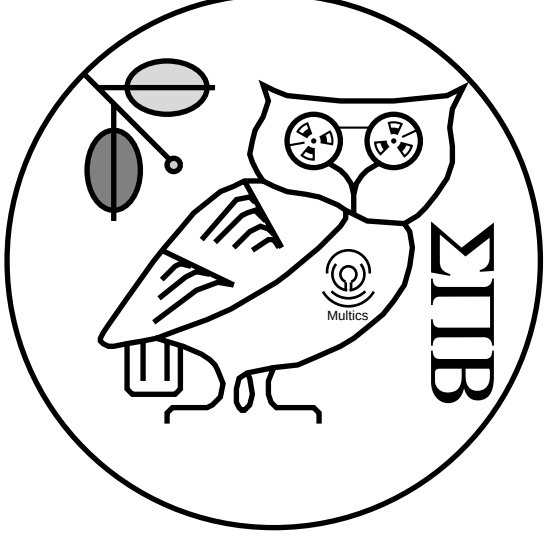

Inessential Zephyr

The Student Information Processing Board
Abbe Cohen
Heather Anne Harrison
Alessondra Springmann
Arun Tharuvai

June 28, 2008

Contents

1 Introduction to Zephyr

Zephyr was a system designed to let system administrators send important messages to users in an easily noticeable format. It was meant to have a low volume of traffic and be used only for official notices. This is obviously not what Zephyr is today. It is still used in the way it was intended: notice that you get official zephyrgrams as you log in, with important information about Athena services and planned outages. However, the most common usage is by average users exchanging information about classes, how their days are going, and talking on Zephyr classes and instances about everything from the latest episode of Battlestar Galactica to the next 6.01 problem set. The usage of Zephyr has far exceeded original expectations. Over time, people have also created programs that give Zephyr a graphical interface, and programs that give zephyr a purely textual interface, that can be used entirely within a ssh terminal. Some of these *zephyr clients* have become so widely used that there are users who do not know that there are other ways to send (and receive) zephyrgrams. This document will cover the traditional commands, typed at the `athena%` prompt, as well as the more common modern zephyr client **BarnOwl**. Other programs, such as `xzewd`, will be mentioned later in this document. They are not Zephyr, but are graphical programs that execute Zephyr commands.

Zephyr allows you to communicate with other users in real time. Zephyr messages appear on your screen as windows, called windowgrams, containing text and information about the sender. If you are using a modern zephyr client, the zephyrgrams will appear in other ways. Zephyrgrams can only be received when you are logged in and you can only send zephyrgrams to other users who are logged in and subscribed to messages. However, it has become common for BarnOwl users to leave themselves permanently logged in, or "permalogged", by having a zephyr client that is always receiving messages, even when they are not logged-in to read them. Text-based zephyr clients, such as BarnOwl, and the UNIX utility `screen(1)` (that allows for programs to continue running when you log out, and be resumed (reattached) when you log in again), have made this usage possible.

2 Getting Started

Here is a quick rundown of the basic commands and terms related to Zephyr. They will be described in greater detail in later sections.

2.1 Basic Terms

- **windowgrams** are the small windows that appear on your screen with a message from another person.
- **Zephyrgrams** are the messages that you send and receive when you are using Zephyr. They can appear as windowgrams or in other forms, depending on which client you are using.
- A **Zephyr client** is a program that lets you receive zephyrgrams. The `zwgc` client starts automatically when you login to an athena workstation; you can start BarnOwl manually with `add barnowl; barnowl`
- **Zephyr** is the name of the whole system that makes all of this happen.

2.2 Basic Commands

Basic Barnowl Commands

BarnOwl is normally in a viewing window where you can scroll through the history of already-received messages with `n` and `p` (next and previous), or the up and down arrows. To use many of the commands, you will need to enter command mode, which is done by typing a colon character (`:`). Many common commands have shortcut keys that will enter command mode and fill the beginning of the command.

zwrite *username* will bring up a composition window to send a zephyrgram to the user *username*; a period (`.`) on a line by itself will send the message.

subscribe *classname* *** *** will subscribe you to the zephyr class *classname*, so that you will receive the messages sent there.

You can also **zwrite -c** *classname* **-i** *instance* to send a zephyrgram to class *classname*, instance *instance*.

You won't have to type this every time you want to send a zephyrgram, though — you can navigate to the message you want to reply to, in the view window, and then just hit the `r` key, which will bring up the correct composition window to reply to that message. If you want to reply to just the sender of a message, and not the class and/or instance, use a capital `R` instead.

To find out more details about a particular message, navigate to it in the view window, and press `i` (for information) – a window will pop up with many fields, including the "host" (which computer it was sent from) and the full timestamp.

If you want to explore more of the commands available in BarnOwl, you can use the **help** command, which will bring up the on-line help window, which is a quick reference for common keystrokes as well as a place to start looking for more detailed commands.

