; they all allow the user to type a relatively short message and then send that message to one or more intended recipients.

Messages are transported through the system via a communication block called a *notice.* A notice contains information such as the recipient's name, the sender's name, and the sender's location, in addition to the message text itself. During the course of transportation notices are broken up into one or more *packets,* each containing part of the notice's information. A packet is the primary unit of network transmission and represents little more than an abstraction of the network layer's user datagram packet (UDP). The packets are reassembled at the destination to reconstruct the original notice.

Figure 2-1 details the steps taken by the Zephyr System in communicating a message. Forward arrows represent the flow of the message from sender to receiver and backward arrows represent acknowledgements returned by various components at different stages of message delivery.

Sending a message hands the notice data structure over to the Zephyr System for processing followed by communication to the zephyr server (zephyrd). Though multiple zephyr servers may currently be in use, this resource duplication provides redundancy for efficiency only. The distinctions between single and multiple server environments is not material to this discussion and is mostly ignored in the rest of this document.

Processing of notices is handled by the zephyr library (libzephyr). The library provides an interface for managing notice delivery and receipt and is employed for this purpose by every component of the Zephyr System including zwrite, zwgc, zhm, and zephyrd. The next chapter details how the zephyr library was redesigned for the Windows platform; however, both the UNIX library and the new Windows library speak the same binary protocol across the network. For this reason, changes in internal management and processing on Windows machines are transparent to the server program running on a UNIX machine and utilizing the old zephyr library.

During the course of processing, the library breaks the notice down into one or more packets. Each packet is then sent to the Zephyr Host Manager (zhm). In Figure 2-1, this transfer and its acknowledgment is shown as steps 1 and 2. Zhm is responsible for communicating the packets to the zephyr server. Zhm handles all packet retries and other problems that may occur due to network instability. In addition, zhm can perform special processing on some notices. All communication between message authoring programs and the zephyr server are handled through zhm. Furthermore, zhm maintains a connection to a zephyr server at all times. If the zephyr server becomes unreachable for any reason, it is the responsibility of zhm to find another server. If no server is reachable, zhm will continue to retry packet transmission until a server does become available.

The zephyr server receives packets from zhm. This transfer and its acknowledgement are shown in Figure 2-1 as steps 3 and 6[5]. Each packet is decoded by the zephyr library into a partial notice. Each partial notice (and hence, each packet) must contain enough information to provide the server with recipient, sender, and authentication information in addition to a part of the text of the message itself. The server does not attempt to reconstruct whole messages at this time. Instead, the server checks the sender's authentication and authorization information against access control lists (ACL) that it maintains. If the sender's authentication information permits the transmission of this message to the intended recipients, the server then checks for recipient availability. If any of the intended recipients are currently available (that is, they are logged in and connected to the Zephyr System), the server routes a single copy of the message to each recipient's respective incoming notice client.

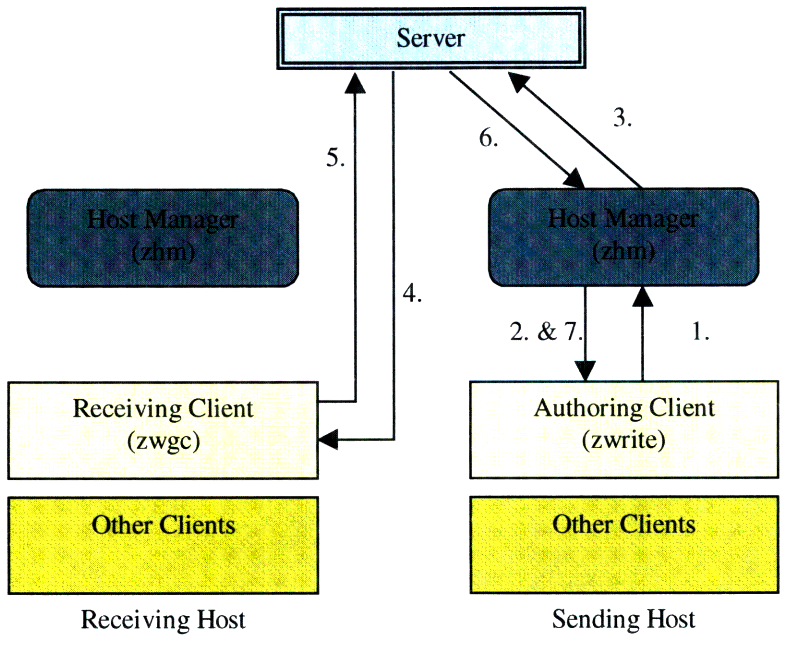


Figure 2-1: Zephyr System

[5] The server does not send an acknowledgement to zhm until the receiving client has acknowledged its receipt of the message. This is why step 6 follows step 5 in the diagram.

The incoming notice client normally referred to as zwgc (pronounced *zwig-see* – or Zephyr Windowgram Client) registers itself with the zephyr server upon startup. During the registration processes, zwgc indicates to the server which types of messages it would like to receive through the transmission of a set of subscriptions. Each subscription represents a notice type or set of types. In general, only these types of messages are forwarded by the server and others are discarded.

The zephyr library is responsible for receiving packets from the zephyr server. Client receipt and acknowledgment are shown in Figure 2-1 as steps 4 and 5. The library reconstructs an entire notice from a series of packets and hands that notice off to the receiving client when it is complete. Upon receipt of a complete message, zwgc creates a new window in which to display the message. Only one message is displayed per window. These windows are then made to appear in a configurable location on the receiving user's screen. When the user clicks on such a window, the message disappears. The notice is then destroyed. In the normal case, no record of this message is kept after the destruction of its associated window.

2.2 Other Zephyr Clients

In addition to the two basic clients that send and receive messages, a number of other clients have been built on top of the Zephyr architecture. Each of these provided some additional service though only one of them (the Zephyr Location Service) required additional server-side support.

2.2.1 The Zephyr Location Service

The Zephyr Location Service provides the additional functionality of user tracking. This service was added to the Zephyr package to promote communication between users. The service notifies users when other users, with whom they may wish to exchange messages, log in and out of the system. To support this service, the server was extended with a database containing the *locations* of each user registered with the service. A user can query this database to determine the current login status of a particular user. In addition, placing entries in this database or taking them out results in the generation of LOGIN/LOGOUT notices. These are notices generated by the server that may be subscribed to in the normal manner. Users who are interested in being continually advised of a another user's login status may subscribe to this type of message on a user by user basis.

The original UNIX system provides two programs for the management of LOGIN subscriptions and Location Database queries: zlocate and znol. Zlocate is a simple command line tool that allows users to query the Location Database for the status of a single user. Znol is also a command line utility but with broader functionality. Its primary purpose is to track the locations of a list of users. This list is kept in a disk file in the user's home directory. The znol program performs a database query for each username that appears in this disk file. In addition to displaying the results of these queries, znol attempts to locate a running windowgram client. If such a client is active, znol will request that the client subscribe to a message type corresponding to the LOGIN notices of each user whose username appears in the file. This results in the future receipt of LOGIN/LOGOUT notices, thus allowing the user to track the status of interesting parties.

2.2.2 Zaway, Zleave, Zpopnotify and Zmailnotify

These four clients are designed to notify users of some event that may be of interest to them. Zaway allows a user to specify a message indicating he is currently away from his desk. Notices that are received during the execution of zaway will automatically be responded to with this message. This client is intended only for short periods of absence. Longer periods should be dealt with simply by logging out of the Zephyr System.

Zleave is a simple timer client. This client sends a message to its own user at a specified time. This service could be implemented outside the scope of the Zephyr System. However, by taking advantage of the Zephyr System and the functionality exported by its library, zleave demonstrates how an event-based service can be very small, leveraging the Zephyr System for all of its display purposes.

Zpopnotify and zmailnotify are extensions to the email servers in the Athena environment. Zpopnotify queries the email servers to check whether new email has been received by the user. In the event that new email has been received, the zpopnotify program constructs a notice that is then sent to the client's own user to indicate this event. Zmailnotify takes this process one step further by including the new email's headers and a small portion of the body in the notice.

Each of these clients demonstrates how the Zephyr System can be used to communicate a variety of generic events to interested users. Furthermore, it shows how client programs can take advantage of the unobtrusive notice display paradigm inherent in notice delivery to convey information other than person to person messages.

2.2.3 Zinit and Attach

Like the programs above, zinit and attach are programs written on top of the Zephyr System. These two programs[6] subscribe to a series of messages of the type filsrv. These messages are generated by the file system services when certain types of loss of service will effect the user.

3. Zephyr for Windows

Work on this thesis began as an effort to bring an existing set of UNIX based services to the Windows 3.1 platform. It was soon clear that the Windows 3.1 platform was not a desirable end goal, and this portion of the project was altered. For reasons discussed below, Windows NT and its companion Windows 95 were chosen as the target platform.

With the idea of maintaining as much of the existing architecture as possible, the UNIX code structure was examined to assess its portability and to try to identify places where problems would result. The UNIX architecture is centered around a base library that offers a group of abstractions and services that are used by all other elements of the system. Obviously any porting task would begin with the translation of this library. However, although the original code did conform to most respected software design methodologies, it has become a little dated in its use of modern technologies. In particular, the abstractions that were provided were written in C and were mostly exported through a loosely coupled set of procedures. Modern object-oriented techniques had not been used. Because there were differences in the way the operating systems provided their system services, and differences in expectation of programmers in each environment, I decided that a redesign of the library to take advantage of more recent technologies in compiler support and object-oriented management would be beneficial to the long-term future of the new code. Therefore, I decided to rewrite the code from scratch while maintaining as similar an interface as possible, in order to simplify the porting of the other tools in the Zephyr package.

The remainder of this chapter discusses the design goals of the porting effort. Section 3.1 gives a brief overview of the Windows operating systems and suggests the primary motivations for making the alterations described in the rest of the chapter. Sections 3.2 through 3.5 each identify a particular technology whose absence was noted from the original MIT implementation. Each section explains the desirability of each technology and outlines how the Windows implementation attempts to take advantage of it. Most of the alterations

[6] Actually these are really the same program which has different behavior depending on how it is called.

described in this section relate only to the implementation of the zephyr library (libzephyr.dll). Specific design issues concerning the ZephyrNT environment are left to Chapter 5.

3.1 The Windows Family of Operating Systems

The Windows family of operating systems, developed by Microsoft Corporation, first appeared in 1985. The original goal of these systems was to allow users of low power, low cost computers (personal computers) running the MS-DOS Operating System to multitask several applications and thereby gain additional efficiency and usability. Rapid developments in inexpensive microprocessor manufacturing brought incredible productivity gains to companies and individuals using Windows over those using MS-DOS alone. However, it wasn't until the introduction of Windows 3.0 (and Windows 3.1 shortly there after) in 1990 that Windows began to take hold in the corporate marketplace. The success of Windows 3.1 drew attention to Microsoft's style of user interface, despite the obvious instability of the Windows architecture. The coupling of the older MS-DOS Operating System with Windows 3.1 seriously limited the expansive functionality of the resulting united environment. Although improvements had been made at the processor level for increased security, speed, and protection, Windows 3.1 was not able to take full advantage of these features.

From this was born Windows NT. Windows NT is an instance of the Mach micro-kernel and is similar architecturally in design to the NeXT Operating System. Windows NT was Microsoft's attempt to write a stable, mission critical operating system[7]. Windows NT provides support for the traditional set of services that have become synonymous with preemptive multi-tasking operating systems. This set includes networking, virtual memory, security, memory protection, multi-user support, and a windowing system. Unlike many of its UNIX ancestors, Windows NT has taken a different approach to a number of these traditional services. For instance, Windows NT includes support at the kernel level for multiple thread programming. In fact, the thread is the base unit for Windows NT's scheduler. Furthermore, Windows NT stresses the paradigm of a single user logged in at one time. The system does recognize multiple users who may log in at different times but only one of these users is expected to be logged in at any given time. There are several reasons for this choice but the one that seem to have been most influential is the design of the windowing system.

[7] Incidentally, this was not Microsoft's first attempt to write a robust operating system. Microsoft had previously collaborated with IBM on the creation of OS2. Development and marketing concerns forced the

The windowing system provided by Windows NT is a direct extension of that provided by Windows 3.1. This system, called GDI for Graphics Device Interface, encapsulates the available hardware behind a rich Application Programmer Interface (API). This interface is responsible for allowing the Windows family of operating system to run on a broad range of different hardware without the need for operating system customization or application porting. In fact, the Windows operating systems can run with a wider variety of display hardware and input/output software than any other operating system on the market today. Despite GDI's advantages it has the disadvantage that its windows cannot be displayed on a machine other than the one in which the application is running. Furthermore, since Windows NT does not encourage the creation of console-based applications, almost all Windows NT programs are window-based. Other users logged in simultaneously have no way of displaying or interacting with windowed programs, and are therefore severely limited in available services. This differs from UNIX systems, which are typically configured with X Windows, a networked display technology. Furthermore, many more UNIX programs may be run from the command line and are therefore available to users logged into the system remotely.

Because of the windowing constraints, Windows NT machines are, typically single-user machines. This has had a serious impact on the design of software for this system. As mentioned before, most programs are window-based and a heavy emphasis is placed on making effective use of the graphics display as a means to convey and control information rather than through more traditional methods common in UNIX systems. For instance, Windows programs generally require a lot of mouse clicking where the equivalent actions in a typical UNIX program would require a series of keys to be typed on the keyboard. The benefits of this design have been a dramatic decrease in the learning curve for new users but the downside is the loss of efficiency for more experienced users. A well-designed Windows program needs to find a compromise between an easily manageable mouse interface and a speedy set of keyboard accelerators.