Basic zwgc Commands

zlocate *friend* is the command used to find out if user "friend" is logged in and subscribing to zephyrgrams. If they are logged in and subscribing to messages you will receive information about where they are logged in. If they are not logged in you will receive the message "Hidden or not logged-in." This means they either do not want to be found or are not logged in.

zwrite *friend* is used to send a message to `friend`. Just follow the instructions given. If you get an error saying the person is hidden or not logged in then your message has not been sent and the person you are trying to reach is not logged in or is not subscribing to messages and you should try sending e-mail instead.

zctl hide can be used to "hide" yourself. When hidden you are not `zlocatable`, but if someone tries to `zwrite` you anyway they will succeed.

zctl wg_shutdown should be used if you want to stop receiving zephyrgrams for this session.

zctl set zwrite-signature "foo" (quotes are mandatory) will change your Zephyr signature, or zsig, to `foo`. By default your zsig is your name as it exists in your finger information. It shows up in a zephyrgram before your username. You can change it to almost anything you like, although you should avoid very long zsigs since they tend to annoy people.

zaway is used to let people know you are away from the terminal and not deliberately ignoring their messages. It sends a message to whoever sends you a personal zephyrgram that lets them know that you are away (and will probably respond later).

zwgc -ttymode will start up a Zephyr client when you are logged in remotely. Zephyrs appear as plain text on your screen.

znol will let you know which people on a list are logged in. Your `~/.anyone` file should contain the list of usernames you want to know about (it should have one name per line and no spaces). You will also be sent login and logout notices in the form of a zephyrgram whenever one of the users in your list logs in or out (if they are announced, see below) after you have run `znol` during a session.

zctl set exposure exposurelevel will set your exposure (how other people know when you are logged in). An exposure level of `net-announced` causes login and logout notices to be sent to people who have you in their `.anyone` file, and you will be zlocatable. `net-visible` is the same except login and logout notices are not sent. The `realm-announced` and `realm-visible` settings require authentication before your information is divulged, but behave in most situations in the same way as `net-announced` and `net-visible`, respectively. The `opstaff` setting makes you unable to be zlocated and does not send login and logout notices. Finally the `none` setting provides no information about you and **you will not be able to receive zephyrgrams.**

zctl sub message foo * will subscribe you to a Zephyr "instance" named `foo`. Zephyr instances (and classes) allow groups of people to have conversations via Zephyr. The above `zctl` command will subscribe you to the instance `foo` for your current login only; to make it more permanent replace `sub` with `add`. To unsubscribe for this login only change `sub` to `unsub`, and to unsubscribe permanently use `delete` instead.

zwrite -i foo will send a message to the Zephyr instance `foo`.

zctl sub foo * * subscribes you to the Zephyr class `foo`. Zephyr classes are slightly more private than instances as you must know the name of the class to subscribe to it. `unsub`, `add` and `delete` work the same way for classes as for instances.

zwrite -c foo sends a zephyr(gram) to class `foo`.

3 Locating Other Users

To find out if another user is logged in and `zwrite`able, you can use the following command:

`zlocate` *username*

which will either give you the name of the machine the user is logged in on, or tell you that the user is `Hidden or not logged-in`. The latter message indicates that the user is either not logged in, is not subscribing to zephyrgrams, or does not want to be `zlocate`d.

You can check whether or not your friends (and enemies!) are logged in using the `znol` command. First, you need to create a file called `.anyone` in your home directory. This file should contain a list of the usernames of all the people you want to look for (friends and enemies alike), with one username on each line and no spaces. Then, you can type

`znol`

at your `athena%` prompt. You will see a list of all the people in your `.anyone` file who are logged in. You must be logged in to use the `znol` command, with the exception of `znol -l` (list) command. The `znol -l` command may be used at any time and will not subscribe you to login and logout zephyrgrams or require that you be subscribed to messages.

The `znol` command will also allow you to get zephyrgrams announcing when people in your `.anyone` file log in and log out, if they expose this information. You can use `znol -q` to receive login and logout zephyrgrams without listing who is currently logged in.

Another use of `znol` is

`znol -u` *username*,

which allows you to just check one username and get zephyrgrams when that person logs in and out. This differs from just zlocating that person because you will get login and logout zephyrgrams. It can be done in addition to running `znol` normally, so you can basically add individual users to the list of people you want to check on (in this session only) without adding them to your `.anyone` file.

You can use the `-f` option to use a file other than `.anyone` as the list of people. For instance, you might list the usernames of members of a club in a file called `.anyone.club`, and check if any of them are logged in with `znol -f ~/.anyone.club`.