A final point to be made about Windows NT concerns the programming environment available to designers. This environment, as mentioned before, offers all of the typical services that have become common in multitasking operating system, most notably UNIX. Furthermore, the API's for controlling these services are similar enough to their UNIX counterparts that cross-platform development is possible. However, the design strategy of typical Windows NT applications, as discussed above, make such development frequently

two companies to sever their association. Some of the original Windows NT code was taken from this undesirable or impractical. For this reason, a well designed port of a UNIX program (for instance, the Zephyr Notification Service) will require some architectural changes to best take advantage of the way Windows NT offers up its services. In addition, it will require redesign of some interface issues to better meet the expectations of typical users of the Windows family of operating systems.

3.2 Class Libraries

The new library was designed in C++. Utilizing an existing C++ class library would spend code creation and ease code maintenance However, there were several available depending on which compiler you chose. Furthermore, many of the library could not be carried from compiler to compiler and using them would make the code dependant on a particular compiler vendor. I examined offerings from Borland, Symantec, and Microsoft as well as Win32 ports of the standard GNU libraries from gcc. All of the class libraries provided approximately the same functionality, but each came with different tools for managing projects, doing revision control, debugging and interface design. In the end I decided to go with the Microsoft compiler because it provided a better set of tools. Microsoft's compiler suite came with two different sets of class libraries: the Microsoft Foundation Classes (MFC) and an implementation of the Standard Template Library (STL). Though I had some reservations about marrying this new project to a library that was difficult to take away from the its compiler, I ultimately decided to base the code on MFC. MFC provided a very rich set of tools for doing interface design, and I hoped to be able to take advantage of these features later when constructing the client code. Furthermore, there were a number of third-party tools that I thought might be useful. Implementing the library with MFC would facilitate its use by MFC client code.

Each abstraction in the C code was replaced by a corresponding class in the new library. The following table shows a summary of the code migration.

C Library Function	**C++ Encapsulation Class**
ZNotice_t, ZNotice_Kind_t, ZSendNotice, ZSendList, ZMakeAscii, ZCompareUIDPred, ZCompareMultiUIDPred, ZFreeNotice, ZIfNotice, ZCompareUID, ZReceiveNotice, ZReadAscii, ZCheckAuthentication, ZFormatNotice, ZFormatNoticeList, ZFormatRawNotice, ZFormatSmallRawNotice,	*CNotice*

earlier attempt.

ZFormatRawNoticeList, ZFormatSmallRawNoticeList, ZSendRawNotice, ZSendRawList, ZFormatAuthenticNotice	
ZPending, ZQLength, ZPeekNotice, ZPeekIfNotice	*CNoticeQueue*
ZSendPacket, ZReceivePacket, ZParseNotice	*CPacket*
ZInitialize, ZOpenPort, ZClosePort, ZGetSender, ZGetRealm, ZGetWGPort, ZSetFD, ZGetFD, ZGetDestAddr, ZSetDestAddr, ZSetServerState	*CZephyrLib*
ZCancelSubscriptions, ZSubscribeTo, ZUnsubscribeTo, ZRetrieveSubscriptions, ZRetrieveDefaultSubscriptions, ZGetSubscriptions, ZFlushSubscriptions	*CSubscriptionList, CSubscription*
ZLocateUser, ZNewLocateUser, ZGetLocations, ZFlushLocations, ZSetLocation, ZUnsetLocation, ZFlushMyLocations	*CLocationList, CLocation*

Figure 3-1: Porting Summary

3.3 Exceptions

Exception-based error handling was strongly motivated by a desire to simplify the interface of the new class methods. All of the original C functions return error codes requiring all return-values to be passed as out-parameters. This made much of the code clumsy. Since the current project was going to require redesign, it seemed like a good time to put the extra effort in to adding exception support. I designed a small hierarchy of exception classes to handle this situation:

The leading goal of this design was to allow clients to filter out only the exceptions they wished to handle, while allowing enough higher-order structure that general exception handlers for classes of exceptions could be written. The library's design attempted to take advantage of C++'s normally unfortunate ability to propagate uncaught exceptions to a higher level. This propagation was designed into the library, allowing uncaught exceptions to be the expected way that a error would work its way up multiple levels of library calls to return to the user code where it would be appropriately handled. Care was taken that this unwrapping process would correctly clean up resources along the way, leaving the user code in a predictable state. Return codes were almost completely eliminated from the library routines, though the actual error values themselves were maintained in the exception class definitions. Conversion methods were supplied by the library exception classes to allow translation between the old-style return codes and the new exception classes.

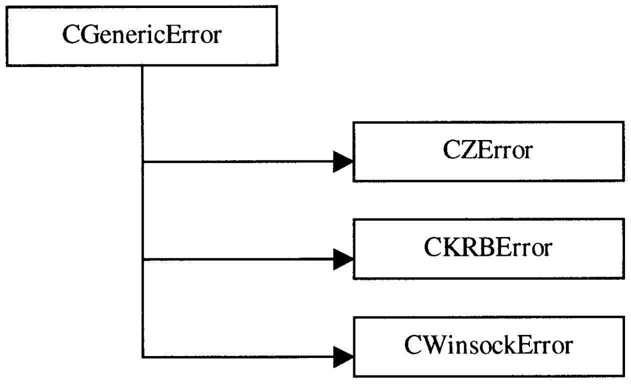


Figure 3-2: Exception Hierarchy

3.4 Multithreading and Shared Libraries

In addition to allowing exceptions, I wished to take advantage of three other system services. The first of these is multithreading. The original architecture intended that each of the tools in the system be run as a separate process. Although this is a typical methodology in a UNIX environment, it is not the most efficient on the target platform. Because Windows NT performs scheduling on a thread basis, it is common for NT programs to be multithreaded. In anticipation of this potential use, the resultant library was designed to be multithread compliant. In particular, each thread has independent access to the full set of library services without requiring knowledge of other threads' usage. Meeting this constraint required two main alterations. First, all state that is observable across library calls must be kept in thread-specific storage. This required some reorganization of library wide data that was formerly stored in the global variables. Because this data had been gathered together in a new abstraction designed to represent the current state of the library, it was not difficult to allocate a separate instance of this object for each thread accessing the library. Second, all calls must be reentrant. This required the removal of local static variables. In several operations, these statics were used to implement a blocking-call that supported timeout behavior. This functionality was obtained through message-based control flow instead.

In addition to multiple threads within a single process accessing the library, multiple processes would also likely be used. The original library was a statically linked object. Every program taking advantage of the library received a separate copy of the library's code. Like many UNIX systems, Windows NT supports shared libraries. More efficient use of the library code was obtained through such a shared library. Because NT's support for shared libraries is extensive, supporting such a library required only the creation of a process management function.

3.5 Message Handling

Windows NT provides many of its services, especially those that are asynchronous, through messaging. In particular, socket management is supported through synchronous, callback, and message-based interfaces. The original code did some control flow management through the use of signals and signal handlers. Although signals are supported in NT to ease porting of UNIX-based applications, the message-based versions are preferred and generally provide greater efficiency. All such uses of signals were converted to messaging as well as the implementation of timeouts and other blocking call features. Handling blocking calls in this way was particularly important for a windowed application under the target OS, because waiting on a message does not actually block the thread even though it may block the currently executing message handler. When a handler is called by the message queue it generally takes control of thread processing; however, if the handler cannot complete processing of its message, it may surrender control of thread temporarily to the message queue and allow another handler to execute while it is waiting. This allows painting of window updates during blocking calls. Of course, care must be taken when accessing data in the painting code that may be in use by a blocked message handler. Problems in this area were avoided with mutexes or additional messages for transferring data between handlers that must be synchronized.

As a side note, closing down applications in Windows is also handled through messaging. Because the zephyr library spends most of its time waiting for incoming messages from the network, the process is normally inside a blocking call waiting for messages. The user would not be able to shut the program down until a message came in. In UNIX, blocking calls are usually handled by polling the library for messages rather than allowing it to execute a blocking call. However, this complicates the library's interface. To keep the library's interface as simple as a blocking call, it is necessary that some messages, like WM_CLOSE and WM_PAINT, be handled even when the process is essentially blocked. Allowing the library to route these messages despite the fact that it is in a blocking call allows client programs to be greatly simplified. A simple user program can implement handlers for the above messages and then make a single blocking call to the library. The message handlers will be called in response to user-interface events despite the fact that the process should be stopped.

4. Initial Porting Effort

Coding the newly designed zephyr library began in January of 1996 and was complete by the beginning of February. Upon completion, focus in this project was turned to the translation of Zephyr clients to the new platform and new library. These clients would hopefully provide both useful functionality and the means for a more robust test bed for the new library. First efforts were made to translate *zwrite,* the UNIX command line Zephyr authoring client.

Zwrite, as discussed above, requires an active *zephyr host manager (zhm)* process running on the same machine in order to function properly. This presents a design dilemma. *Zhm* is a far more complicated client then *zwrite*. In addition, *zwrite* directly accesses the critical path of the zephyr library without straying too far into less important functionality. *Zwrite*, therefore, seemed a much better candidate for initial implementation. However, developing *zwrite* first would require avoiding *zwrite's* dependence on *zhm*. To get around this dependence, a modified version of the UNIX *zhm* was created from the original UNIX source. Unlike the distribution release, this modified version of *zhm* accepted connections from clients running on any machine. The initial port of *zwrite* was written to look for a *zhm* instance on a local UNIX debugging machine rather than its own machine.

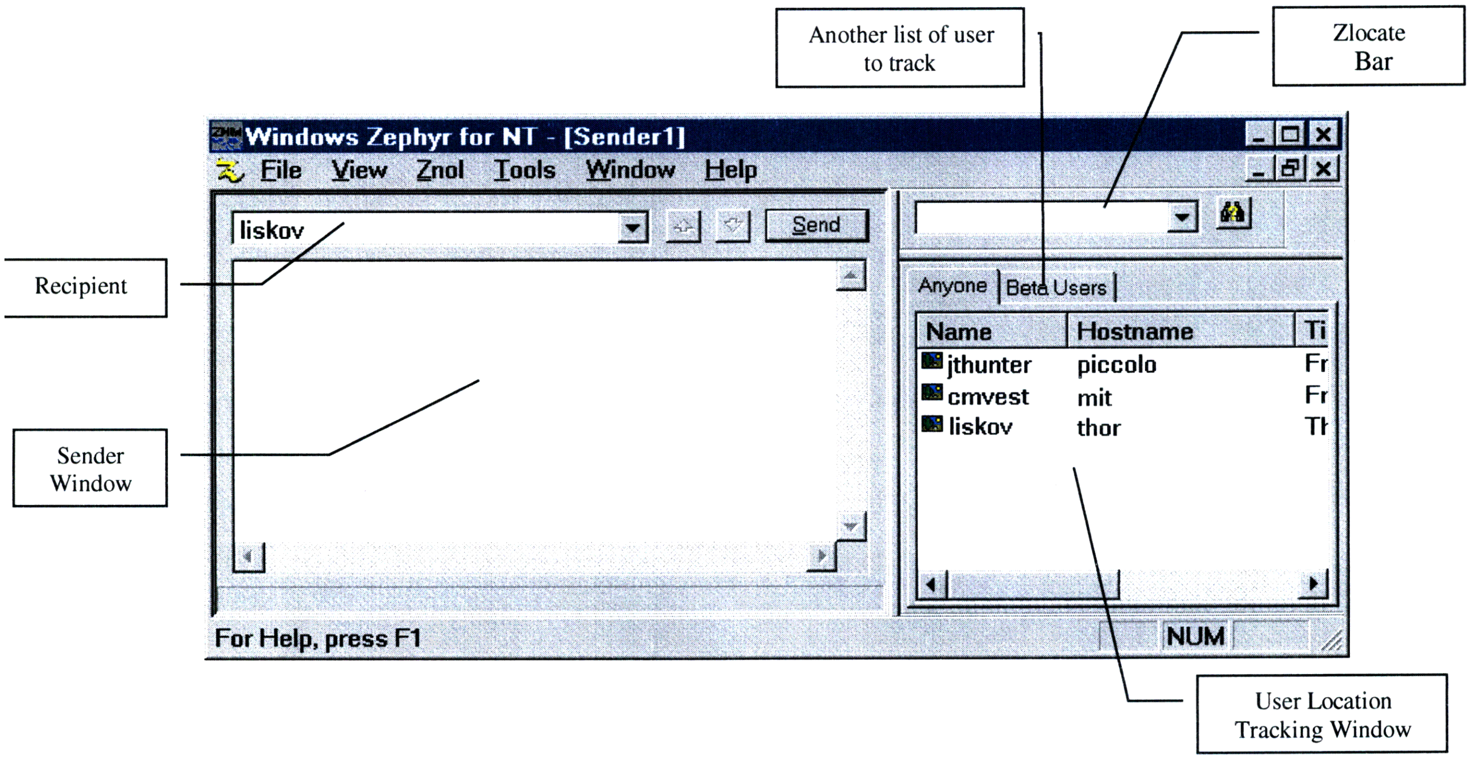
Utilizing the same stub *zhm,* an implementation of a simple receiving client was done. This client had no main window. However, it did provide very simplistic popup window service. No direct port of the *zwgc* client was ever attempted. The notice receiving interface of the new library is slightly different than the old. Furthermore, there was no advantage to directly porting the X Windows based display code and the formatting and parsing capabilities of *zwgc* were likely to be removed (see Section 6.1). Therefore, receiving and display clients for Windows were to be designed from scratch. The client described here was never intended to be more than a testing aid for the zephyr library, however, its creation and that of the *zhm* port that followed began to bring up interesting questions about how Windows users would react to this multi-program environment.

Zhm porting had begun by mid-February and was complete by mid-March. The *zhm* port was a optionally windowed program that could be started either as a foreground or background process. The main window (appearing when started in debug mode) allowed the operator to monitor the activity of *zhm*. Summary information about each message handled by the client was displayed in the window. The original debugging version bears strong resemblance to its counterpart in the ZephyrNT product. See Figure 5-19 and its associated discussion for a complete description of its features.

With the completion of a working *zhm* client, it became possible to use the Windows NT zephyr system standalone. Initial use testing and studies indicated that the separate component model was not going to work very well in the Windows environment. Though support is provided for starting programs when a user logs in initially, starting each of the three clients proved to be a nuisance. Common usage in the initial test group showed that many users wished to use the Windows based package over dialup lines. This made it less desirable to have the three programs running all of the time from login to logout. It is far more desirable to bring the zephyr system on line after a dialup connection is made and shut it down prior to breaking that connection. Furthermore, Windows does not provide well-known services for shutting down background processes. This makes it difficult for less advanced users to bring up and shut down the *zhm* process. Windows users are accustomed to a single main-window style interface to programs even if these programs actually are a system of separate processes. Good examples of this include Visual C++, Microsoft Word, and Netscape Navigator. All of these programs are actually several concurrent processes that interact with the user through a single windowed space. These constraints led to the design of the ZephyrNT environment described in the next chapter. This environment takes advantage of the enhanced capabilities of the Windows zephyr library to present a single windowed environment for managing the Windows zephyr package. The package handles management of all clients, *zhm* initialization, and authentication.

5. ZephyrNT

The ZephyrNT client package described in this chapter and the chapter that follows has been fully implemented as detailed in this document. The software is currently in use by a beta test group of approximately one hundred and fifty students and faculty at MIT. ZephyrNT has been available to the MIT community for about a year and half as of the writing this document.


Another list of user to track
Zlocate Bar
Recipient
Sender Window
User Location Tracking Window
Windows Zephyr for NT - [Sender1]
File View Znol Tools Window Help
liskov
Send
Anyone
Beta Users
For Help, press F1
NUM

Name	Hostname	Ti
jthunter	piccolo	Fr
cmvest	mit	Fr
liskov	thor	Th

Figure 5-1: Main Window

The following sections describe the actual design of the ZephyrNT client package. Sections 5.1 and 5.2 give a high level description of the program's features and user interface. Section 5.3 elaborates on the internal structure of the client. This last section basically explains how the client works.

5.1 Single integrated environment

The ZephyrNT environment combines all of the client-side tools in a single user program. This section and the one that follows will outline the how each component of the visual interface implements each of the original zephyr clients. ZephyrNT implements in some form *zwrite*, *zwgc*, *zlocate*, *znol* and *zhm* as well as a number of additional features that have been added to the UNIX tool set by *zwgc* dotfile modifications. Furthermore, the best features from hybrid user tools such as *Emacs-Zwgc*, *Xzewd* and *MacZephyr* were incorporated into the design where appropriate.

5.1.1 The Sender Window

Figure 5-1 shows an image of the main ZephyrNT window in a common configuration. Several of the main components can be seen in this picture. The left side of the image contains the sender window (shown here as a maximized MDI window). This window implements most of the functionality of Zwrite. The edit box of this window is used for authoring, editing and reviewing messages. The recipient of a message is indicated by placing that persons username in the combo box above the edit window. The username *liskov* appears there in the image. Messages can be sent to *instances* or *classes* by prepending a *–i* or *–c* respectively to the beginning of the name. For example, a message could be sent to the *help instance* by typing "-i help" in the combo box. The combo box remembers a history of the recipients to whom you have sent messages during this session. In addition to its drop down functionality (indicated by the arrow to the right of the combo) partial username completion is provided via the down-arrow key. Completion names are taken only from those to whom you have sent messages during this session and the list is not maintained across sessions. Future releases should allow history lists to be maintained across sessions and to correlate with the location client windows that are currently open in the environment. This is discussed further where location clients are discussed below.

Once a message has been completed, it can be sent by clicking on the **Send** button. The sender window also maintains a history of the messages that have recently been sent. To scroll through the history of sent messages, click on the **Previous** and **Next** buttons (these are the arrow icon buttons above and to right of the edit window). Messages are appended to this list whenever the send button is clicked. Currently no attempt is made to filter out duplicates. As with the username history list, this list is not maintained across sessions. Furthermore, it is possible to open several sender windows at one time, allowing multiple messages to be authored and sent without mutual disruption. However, the history list of each sender window is kept distinct. Future releases should allow the option of a single merged history list for names and/or messages.

The following table shows the accelerator mappings and the send menu. The **Show as Toolbar** menu option is discussed later in Section 5.2.3. All of the button functionality for this window is also provided via the Send Menu and through a set of keyboard accelerators.

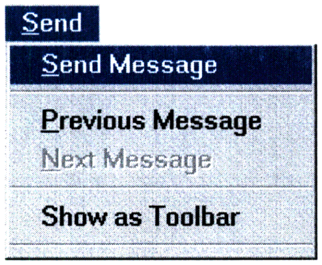


Figure 5-2: Send Menu

Action	Key
Send Message	Ctrl-Enter, Alt-S
Previous Message	Ctrl-Up-Arrow
Next Message	Ctrl-Down-Arrow
Username Completion	Down-Arrow (in combo box only)

Figure 5-3: Send Accelerators

5.1.2 The Location Client Window

Two location clients are pictured in Figure 5-1 (both appear as mounted toolbars). In the image, the location clients (or znol windows) take up most of the right side of the window. A location client window implements much of the functionality of the *znol* utility. Each client maintains a list of usernames (replacing the disk-file normally named *.anyone*). The window continually displays the login status of the users in the list. The list of names appearing in the location window indicate that each of those users is currently *logged into the zephyr system*. Absence of a username does not imply that the user is not logged in to a machine, but merely that the user does not currently wish to receive messages. (Actually, the user exposure also affects their appearance in this window and more is said about this below.)

If a user is currently logged in, and his username appears in the list of users to track, the user is listed in the location window along with the current machine he is logged into, the time he first connected to the system, and a display string. On UNIX systems, the **display** string actually represents the terminal session of the user; however, Windows NT maintains only a single terminal session for the desktop user, so this field is not needed. As described at the end of Section 5.1.4, ZephyrNT will allow you to configure this string. For ZephyrNT users, this string provides a convenient place to give additional information about the location of the machine that on which ZephyrNT is running. If unset, the string defaults to "ZephyrNT." For example, this could be set to "In Office" or "At Home" to indicate to others that the network is being accessed from the office or a remote location.

Another difference between the UNIX interface and ZephyrNT is the use of the **hostname** field. The UNIX implementation of this field provides the fully qualified DNS name for the user's machine. Machines are named slightly differently in the Windows world, using NT Domain names followed by sub-Domain names and then machine names. For example, the machine in my office has the DNS name: piccolo.lcs.mit.edu. Its WINS (Windows Internet Name Service) name is: PMG\piccolo. In a single domain environment ZephyrNT places the machine name (piccolo in this case) into the hostname field. In a multi-domain environment ZephyrNT would fill this field with the full WINS name (PMG\piccolo). However, it never places the DNS name (piccolo.lcs.mit.edu)in this field. This causes some confusion to those who are familiar with the UNIX environment generating the question: "Why does ZephyrNT not display the *.lcs.mit.edu* part of the hostname?" The reason for this alteration is not simply aesthetic. Windows NT provides significant network browsing and file sharing capabilities. All of these features make use of the WINS naming scheme for locating, viewing, and connecting to machine on the network. Providing the WINS name in this field better allows users of ZephyrNT users to interact with other Windows NT machines.

Currently ZephyrNT does place usernames in the **name** field. However, it is capable of placing the full name of the user as listed in their Domain User Profile at the local Windows NT server. This functionality is currently disabled because the overwhelming usage in the MIT environment will include listing, almost entirely, only Athena registered users rather than users from a local Windows NT network. Furthermore, usernames are displayed in the short format (jthunter) as opposed to a fully qualified Kerberos Principal (jthunter@ATHENA.MIT.EDU) for simplicity. Windows NT Profile Names are specified using the WINS format, E.g. PMG\jthunter is my current login name. In a multi-domain environment it would be necessary for ZephyrNT to display either a user's full name (Jason T. Hunter) and/or a fully qualified username (PMG\jthunter). This additional functionality will appear in later releases.

Double clicking on a name listed in a location client causes the client to search for an existing Sender Window. The username of the click line is placed in the recipient box and the cursor is placed in the message edit box for authoring and transmission. If no sender window is open, a new one is created. If multiple sender windows exist, the first one not currently transmitting a message is chosen. Each location client window maintains a separate list of users to track. Tracking of each list is completely independent, which allows users of the system to keep separate lists of users for different purposes. For example, the image of Figure 5-1 shows two location clients. One of these clients is named *Anyone* and contains a list of users that I frequently talk to. The other is named *Beta Users* and tracks the list of users currently using the ZephyrNT beta product. Each location client is configured through the Znol Menu pictured below:

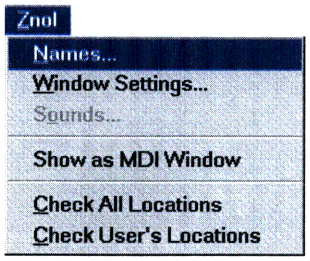
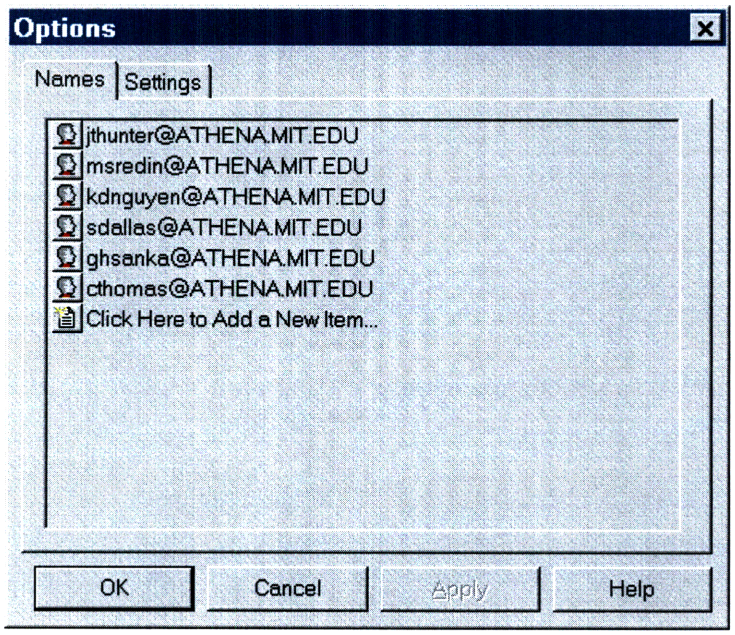


Figure 5-4: Znol Menu

Each command in the first group brings up the configuration property page for this location client window. This property page, pictured in Figure 5-5 and Figure 5-6, allows the user to set various parameters of the client. The name list can contain any number of names.

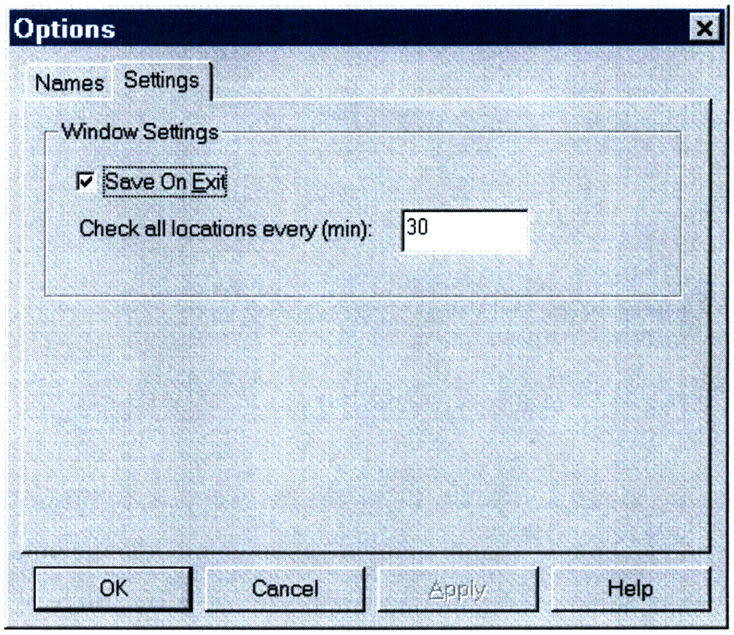
The domain name extension need not be entered and will be acquired from the zephyr library for the currently connected zephyr server. The Settings page allows various window settings to be configured. This page appears on the property sheets of location clients, windowgram clients, and zhm clients. The parameter **Save on Exit** is a reoccurring parameter for all window types. If it is checked it indicates that all changes to this client should be saved at exit time without requesting further confirmation from the user. If this option is not checked the user will be queried to save the changes for each client that has been modified (and not saved) during this session. If a large number of clients are used frequently this can make shutting ZephyrNT down slow and cumbersome. This automatic save option works well in conjunction with the environment restoration feature of the ZephyrNT as discussed later in this chapter.

The **Check all locations every:** property specifies how frequently this location client should recheck the status of all the usernames in the list. This field sometimes confuses users who are familiar with *znol*. This field does not indicate how frequently the tracking list is updated. The tracking list is updated continually. Each location client appropriately subscribes to LOGIN/LOGOUT messages and interprets their meaning to keep the list update to date. However, on occasion these messages are lost by the zephyr servers or a user logs out or shuts down improperly and no message is sent. These types of failures can lead to the list becoming inaccurate over a long period of time. This option allows the user to specify a time period in which to automatically refresh the list to avoid these disturbances. Values ranging from one to five hours (60-300 minutes) have proven to work quite well. Values shorter than one hour (such as the 30 minutes shown in Figure 5-6) do not show a visible improvement in tracking accuracy. Users are encouraged to play around with these values to find the best fit for their zephyr servers and network environment. However, very short values (like 1 or 2 minutes) should be avoided as they can generate useless network traffic and overload the zephyr servers with location database queries.



jthunter@ATHENA.MIT.EDU
msredin@ATHENA.MIT.EDU
kdnguyen@ATHENA.MIT.EDU
sdallas@ATHENA.MIT.EDU
ghsanka@ATHENA.MIT.EDU
cthomas@ATHENA.MIT.EDU
Click Here to Add a New Item...

Figure 5-5: Name Property Page



Window Settings

☑ Save On Exit

Check all locations every (min): 30

Figure 5-6: Settings Property Page

5.1.3 The Zlocate Bar

The Zlocate Bar is a toolbar that provides the functionality of the *zlocate* program. That is, it allows users to query the location database for the current location of a single user. This feature comes in handy when you need to locate or find the login status of a user that you normally don't track in a location client. The Zlocate Bar appears in Figure 5-1 in the top right corner as a mounted toolbar and again in Figure 5-7 as a floating toolbar. The toolbar's visibility is toggled by the **Zlocate Bar** command on the View Menu. In addition to the toolbar, zlocating can also be done by the **Locate User...** command on the Tools Menu. This command brings up a small dialog box in which to type the name of the user to query for. The history list maintained by the Zlocate Bar is merged with commands given to the menu command for convenience.

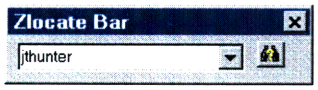
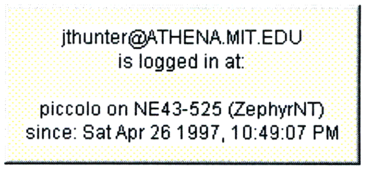


Figure 5-7: Zlocate Bar



jthunter@ATHENA.MIT.EDU
is logged in at:

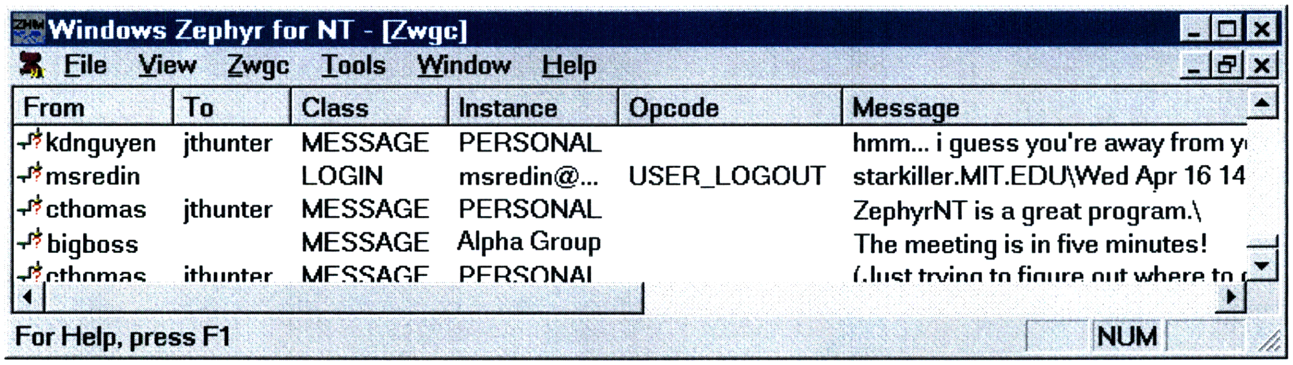
piccolo on NE43-525 (ZephyrNT)
since: Sat Apr 26 1997, 10:49:07 PM

Figure 5-8: Zlocate Tooltip

Regardless of the manner in which a zlocate query is issued, the results are given in a *tooltip window* that appears at the cursor location when the information becomes available. Clicking anywhere outside the tooltip or hitting a key clears the tooltip window. An example of a tooltip window is given in Figure 5-8.

5.1.4 The Windowgram Client Window

The Zephyr Windowgram Client, or Zwgc, is perhaps the most visible aspect of the package. This client is responsible for the receipt and display of messages. The Windowgram Client (pictured in Figure 5-9) provides the basis of this functionality. Each Zwgc window keeps track of a list of subscriptions that specify the class of messages that it expects to receive from the server. When messages come in from the server, the Zwgc window is responsible for deciding how to display that information to the user. The default behavior of the UNIX client is to create a new popup window and display the message as its content. As is explained more fully below, ZephyrNT provides a number of alternatives to this style of display. Nevertheless, popup windows are expected to remain the most popular method of display. Zwgc popups are similar to the tooltip windows described earlier. However, the user must click within the message to cause the window to be cleared.


Windows Zephyr for NT - [Zwgc]
File View Zwgc Tools Window Help
For Help, press F1
NUM

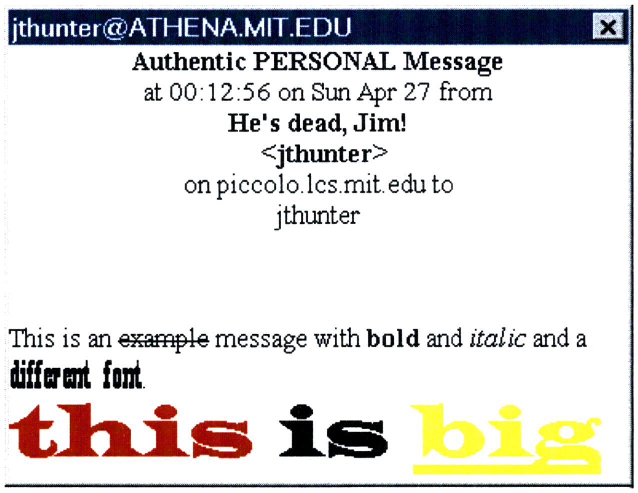
From	To	Class	Instance	Opcode	Message
kdnguyen	jthunter	MESSAGE	PERSONAL		hmm... i guess you're away from y
msredin		LOGIN	msredin@...	USER_LOGOUT	starkiller.MIT.EDU\Wed Apr 16 14
cthomas	jthunter	MESSAGE	PERSONAL		ZephyrNT is a great program.\
bigboss		MESSAGE	Alpha Group		The meeting is in five minutes!
cthomas	jthunter	MESSAGE	PERSONAL		(Just trying to figure out where to

Figure 5-9: Zwgc Window

Figure 5-9 shows a Zwgc client with all of its columns visible. The client is capable of logging messages, as seen here, or displaying messages as popups, or both. Each line of the above log shows the information relating to one message. The **From** column lists the sender of the message. The **To** column lists the intended recipient. If the message is sent to a group of people (via a class or instance), the recipient field is left blank. The **Message** column contains the first few lines of the message. The **Class** and **Instance** columns indicate the subscription (type) that the message belongs to. The **Opcode** field is an extra field that is used by some special purpose messages. In the above example a LOGIN/LOGIN message is displayed and the opcode field indicates the person has logged out.

The default format of a Zwgc client display shows only the **From**, **Instance**, and **Message** columns, because these are the most interesting. The others can be viewed by stretching the column widths until they appear. The log line cannot display an entire message if it is longer than a handful of words. To read long messages from the log line double click on the line of interest. A full popup window will be displayed for the line clicked.

Logs are saved from session to session and can be used to keep a complete or partial record of messages received. How to configure the size of the log is described below.



Authentic PERSONAL Message
at 00:12:56 on Sun Apr 27 from
He's dead, Jim!
<jthunter>
on piccolo.lcs.mit.edu to
jthunter

This is an ~~example~~ message with **bold** and *italic* and a different font
this is big

Figure 5-10: Message Popup

Popup windows for messages contain a different set of fields. Both contain the name of the sender, the recipient, and the message, however, the popup also contains the authentication status of the message, the time the message was sent, the signature of the sender, and the machine from which the message was sent. The message in Figure 5-10 was sent from jthunter to jthunter from the machine piccolo.lcs.mit.edu on Sunday, April 27 at 12:46 AM. The sender's identity was authenticated. The signature "He's dead, Jim!" is displayed before the username. The signature field was originally intended to hold the complete name of the individual sending a message, however, it is more frequently used to hold small quotes or other messages that are unique to the individual sender. Signatures are frequently humorous and many user modifications have sought to expand Zephyr's use of them. More will be said about the features of signatures adopted by ZephyrNT in the next section.

The rest of the text appearing in the popup above is the text of the message. Most frequently, messages are just short pieces of text. However, the Zephyr system in its original form does provide some capability for formatting messages. ZephyrNT extended this functionality in a number of ways. How this was extended is the main topic of Chapter 6. The message shown here is an RTF message showing how ZephyrNT can be made to send and display messages with a variety of text sizes, formats, fonts, and colors.

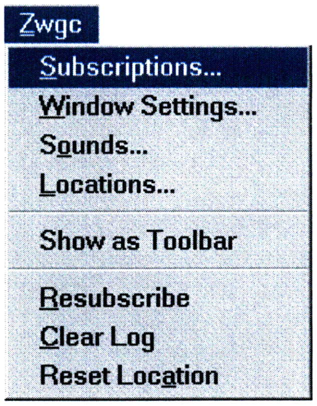


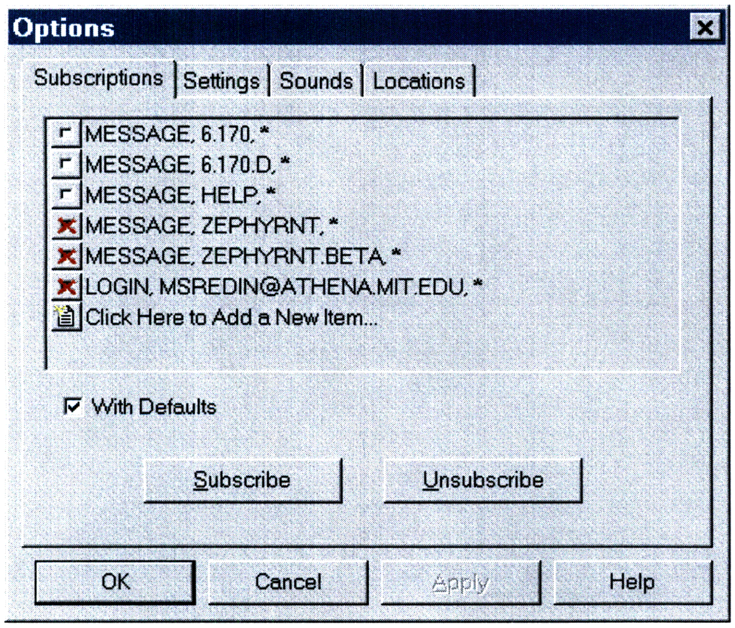
Figure 5-11: Zwgc Menu

The Zwgc Menu, like the other menus seen, allows the user to configure the behavior of the client. The first four commands of the menu access the client's property sheet. The last three resynchronize the client with the zephyr server. The **Resubscribe** command causes the client to cancel all of its subscriptions and then resubscribe. This ensures that the system will send the client only the messages that the client expects. The **Reset Location** command performs the analogous operation for the location maintained by this client. (More about locations below.) The **Clear Log** command does what it implies, it clears the current log of all messages. Note that clearing the log of one client does not effect the logs of other clients. Once a message is cleared from a log, it is lost forever.

Each command in the property sheet section of the menu corresponds to a page of the property sheet. Actually, all pages of the sheet can be reached by any of the four commands. The redundant commands are provided for convenience. The **Subscriptions** page (shown in Figure 5-12) maintains the list of subscriptions. Each line in the list represents one type of message this client is to receive. The list can also maintain subscriptions to which this client is not currently subscribed. For instance, the first three subscriptions in the figure are not being used by this client. It is convenient to being able to maintain subscriptions in the list that may be used at certain times and not used at others. Toggling whether a subscription is currently in use is done through the **Subscribe** and **Unsubscribe** buttons below the list. Subscriptions that are in use are indicated by the red X in the box to the right of the line.

Subscriptions can be added to the list via the **Add Subscription Line**. The subscription list attempts to anticipate how the user will enter subscriptions allowing the user to type only the minimum information necessary. For instance, typing a single word will add a subscription of class MESSAGE, instance <the word typed>, recipient * (all recipients). If a specific triple <class, instance, recipient> is desired it can be typed in with each component separated by commas or white space. Furthermore, usernames appearing in the recipient column are automatically completed with the current domain context if none is provided. Subscriptions can be removed from the list permanently by selecting them and hitting the **delete** key on the keyboard. Additional functionality for moving subscriptions to and from the clipboard is provided via the right-mouse button context menu for the list. As with all property pages no changes are made permanent until the **OK** or **Apply** button is pressed. Only if there are changes in the current subscriptions is a **Resubscribe** executed. This **Resubscribe** is performed after the dialog box is closed to reduce traffic between clients and the zephyr server.

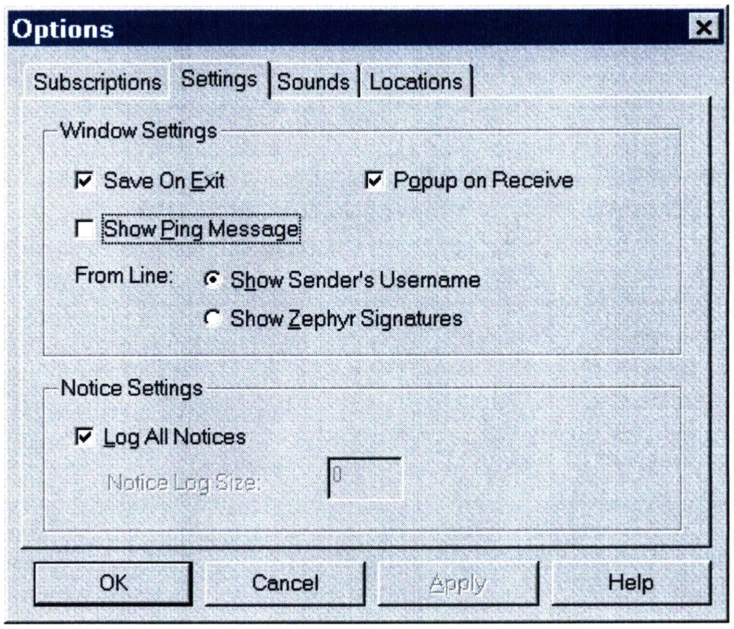
The **With Defaults** option indicates whether this client is a default client. Default clients are automatically subscribed to PERSONAL messages as well as system error messages. PERSONAL messages are those messages that are sent directly to a user. This automatic subscription is equivalent to the triple <MESSAGE,PERSONAL,username>. System error messages are those sent by the Zephyr System itself to indicate changes in the status of the service. If there are multiple Zwgc clients currently open in the environment, normally only one of them will be marked as default. If no client is marked as default and no client explicitly includes the above triple, then no personal messages will be received.



MESSAGE, 6.170, *
MESSAGE, 6.170.D, *
MESSAGE, HELP, *
✗ MESSAGE, ZEPHYRNT, *
✗ MESSAGE, ZEPHYRNT.BETA, *
✗ LOGIN, MSREDIN@ATHENA.MIT.EDU, *
Click Here to Add a New Item...

[✓] With Defaults

Figure 5-12: Subscriptions Property Page



Window Settings

[✓] Save On Exit   [✓] Popup on Receive

☐ Show Ping Message

From Line: [● Show Sender's Username] ○ Show Zephyr Signatures

Notice Settings

[✓] Log All Notices

Notice Log Size: 0

Figure 5-13: Settings Property Page

The **Windows Settings** page shows the various configuration settings for the client. The **Save On Exit** command has already been described. The **Popup on Receive** option toggles the display of popup windows on notice receipt. If popups are turned off messages are only logged. If both popups and logging are turned off messages are discarded. The **Log All**

Notices option toggles whether the log is a fixed length or grows to hold all messages received. If **Log All Notices** is not checked a log size must be entered into the edit box provided. If the log size is set to zero logging is disabled.

The **Show Ping Message** option toggles the display of pings. Ping messages are generated by the UNIX command line client when a message is begun. Pings are intended to ensure that a user is logged in before a message is authored. Some users like to have pings displayed as they indicate that a message is imminent and provides the username of the author of the new message. For others, ping messages are merely a nuisance. The option to toggle these messages is a common customization from zwgc.desc files in the UNIX version of *zwgc*. More will be said about this and other common configuration extensions in the next section. The last option, **From Line:** allows the user to decide whether usernames or Zephyr Signatures should be displayed in the client log window. In an environment where signatures are normally used to hold the complete name of the user, there is some benefit to displaying the signature rather than the username. In the MIT environment, where most signatures do not give any indication of the message's author, showing usernames is more appropriate.

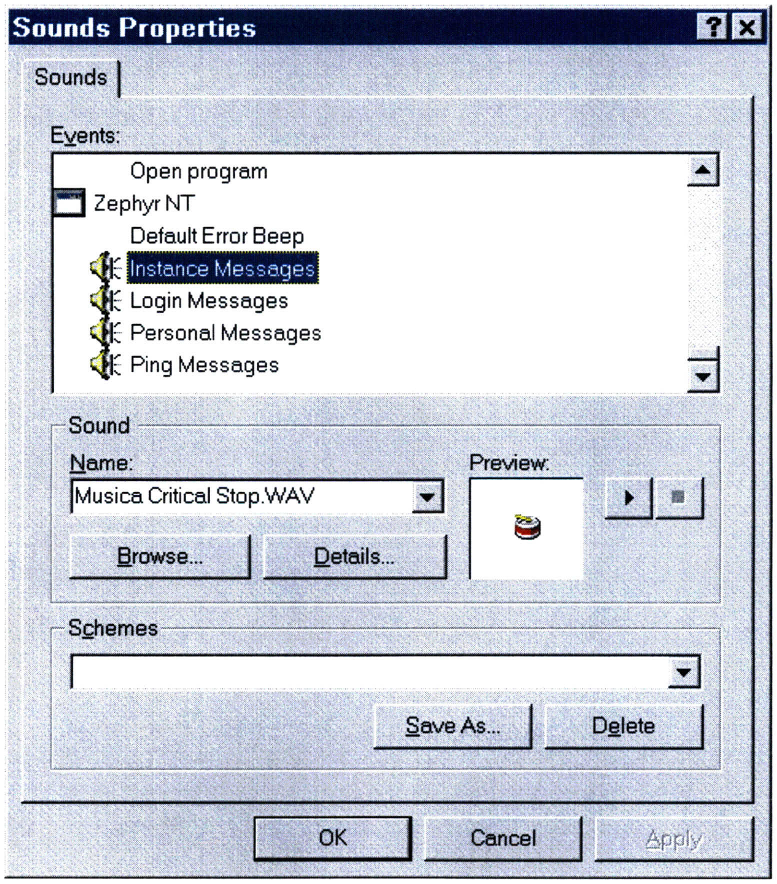
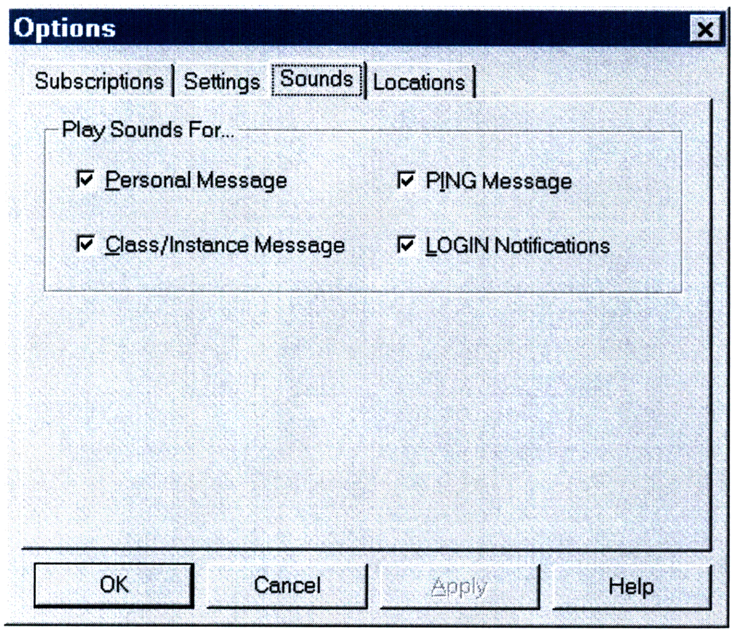


Figure 5-14: Sound Control Panel Applet

The **Sounds** page allows the user to toggle the use of sound effects for different types of messages. Toggling on sounds for a message type indicates that the sound associated with that type should be played whenever such a message is received by this client. Which sound is actually played is mapped by the standard Sound Control Panel applet. When ZephyrNT is installed a new section for its sounds will appear in the applet's list (shown in Figure 5-14). Sounds may be bound to the different events as they are with other programs in Windows NT.

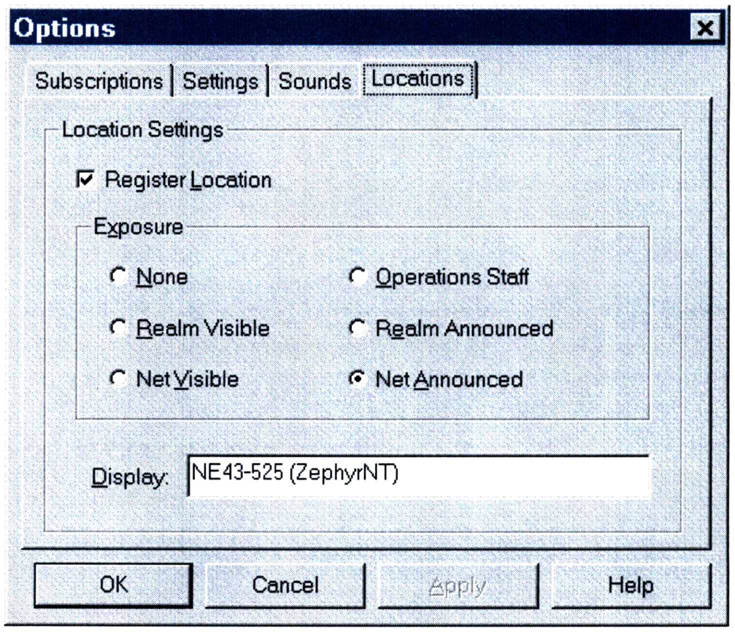
The **Locations** page configures what entries are made to the locations database. As with the **With Defaults** command above, normally only a single Zwgc client will register a location at once. Selecting **Register Location** toggles on and off location registration for this client. The **Exposure** box allows the user to set the extent to which other users on the system will be able to view the location database information. *None* is essentially equivalent to not registering a location, as no one will be able to query the entry. *Operations Staff* indicates that only Zephyr System Operators should be able to query for the entry. *Realm Visible* and *Net Visible* indicate that queries should be allowed by the immediate domain or the entire network respectively. The announced versions of these two entries indicate the same exposure level as their non-announced versions but additionally they indicate that the location database should issue LOGIN/LOGOUT notices when the entry is added and removed from the database. These are the notices we saw earlier that are tracked by the location client. Unannounced users can still be queries but they will not be tracked properly (this is one of the reasons for the periodic updating done by the location clients). The normal value for most reasonable usage is *Net Announced*.



Play Sounds For...

- [✓] Personal Message
- [✓] PING Message
- [✓] Class/Instance Message
- [✓] LOGIN Notifications

Figure 5-15: Sounds Property Page



Location Settings

- [✓] Register Location

Exposure

None, Operations Staff, Realm Visible, Realm Announced, Net Visible, [✓ Net Announced]

Display: NE43-525 (ZephyrNT)

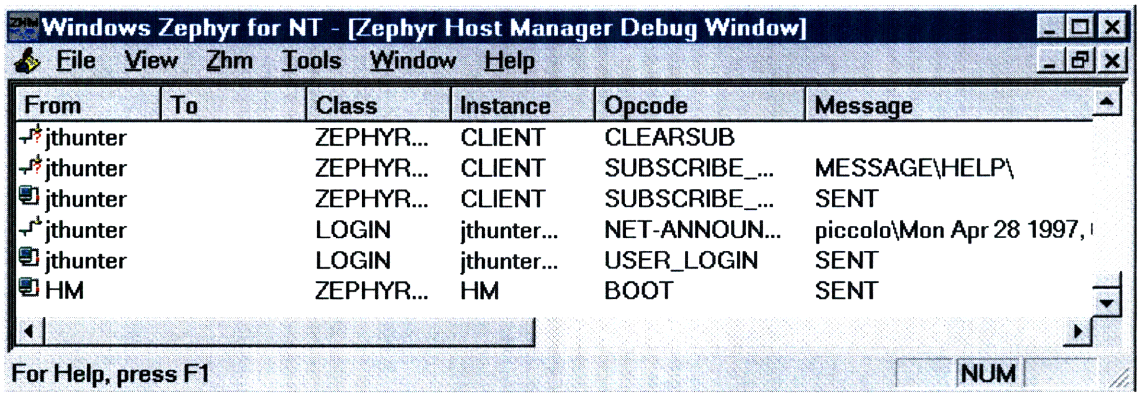
Figure 5-16: Locations Property Page

The **Display** field allows the user to configure the display string. As was described earlier, this is a convenient field to provide additional information about where or how the user is connected to the network. In Figure 5-16 above the display field is being used to indicate an office number and that ZephyrNT is being used (rather than the UNIX client).

5.1.5 The Zhm Client

The Zhm client implements the same functionality as its UNIX counterpart. Furthermore, as with its UNIX counterpart, only a single Zhm client can be running on a particular machine. When ZephyrNT is started, it attempts to detect whether an existing Zhm client is already running on the system (i.e., another version of ZephyrNT is already being run by this or another user). If an existing client is detected, a new one is not started. When ZephyrNT shuts down and is the last instance of ZephyrNT in the system, it shuts down the current Zhm client. Furthermore, if at any time ZephyrNT detects that no Zhm client is currently running, it notifies the user that the current Zhm client has closed and asks the user if he wishes to start a new one. If no Zhm client is started, all instances of ZephyrNT will halt message transmission until one is started. The Zhm client is shut down when no instances of ZephyrNT are running, to save resources when no clients can receive messages. Synchronization of existing Zhm clients is handled through system-wide shared memory coupled with a mutex variable. If other client packages were designed to utilize the ZephyrNT zephyr library, they would need to use the same negotiation protocol to ensure that an existing Zhm client was in operation before they proceeded.

As described above in Section 2, Zhm is responsible for coordinating communication between the clients and the server. Configuring Zhm is described in Section 5.2.2: Environment Settings. Normally Zhm is invisible to the user; however, if ZephyrNT is started in debug mode (via the /debug command line flag) and that instance of ZephyrNT starts a new Zhm (this can be forced with the /StartZhm flag) ZephyrNT will attach a Zephyr Host Manager Debug Window client to the active Zhm. This client can be useful to administrators of a site in finding trouble that may lead to loss of service for Zephyr clients. The window shown below in Figure 5-19, looks similar to the Zwgc client windows already seen. The default configuration for this client is to display all columns for review. This window includes all messages sent by clients in the system as well as all special messages sent by the zephyr library. The icon on the right indicates the type of message[8].


Windows Zephyr for NT - [Zephyr Host Manager Debug Window]
File View Zhm Tools Window Help
For Help, press F1
NUM

From	To	Class	Instance	Opcode	Message
jthunter		ZEPHYR...	CLIENT	CLEARSUB	
jthunter		ZEPHYR...	CLIENT	SUBSCRIBE_...	MESSAGE\HELP\
jthunter		ZEPHYR...	CLIENT	SUBSCRIBE_...	SENT
jthunter		LOGIN	jthunter...	NET-ANNOUN...	piccolo\Mon Apr 28 1997,
jthunter		LOGIN	jthunter...	USER_LOGIN	SENT
HM		ZEPHYR...	HM	BOOT	SENT

Figure 5-19: Zhm Debug Window

5.2 Environment Configuration

ZephyrNT stresses visual configuration. This paradigm is a keen aspect of the look-and-feel of Windows NT (and other Windows products) and strongly differentiates NT from its UNIX counterparts. Traditional UNIX programs are configured via a series of text files, commonly known as dotfiles[9]. The trouble with the dotfile approach is management. A user must remember which dotfiles belong to which applications. Furthermore, since little standardization is done across them, users must keep track of which conventions are used in each specific dotfile. Windows first endeavored to avoid this problem by providing a standardized API for creating and modifying these files[10]. This API defined a specific format for configuration files, known in Windows and .ini or initialization files. Furthermore, the

[8] These icons also appear in the Zwgc client window but they are always the same because the Zwgc window only receive one type of message.
[9] The name dotfiles is due to their propensity for having names that begin with a single period. Files whose names begin with a single period are render invisible in normal directory lists under most versions of UNIX.
[10] These are .ini files to Windows users.

API allowed Windows application designers to provide visual interfaces for editing this information quickly and easily.

With the advent of Windows NT, it became necessary to extend the ini infrastructure to handle an multi-user environment. Windows NT implemented the Registry, a initialization database. All information formerly maintained by initialization files is now kept in the Registry and the above API set now enters data into the registry instead of the former ini files. The Registry is partitioned into three main sections: USERS, LOCAL_MACHINE, and CLASS_ROOT. The CLASS_ROOT section maintains information about OLE components and is not of interest in the present discussion. The LOCAL_MACHINE section maintains information that is relevant to the machine as a whole while the USERS section maintains information that differs from user to user. ZephyrNT makes heavy use of the Registry for its configuration needs.

Each client described in Section 5.1 is responsible for setting and maintaining its own configuration. The property dialogs shown early form the basis of this configuration mechanism. All client specific properties are maintained in the clients' documents themselves. A more detailed explanation of the document architecture is given below in Section 5.3. In addition to the properties of clients, ZephyrNT has some additional configuration information. This information falls into two basic categories, user specific information and library installation information. This information is stored in the Registry in the USERS section and the LOCAL_MACHINE section respectively. The user specific information includes the information necessary to initialize the environment as well as the users signatures. This information is not specific to any given client and so is the responsibility of the environment itself to maintain. The user modifiable settings are managed by the second two pages of the **Options...** command on the Tools menu. These pages are shown in Figure 5-18 through Figure 5-20.

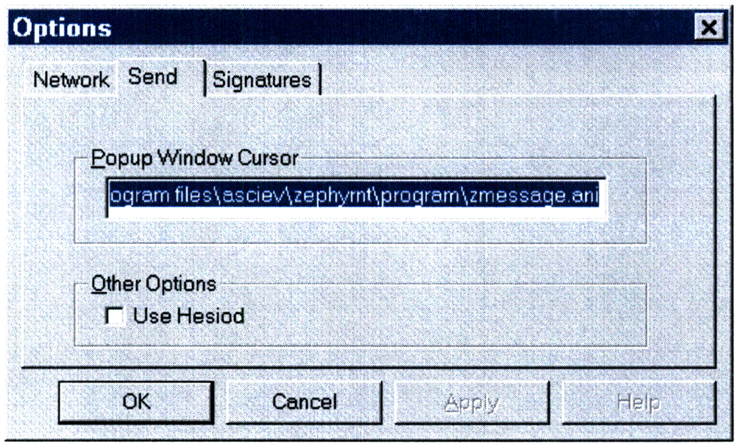


Figure 5-18: Options Send Page

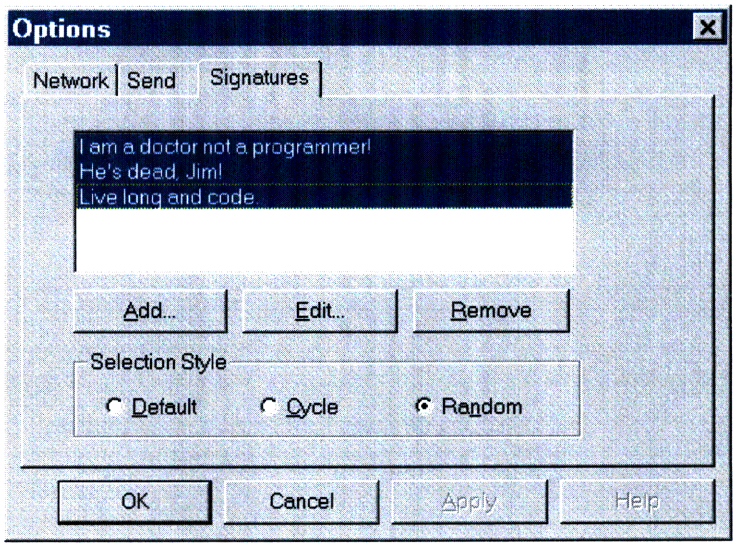


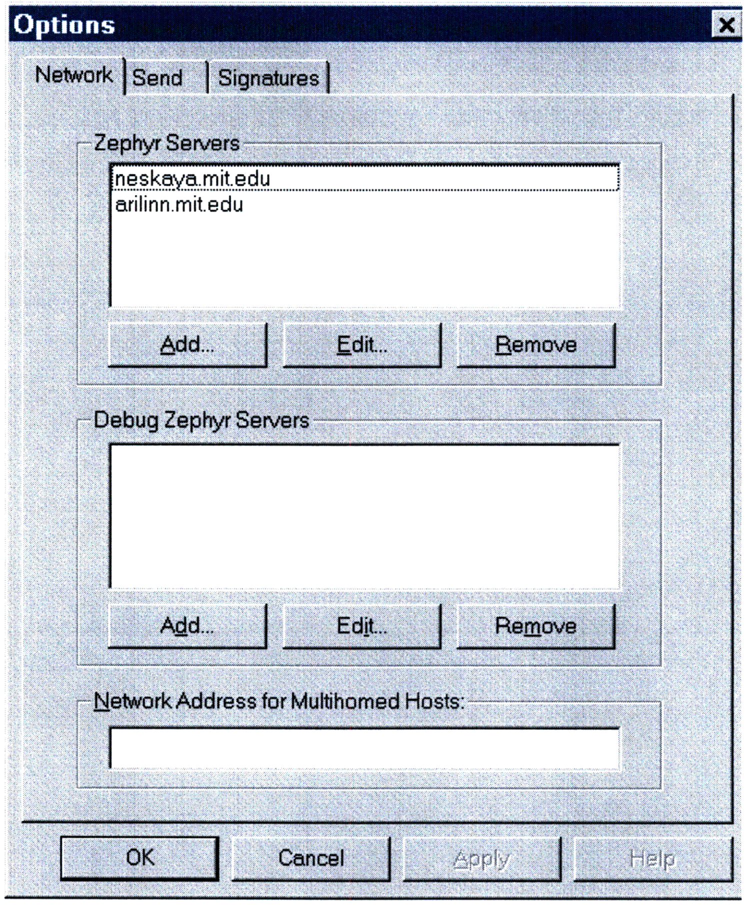
Figure 5-22: Options Signatures Page

5.2.1 User Settings

The **Signatures** page allows the user to include a list of zephyr signatures and the option of how these will be used. The list is a simple list of text strings, however, like the Subscriptions page described above this list allows some signatures to remain in the list while they are not currently in use. The list will use only those signatures that are selected when the **OK** or **Apply** buttons are clicked. The user can select among the list using either shift-click or control-click combinations. (This follows the normal Windows convention for selecting among a group of alternatives.) The option group below allows the user to decide what will be done with the selected signatures. **Default** indicates that only the first selected signature should be used. **Cycle** causes ZephyrNT to begin at the first selection and cycle downward through each of the selected items in turn. **Random** indicates that ZephyrNT should choose a signature at random from among those selected whenever a new message is created.

The **Send** page contains only two entries. The first is a user specific entry and the second is a environment entry. The **Popup Window Cursor** allows the user to specify a cursor resource that will be used as the mouse cursor for the popup windows. The field must contain the full path to this resource. In the example image, this field points to the animated cursor resource file zmessage.ani which is included with the beta version of the software.

In addition to the options on the above pages, the USERS section of the Registry is used to store the names of all currently open clients as well as the location and state of all toolbars and docked windows. The environment reopens all clients that were open when the last session ended, returning them to their former location and orientation. This frees the user from managing the separate client documents. All of these parameters are maintained separately for each user. If multiple users access ZephyrNT on the same machine, they can maintain completely independent environments from each other.



neskaya.mit.edu
arilinn.mit.edu

Figure 5-20: Options Network Page

5.2.2 Environment Settings

The environment is also responsible for maintaining parameters for initializing the library or the Zhm. Most of these parameters can be found on the **Network** page of the Options property sheet (shown in Figure 5-20). This page contains three parameters. The first two are lists of Zephyr System servers to use. Entries are given as full DNS names of machines running the server side of the system. Two are shown in the above example. The Debug server list is for use in system debugging. These servers are used instead of the regular servers when ZephyrNT is started up in debug mode. These entries can work with or replace the remaining entry on the **Send** page. The **Use Hesiod** option indicates that the Zhm should attempt to get a list of servers from the Hesiod service. This service is in use, for the most part, only at MIT. If ZephyrNT is able to retrieve a list of servers from the Hesiod service, this list is appended to the list provided by the first entry on the **Network** page. Servers are chosen from the combined list at random. This ensures an even distribution of server activity. If a server becomes unavailable, it is temporarily removed from the list until a live server can be found. If a server name ever becomes unrecognized by the DNS servers, it is removed from the list. If no server can be contacted or all servers are removed from the list for one of the two reasons listed above, ZephyrNT notifies the user and halts message transmission until the problem is corrected. Though many servers may appear in the list ultimately generated by the above process, only a single server is in use at any given time.

The last entry on the **Network** page allows the user to pick which of several IP addresses to use. This field is intended only for configuration on multi-homed hosts, that is hosts with more than one network card connected to multiple networks. If the default network address does not lead to a network where the zephyr servers are located, zephyr packets may never be appropriately communicated and system communication will fail. This field allows an administrator to configure which IP address is connected to the same network as the Zephyr System. This field should be left blank if only a single network card is installed in the machine. Improper use of this field can cause system malfunction resulting in the loss of communication.

All network parameters affect how the zephyr library or Zhm access the Zephyr System and, therefore, are machine dependent rather than user dependent. They indicate facts about the state of the network on which the machine lies. These parameters are stored in the LOCAL_MACHINE section of the Registry and only a single copy of them is stored for all users of the system. Security permissions on this section of the Registry prevent unauthorized users from making modifications to these parameters. Incorrect modification of these parameters could render the system useless.

5.2.3 Trading off Dotfiles vs. User Interface

Giving up dotfiles, as with all design decisions, does have its down side. The UNIX package is made of a number of different programs. Each program has its own set of configuration files and each of these files allows the system to be fine tuned for different uses. The three files that are of the most interest to the client side of operations, the functionality that ZephyrNT currently provides, are the dotfile for zwrite (.zephyr.subs) and the two dotfiles for zwgc (.zephyr.vars and .zwgc.desc). The .zephyr.subs file is responsible for maintaining a list of the user's current subscriptions. The .zephyr.vars file is primarily used to set location exposure. Both of these features are provided in full by the Zwgc client described in Section 5.1.4. However, the .zwgc.desc file provides a greater challenge. This file determines what actions are taken to format and display incoming messages. The *zwgc* program provides a command language to express these actions. Though this command language is simplistic, it can be quite powerful. The .zwgc.desc file is the source of many user customizations and modifications. Because ZephyrNT's design goal was to avoid dotfile style configuration, it became necessary to determine what the *correct* actions for incoming messages were to be.

It was clear that the default actions of *zwgc* (when the .zwgc.desc file does not exist) were unacceptable. This observation can easily be appreciated by recognizing the enormous diversity of existing .zwgc.desc files. It is difficult to anticipate what every user of the system in the future will hope to do. Fortunately, the problem was not quite this bad. ZephyrNT's aim was to take advantage of the vast range of development already put into .zwgc.desc files. A broad range of .zwgc.desc files were examined to make a list of the common customizations that were being done. Existing customization led, in part, to the set of features that ZephyrNT attempted to export. In particular, logging of message, hiding pings, and manipulating lists of signatures seemed to be reoccurring features that had broad appeal. Each of these features and how they have been implemented in ZephyrNT has already been described in the above sections.

Due to the great variety in customizations uncovered through .zwgc.desc file examination, it was not possible to include all customizations. The above set was chosen because of their apparent commonality and usefulness. However, to better accommodate the needs of users, the interface of ZephyrNT was designed to be as flexible as possible. For instance, the interface attempts to provide a complete set of keyboard accelerators for accomplishing all common tasks in addition to a menu/mouse-driven approach for less advanced users. Zwgc windows allow the option of displaying log windows (like *Emac-Zwgc* and *MacZephyr*) or popups (like the X-Windows *zwgc* client). Sound binding to message types was provided to anticipate users' desire for audible feedback. Finally, extensions to zephyr content transmission and display capabilities were made. These will be described in more detail in Chapter 6.

In addition to providing some of the customizations that existed prior to this project, ZephyrNT attempts to take as much advantage as possible of the windowing forms present in other applications for Windows NT. The end result of this effort is the two commands that have appeared on several of the client menus: **Show as Toolbar** and **Show as MDI**

Window. These two commands are complementary and cause the associated client to move between windowing states.

MDI, or Multiple Document Interface, windows are standard overlapping windows. MDI is an older standard for Windows based applications that is being phased out by current vendors. However, many current Windows applications take advantage of its commonality and many users are accustomed to its features. The MDI interface defines a set of keyboard accelerators and other features for managing MDI client windows. All programs compliant with MDI provide this set of features. In particular MDI defines accelerators for changing between windows, closing windows, opening windows, minimizing windows (to an icon at the bottom of the main window) and maximizing windows (to encompass the entire client area of the main window). MDI windows all share a common menu bar and are clipped by the main window (the main window is the one with the menu bar). In addition to accelerators, MDI defines the Window Menu. This menu contains standard commands for managing and arranging the MDI windows as well as a dynamic list of the currently open MDI windows. ZephyrNT is an MDI compliant program. All clients (except the Zlocate bar) can exist as MDI windows. All of the above features are implemented for these windows.

In addition to MDI windows, a common interface element that has been gaining increasing acceptance by program vendors is the dockable floating toolbar. Toolbars are windows with a smaller title bar that appear on top of an applications main window (when not docked). These windows always remain above the main window and any existing MDI windows but may be overlapped among themselves. In addition to floating above the main window, toolbars may be docked to the sides of the main window. When docked, a toolbar becomes a feature of the main window's outer frame itself rather than a MDI window, which appears inside the frame. Traditionally toolbars have been used only for holding small toolboxes or toolbar buttons (the mouse equivalent of a keyboard accelerator). These windows were almost never resizable and frequently could only be docked in limited locations. ZephyrNT and many other application vendors have begun to realize that these windows make good client windows if managed correctly. ZephyrNT pushes the boundaries of this interface by allowing all clients to switch between MDI mode and extended toolbar mode. These extended toolbars can be docked anywhere in the main window. Furthermore, they are resizable both when floating and when docked. In addition it was observed that there is limited real-estate around the edge of the main window for toolbars to be docked but ZephyrNT's other design features encourages users to maintain a large number of open clients. Therefore, ZephyrNT merges clients of similar types into a single floating toolbar, providing the user with a tabbed box within the toolbar for switching between clients in toolbar form. ZephyrNT provides the Zwgc-Bar where location clients and zwgc clients are merged and a Sender-Bar where sender windows are merged.

By providing both MDI and toolbar windowing environments, ZephyrNT hopes to be able to provide the specific user interface that each user is interested in. Users who are familiar with the older MDI windows may use those exclusively. Users who are more comfortable with the free floating toolbars may use those. More advanced users may choose to use a combination of MDI windows *and* toolbars as some of the example images in this document have done.

5.3 How It Works

ZephyrNT divides its functionality mostly along document boundaries. Each document type provides its own set of functionality and user interface. This is not an entirely new concept in the Windows environment. Early products such as Works made extensive use of this technique. More recent products such as Visual C++ and Microsoft Office take a more subtle approach. ZephyrNT's design fits into this second, more recent, model. The idea behind this approach is to present a collection of disparate programs in a manner that emphasizes their unity in an overall system for completing a given task. Visual C++ presents programs for editing code, compiling code, editing binaries resources, drawing bitmap images, managing source control, and searching specifications and help files. These tools are vastly different in interface and function from each other but they are all necessary for generating a well crafted Windows-based program. ZephyrNT presents its functionality in the same manner. ZephyrNT provides tools for authoring and sending messages, receiving and storing messages, managing user locations and tracking, and local/remote message communication. Each task is different, but together they all provide the tools necessary for communicating with other Zephyr users.

5.3.1 Document Division Advantages

Document based functionality provides an additional advantage. Multiple simultaneous documents is a well accepted paradigm. This should be clear from the discussion of the MDI interface in section 5.2.3. Multiple documents can be used not only to display several different document types but to display several documents of the same type. This allows the user to further divide his goals along other, more personal, lines. For instance, multiple znol's can track different sets of users and multiple zwgc's can manage/log different sets of subscriptions. Every document type in ZephyrNT, except the zhm, can be multiply instanced.

5.3.2 Document Threading

Although each document encapsulates its own functionality, they all share a single main window. In addition, each document window is a child of the main window. This has several implications. First, if a single document performs a lengthy operation it may interfere with the user responsiveness of other documents. Though the zephyr library does provide processing of user interface messages during blocking states, it does not attempt to process these messages during the course of normal operations. When more then a few clients are working simultaneously, the library may spend most of its time servicing request from these clients. This results in user events not being responded to in a timely manner. There are two immediate solutions to this problem: force the library to perform message processing periodically during long operations or thread the document classes, allowing their work to be done outside the user interface thread. The latter choice was opted for ZephyrNT since the library was already designed with threading support. Choosing the former would have

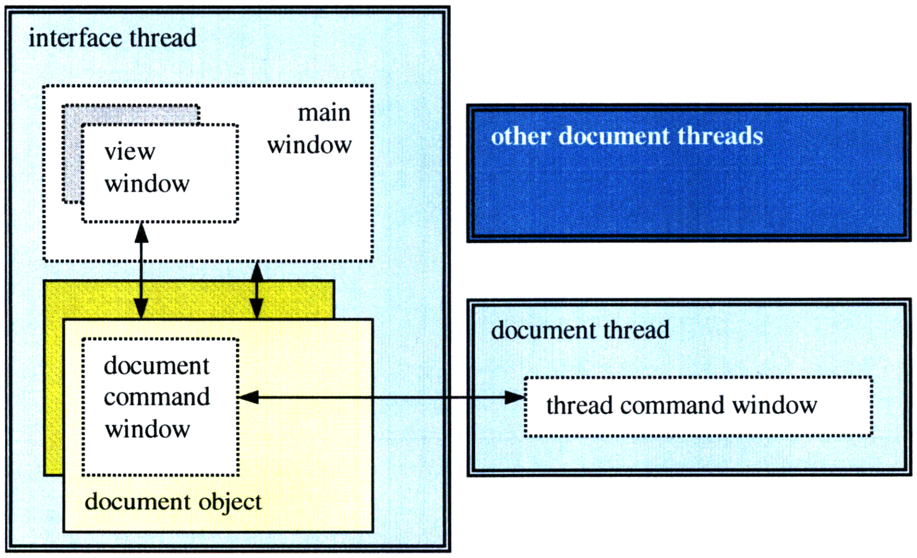


Figure 5-24: Document Objects

unacceptably complicated the code implementing the zephyr library.

A single main thread is responsible for maintaining and reacting to all user interface elements. This thread is also responsible for creating the other threads upon document creation and for shutting down all of the threads when the program completes. Each document type is divided into a collection of objects, each of which is responsible for a part of the overall task. Some of these objects exist in the main thread, while others exist only in the separate document threads. Figure 5-24 shows a pictorial representation of the object relationships. Giving each document a separate worker thread allows the main thread to provide very rapid responsiveness. Lengthy tasks that need not interfere with the continued operation of other documents are queued to worker threads. Some lengthy tasks are performed by the main thread, such as document loading during environment initialization. When this situation becomes necessary, ZephyrNT displays wait animations to indicate to the user that the application is currently unavailable.

Each document instance maintains a single *document object*. This document object is responsible for all of the persistent state of the document. *Document objects* are created by the main thread either when a new document is requested or an old document is loaded from a file. After initializing the current state of the document (either as empty or from the file contents), the *document object* creates a *document command window*. The *document command window* will be the channel of communication between the document and its worker thread. The *document object* then creates a *document thread object* passing it the handle of the *command window* created in the previous step. *Thread objects* are the worker threads, one for each document. The *thread object* in turn creates a new *thread command window* and passes its handle back to the owning *document object* before completing its own initialization process. Only then does the document create the *view window*. The *view window* is the MDI child window that appears in the main window. This *view* along with the shared menu bar and other tool or popup windows provide all input from the user to the document. Interaction with these windows cause each document type to generate the appropriate tasks for its worker thread.

When a document needs to begin a new task it constructs a task message and sends that message to its worker thread's *thread command window* through the handle return during the initialization process. Similarly, when a worker thread finishes a task, it constructs a response message which is posted to *document command window* passed it during creation. ZephyrNT enforces a strict protocol requiring the creator of an inter-thread message to allocate storage for the message contents. Once a message is sent, the creator must assume any handle it held to that memory is no longer valid. The receiver of a message is responsible for freeing all storage associated with the message. The Windows messaging queue infrastructure is used for message passing because it provides a clean, thread-safe queuing mechanism. Worker threads perform a single task at a time, taking one from the top of the queue completing it, sending a response message, and then returning to the queue for the next. This implies that tasks are completed in the order that they are queued, i.e. FIFO style. Though true in this implementation, ZephyrNT's architecture does not rely on this fact. A single copy of all document related data is maintained by the *document object*. Worker threads are permitted to access this information when necessary through a document data mutex.

Having a separate thread for each document does generate a small amount of overhead for the operating system. However, most worker threads spend most of their time waiting for network messages. As discussed in Chapter 3, the zephyr library converts network messages to windows messages and places them in the message queue. Furthermore, the zephyr library processes queued window messages even while waiting for a network response. If a thread's message queue has no pending network message, and no pending windows messages, the operating system will place that thread in a wait state until a message becomes available. Therefore, though worker threads do generate overhead in the scheduler's list of pending threads they do not consume CPU cycles when they have no useful work to do. ZephyrNT makes use of additional worker threads for some of its user interface features. For example, during lengthy tasks ZephyrNT spawns a worker thread to display its folder-passing animation. Usage history for ZephyrNT over the course of this project shows typical thread count to range between 10 and 15 depending on how many documents a user manages. The average thread count for complex system services is approximately 15-30. The average for single applications is 1-5. This places ZephyrNT roughly in between a single application and a service, which is reasonable for its intended functionality.

5.3.3 Remote Access Connection Support

ZephyrNT was designed with remote access connection in mind. Connecting a computer to a network via remote access is supported by Windows NT and Windows 95 through Microsoft's Remote Access Service. However, connecting remotely can present a number of problems to a communication service like ZephyrNT. In particular, communication is generally slower, less reliable, and prone to disconnect. Nevertheless, ZephyrNT operates quite well in this environment. Zephyr messages are typically small, so network bandwidth is not usually a limitation. The noisiness of phone connections, however, can dramatically increase the replay rate for lost packets. This replay is performed mostly by the zephyr library automatically but can result in more frequent timeouts in the upper-level protocol. For this reason, ZephyrNT uses slightly longer timeout values than those used by the UNIX implementation. In addition, ZephyrNT will lengthen the timeout values still further when run in debug mode, to facilitate debugging of messages in actual operation.

Disconnected operation presents a different sort of problem. Many remote access solutions provide a dynamic IP address upon connect. When a system becomes disconnected and reconnects it is likely to receive a different IP address from the one it previous had. This can cause improper operation for two reasons. One, the zephyr library caches the IP address of the local machine for efficiency. Further, the Kerberos library includes the current IP address as part of its ticket format. If the IP address of the machine changes but new tickets are not acquired, the IP address in the tickets will not match the IP address of the network packet carrying the ticket to the zephyr servers. The zephyr servers will interpret this difference as an authentication error and will refuse the packet. To avoid this problem, ZephyrNT will update its cached address periodically. However, ZephyrNT has no control over the contents of the Kerberos tickets. A copy of Leash32 (the Win32 Kerberos management program) is distributed along with ZephyrNT to re-initialize the Kerberos library in the event that it becomes unsynchronized. Dialup services that present a fixed IP address to each user will not generate the above problem.

6. Content Enhancement

Zephyr was designed as a mechanism to send small textual messages between individuals or groups of people. However, the use of the networks has expanded considerably and current trends in network communication have put an emphasis on more dynamic multimedia content. This chapter outlines a couple of avenues for expanding zephyr's content expression. First, section 6.1 describes the formatting capabilities of the MIT implementation. Sections 6.2 and 6.3 describes two different efforts to extend these capabilities, first through RTF and second through HTML.

6.1 The State of the UNIX Tools

Zephyr's original implementation does allow some content expressiveness in a limited form. Though none of the clients provide support for authoring formatted content in its display form, *zwgc* does provide support for a simple formatting language via the following features.

@roman	Turns off @italic and @bold
@bold	Turns on boldface
@italic	Turns on italics
@left	Left aligned
@center	Center aligned
@right	Right aligned
@large	Large type size
@medium	Medium type size
@small	Small type size
@beep	Ring the X bell
@font	Sets the current font

@color	Sets the current color
@()	Create sub environment.

Figure 6-1: Formatting Commands

The formatting of a message is performed by interspersing the message text with the commands. Most commands set an attribute only for the text within the brackets following the command. The font and color commands set an attribute that remains true until the end of the current environment. The effects of a command can be limited by creating an enclosing environment with the @() command. Most commands can be nested.

6.2 Rich Text Format (RTF)

The first attempt to expand the above formatting tool set was to provide support for RTF. RTF or Rich Text Format is a document interchange format invented by Microsoft for converting between a variety of word processing programs. RTF is not a particularly efficient encoding scheme which is perhaps the prime reason it is not used much commercially. NeXT Computer did use RTF in its operating system for its volumes of help information. In addition, NeXT's default editor produced RTF output. Windows has provided a number of tools for writing RTF throughout its various versions. Furthermore, RTF is a strictly text-based encoding scheme, like the zwgc formatting language above. That is, RTF does not rely on binary characters outside the standard ASCII range to encode formatting information. This makes RTF strictly compatible with the existing Zephyr System. All public beta releases of ZephyrNT, i.e. those released to the MIT community, include the RTF display support described in this section. However, because RTF authoring support was disabled few beta testers are aware of this functionality.

RTF provides a robust set of formatting tools. The message shown in Figure 5-10 on page 27 shows some of the more rudimentary features of RTF. This zephyrgram is an RTF message sent and received by ZephyrNT. The message demonstrates that RTF can perform at least the level of formatting capabilities that can be achieved by the *zwgc* formatting language. RTF also supports more advanced formatting feature such as ruler-based tab stops, bulleted and numbered lists, tables, and embedding graphics.

RTF, however, proved to have a number of drawbacks that lead to the conclusion that it is not a good choice for a transfer medium. The first, and perhaps most important is RTF's size. When the more advanced features of RTF are used, the size of a message can grow quite rapidly. This is exemplified by the use of embedded graphics. Even a small graphic can make the zephyr message so long that it is not likely to ever arrive. The reason for this is that large messages in the existing server architecture are broken up into separate packets. A notice is not displayed until all the packets are received. However, partial messages are only kept around for a short period. If all of their parts have not arrived by then it is assumed the message is dead and its received packets are discarded. If a message is long enough, all of its packets are not likely to be received before the message times out. This has the result that the message is discarded even though it technically was never lost.

The second major drawback to RTF is its rapid versioning. There are several versions of RTF that have been released. Microsoft posts the official specifications to the most recent version of RTF on their web site, however, their authoring products are always one version ahead of this specification. In general this should not be a problem except that the RTF display objects have a tendency to not display versions of RTF that they do not understand. Unless ZephyrNT wishes to implement its own authoring and display technology for some fixed version of RTF it becomes necessary to rely on Microsoft to keep the two technologies in sync. However, the technologies frequently become out of sync, having the disadvantage that when a new version of Windows was released, the display object was updated, but the authoring tool was updated periodically between Windows releases, which causes messages not to appear. The current release of ZephyrNT was designed to display the version of RTF written by Write version 3.51. The display object used in the popup windows is capable of displaying this version of RTF. However, Windows NT 4.0 was the first version of Windows NT that no longer distributed the Write application. Write was replaced with the more advanced program WordPad which authors a version of RTF consistent with that produced by Word 95 (version 7.0). This version of RTF is mostly compatible with ZephyrNT though there are some anomalies. The most recent version of Word (version 97/8.0) produces a RTF that is not compatible with ZephyrNT.

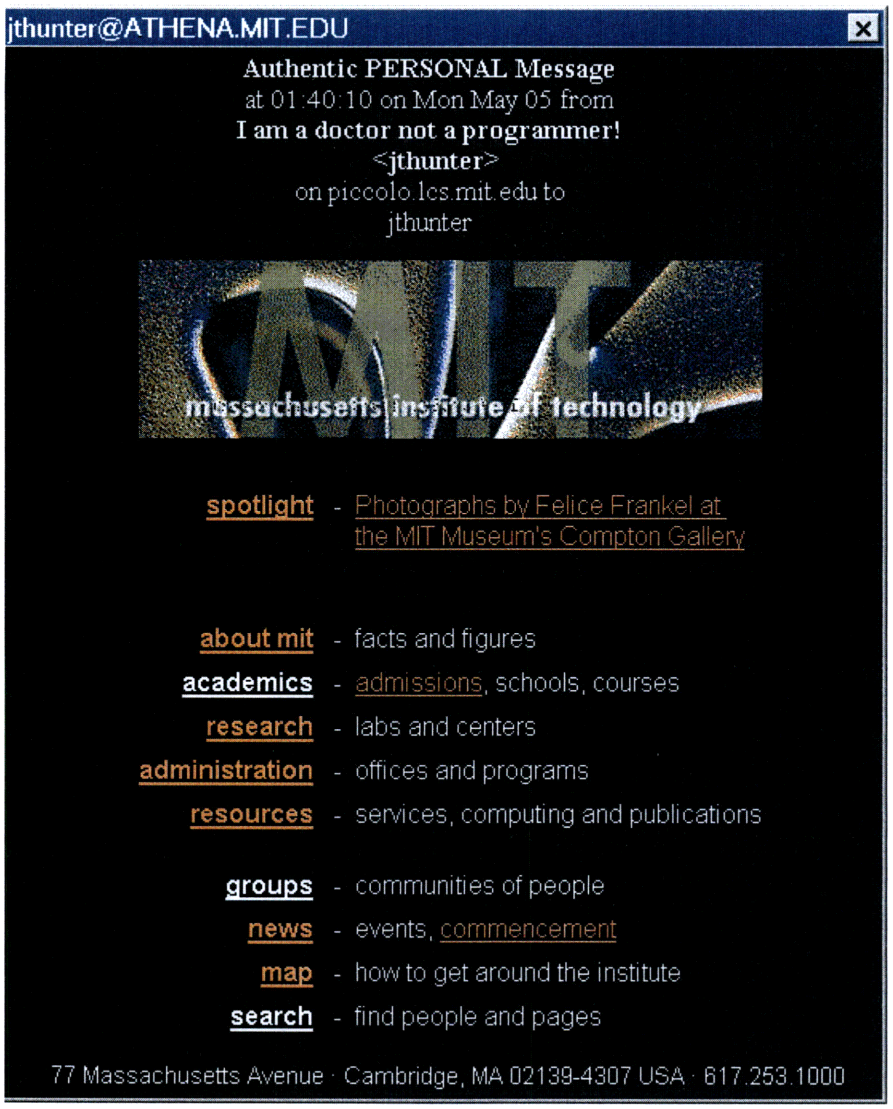
Lastly, RTF proved not to work very well in combination with text messages being used by UNIX users. RTF text ends up with about as many formatting characters as it contains textual data. This has the disadvantage that raw RTF is basically unreadable to those without a RTF display tool. For example, UNIX users might receive a RTF message from a ZephyrNT user that looks like the following:

```
{\rtf1\ansi\deff0\deftab720{\fonttbl{\f0\fswiss MS Sans
Serif;} {\f1\froman\fcharset2 Symbol;}{\f2\froman\fprq2
Times New Roman;} {\f3\froman Times New Roman;}}
{\colortbl\red0\green0\blue0;} \deflang1033\pard\li720\fi-
720{\*\pn\pnlvlblt\pnf1\pnindent720{\pntxtb\'b7}}\plain\f2\f
s20 {\pntext\f1\'b7\tab} \par {\pntext\f1\'b7\tab}a \par
{\pntext\f1\'b7\tab}bullet \par {\pntext\f1\'b7\tab}list\tab
\par \pard\plain\f2\fs20 \par }
```

It is clear that these messages could not co-exist with UNIX users. This problem could be solved by not allowing ZephyrNT to send RTF messages to users not using ZephyrNT as a client. This detection proved difficult and the determination was left up to the user whether they wished to author RTF messages or not. However, further difficulties arose from UNIX users being unaware of the RTF display nature of ZephyrNT's messages. Though the RTF display engine does not have a problem with treating straight text files as RTF without any formatting, there were odd occasions when a user would stumble across a RTF command switch or reserved word. This usually resulted in their message not appearing or appearing as an empty window. The worst instance of this was a couple of MIT users whose zephyr signatures contained reserved words. This had the effect that no message they sent was properly received by a ZephyrNT client.

6.3 HTML

HTML was the second transfer medium supported by ZephyrNT. Moving to HTML addresses a number of the difficulties demonstrated by RTF. In general HTML code is much smaller for the same expressive power. In addition, graphics need not be embedded to be displayed in a message. This means that only the actual HTML text need pass through the zephyr servers. All other linked content is retrieved by the HTML display engine off line from the zephyr service. Figure 6-2 shows an example of an HTML message sent and received by ZephyrNT. This is a copy of the HTML text used to display MIT's home page at http://web.mit.edu as of the date appearing in the message. The graphic at the top of the message was not embedded, but is linked through MIT's web site. Furthermore, HTML is much more tolerant of versioning as the designers of HTML were aware of possible disparities among HTML users. Though HTML still has the problem that it is not very readable in the raw, at least HTML display engines are generally more tolerant of badly formed HTML than the RTF display engines proved to be. This probably results from the basic assumption that HTML has to date mostly been written by hand, while RTF is almost exclusively authored through tools.


jthunter@ATHENA.MIT.EDU

Authentic PERSONAL Message
at 01:40:10 on Mon May 05 from
I am a doctor not a programmer!
<jthunter>
on piccolo.lcs.mit.edu to
jthunter

spotlight - Photographs by Felice Frankel at the MIT Museum's Compton Gallery

about mit - facts and figures
academics - admissions, schools, courses
research - labs and centers
administration - offices and programs
resources - services, computing and publications

groups - communities of people
news - events, commencement
map - how to get around the institute
search - find people and pages

77 Massachusetts Avenue · Cambridge, MA 02139-4307 USA · 617.253.1000

Figure 6-2: HTML Popup

It is interesting to note that the message shown in Figure 6-2 is fully browsable. The links provided in the HTML text can be clicked on and followed. The message window will automatically size itself appropriately for the content in it. If the content is too large to comfortable fit in a popup, the popup will assume a standard size and display scroll bars. In fact, the display engine used here is an embedded Internet Explorer Object. This means that explorer's full range of features could be exported to the user through the popup, including forward and backward movement through pages. In addition, zephyr messages can contain embedded active content like Java applets or ActiveX controls. However, linked objects like these affects the display rate of the message. The popup would appear on the users screen, and the display engine (IE) would download the linked content when it becomes active. The user would have to wait until this content was appropriately loaded.

HTML does have its drawbacks as a display medium. Though the above example demonstrates that HTML offers a very rich display engine, there are as yet no embeddable authoring tools. This is likely to change as early as the next year or two as the use of HTML grows with business interest in the internet. Until then, however, users need to author their own HTML. This does not present a problem for more advanced users who have become familiar with HTML, but it violates the Windows design goal that was a guiding principle in this project: things should be as easy for the inexperienced user as they are efficient for the advanced user. Inexperienced users have a great difficulty authoring their own HTML. Though the display engine is tolerant of badly formatted HTML the results produced by such incorrect formatting is highly undesirable and may discourage beginners from learning and using ZephyrNT.

The second, perhaps more dramatic, drawback to using HTML is surprisingly the rich display engine. Internet Explorer (IE) presents a number of problems as the display architecture for ZephyrNT. If the user of ZephyrNT does not utilize Internet Explorer as his default browser, its display engine may not be installed on his system. This was not true for the RTF display engine because this component is included as part of the operating system. Distribution and installation of IE along with ZephyrNT is not a trivial matter. The current IE distribution alone is about 9 MB. That is almost five times the current size of the entire ZephyrNT package including MFC libraries. In fact, the ZephyrNT program and its associated libraries without the MFC libraries is only about 0.5 MB. This means that the code to display a popup message is almost 20 times the size of the code necessary to operate the rest of the system! If the disk drive footprint (and download speed) of IE does not make it undesirable for ZephyrNT distribution, consider its load speed and memory footprint. IE can take tens of seconds to load upon first execution. The non-beta-locked release of ZephyrNT takes less then a second. IE's typical memory footprint is approximately 10 MB for average usage with active content. ZephyrNT's is typically between 1.5 and 2 MB. The cumulative effect of these statistics is that the display engine for popups completely dwarfs the size and usage constraints of the rest of the system.

In addition to these bloating concerns, IE is not always a friendly coexisting program. That is, users who prefer to use another browser, such as Netscape Navigator, may find themselves hassled by incompatibilities problems. Having IE and Navigator installed on the same system can dramatically decrease the stability of the system. Even if the user uses Navigator exclusively for browsing and IE only for zephyrgrams. Finally, IE is frequently very slow. Such sluggishness can have the drawback that a user receives zephyrgram messages faster than IE is able to display their content. This can have a drastic limitation on the flow of conversation through the Zephyr System. A good analogy to this problem is telephone communications via satellite. The noticeable delay between conversation partners is distracting and slows communication.

For these reasons, no release of ZephyrNT with HTML support has ever been made available to the public beta test group.

7. Future Work

The conclusion of this thesis leaves ZephyrNT in a intermediate state. To meet the ultimate goals of the project, additional work needs to be done. This additional work is beyond the scope of this document; however, a brief introduction to this work is outlined below. The first section discusses why it would be advantageous for the zephyr server to be ported to Windows NT. The second section, assuming completion of the first, outlines a proposed solution to the content extensibility issues discussed throughout chapter 6.

7.1 NT-based Server functionality

For ZephyrNT to achieve its goals of power and simplicity, it is necessary for the entire system to find its way onto Windows NT. The server component remains on the UNIX platform. Conversion of the server component, however, would entail redesign of the message/packet relationship as well as the authentication and ACL services. Messages are currently sent through packet distribution, however, a more object-oriented delivery service could be used if the server component were an NT application. In particular, DCOM technology and Java Beans have been investigated as mechanisms for message encapsulation. Furthermore, the current authentication model utilizes the Kerberos service. This service is available in the MIT environment, but is not widely supported outside this environment. It is, therefore, desirable that ZephyrNT should use Windows NT's authentication model. This would allow authentication directly by the local NT server. ACL's for server management and instance access could be managed through NT groups rather then separately by the zephyr server allowing for more uniform ACL management through NT's User Manager program and distributed management through the Cairo Directory Service. Backward compatibility with existing UNIX servers is desirable, but this can likely be achieved through a gateway service. Such an architecture would decouple backward compatibility issues with future development efforts.

7.2 A Successful Solution to Content Enhancement

Though ZephyrNT and its associated library provide an object-oriented interface to interact with the Zephyr System, the system itself is not object-oriented. Messages are transferred via network packets, as all network traffic is. However, the zephyr servers are directly aware of the networking medium and its limitations. At the time of Zephyr's original design this may have been necessary to achieve appropriate efficiency. It would be desirable to remove such lower level networking concerns from the zephyr code. Object-oriented technologies that allow remote communication of objects have recently matured to a functional state. These technologies need to be evaluated as potential vehicles for zephyr messages. The critical issues that must be evaluated are speed, efficiency, cross-platform compatibility, and ease of integration into each target platform. DCOM (OLE/ActiveX) currently appears to present the best cross section of these parameters. It offers high speed, reasonable efficiency, and ease of integration. However, DCOM is basically limited to Windows NT as a implementation platform and is significantly more complicated to program in than other solutions. On the other hand, Java Beans offer great cross-platform compatibility and a very simplistic programming model. However, Java currently has questionable speed and efficiency and terrible integration. Another possibility may be CORBA which has recently seen a growth in interest.

The benefits of object technology for encapsulating messages are two fold. First, as discussed above, object encapsulation can help decouple the network layer from the zephyr service. However, more importantly, object encapsulation can provide the means for content universality by allowing different content types to associate their own display code. This code can be dynamically downloaded and installed when a message of a new content type is received. This allows ZephyrNT to adapt to the growth of content types. All of the problems remaining from Chapter 6 can be addressed via object encapsulation of content.