4 Being Located by Other Users

Your exposure setting determines who can zlocate you or receive login and logout notices about you. There are several options for exposure settings that will be explained in detail below. The command to change your exposure is

`zctl set exposure` *exposurelevel*

where *exposurelevel* is the setting you have chosen, which may be replaced by any of the exposure settings mentioned in this section. The `net-announced` setting provides the most information to the largest number of users. Anyone (with or without Kerberos tickets) can find out if you are subscribed to Zephyr, and those people who have you in their `~/.anyone` file will receive login and logout notices about you. This means that people using the www zlocate gateway at http://www.mit.edu/zlocate, and those using Zephyr at other realms (e.g. ANDREW.CMU.EDU, FSCK.COM) will be able to zlocate you in addition to people who are logged in with valid Kerberos tickets in the realm ATHENA.MIT.EDU .

The setting `realm-announced` is like `net-announced` except that only people with Kerberos tickets in the realm ATHENA.MIT.EDU (which you receive upon login to your Athena account) can check your online status.

`net-visible` is in practice almost identical to `net-announced`. `realm-visible` is just like realm-announced, except that no login or logout messages are sent.

The `opstaff` setting means that only Operations Staff are able to access information about you. You are not `zlocate`able, nor are login or logout announcements sent when this setting is used.

Lastly, the `none` setting does not allow information to be given out in the form of login or logout notices or through `zlocate`. It also is the only setting in which you are **not** able to receive zephyrgrams.

5 Sending Zephyrgrams

As mentioned in section three, sending a zephyrgram is done by using the `zwrite` command. However, it is possible to customize the zephyrgrams that you send. There are also several useful sending options that have yet to be mentioned.

5.1 `zwrite` Options

Sending a zephyr to multiple recipients it is possible by the command `zwrite` *friend enemy*. This sends the same Zephyr to users *friend* and *enemy*.

`zwrite` *friend* `-m Hello friend`

sends a message to user *friend* with the message "Hello Friend". The `-m` argument must be the last one, as the message text extends until the end of the line.

`zwrite -C` *friend enemy*

sends a message to *friend* and *enemy* and places the line "CC: friend enemy" at the top of the message.

`zwrite -n` *friend*

sends a message to *friend*, but does not ping friend to make sure he is logged in before attempting to send the message. This is slightly faster, especially when the Zephyr system is being used heavily.

If you do not like waiting for the *message sent* notification after you have sent a Zephyr, it is possible to suppress this information by using the `zwrite -q` option.

5.2 zsigs

You can send zephyrgrams with different "zephyr signatures," or "zsigs." Your zsig is on the line of a zephyrgram that says "`From:` *`zsig username`*". Unless you have changed it, your zsig will be your name as it is listed in your finger information. You can change it to read just about anything you want by using the command

`zctl set zwrite-signature '`*mysig*`'`

which will change your zsig permanently until you use the command again. You can also use

`zwrite -s '`*mysig*`'` *username*

to send a zephyrgram with the zsig *mysig*, for only that one zephyrgram.

5.3 All Shapes and Sizes

You can change the font, size, and appearance of the text of a zephyrgram. You can send zephyrgrams with boldface, italic, and different-sized of text. To send part of your message in a large font, you would format it like this:

`@large(`*message*`)`

Only the portion of the message that you enclose in parentheses will be in large text, so you don't have to make the entire message the same size or style of text. You can use any form of bracket to enclose the portion of the message that you want to be affected, including `[ ]`, { }, and `( )`. In addition to `@large` you can use `@small` or `@huge` to affect the size of text in your zephyrgrams. To use boldface or italics, you can use `@bold` (or `@b`) and `@italic` (or `@i`). The formatting for all of these is identical to the example above, replacing `@large` with the typeface you want to use.

You can also use different fonts in zephyrgrams. The three fonts that are readily available for zephyrgrams are Courier, Times, and Helvetica. Courier is the default font. To use a different font, you can use `@`*`fontname`*`(`*`message`*`)` in the text of a message. For example, to send the message "hi!" in the Helvetica font, you would type `@helvetica(hi!)` as your message.

You can also send zephyrgrams with color text, using a slightly different formatting technique. To use color, format your message by typing `@color(`*`colorname`*`)`*`message`*. For example, if you wanted the message "I like blue" to be printed in blue, you could use `@color(blue)I like blue` as your message. Unlike fonts, sizes, and typefaces, this will make the rest of the message stay that color, unless you switch it back with another `@color` in the same message. Another way to have color text limited to a portion of the message is to format it like this:

`I like @(@color(purple) purple), but I don't like @(@color(green) green).`

If you sent this as your message, the word purple would appear in purple and green would appear in green. The colors available vary by system architecture; a full list can be found in the file

`/usr/openwin/lib/rgb.txt` on machines running Solaris, and in `/etc/X11/rgb.txt` on linux-athena. Note that some colors will be invisible on some backgrounds, especially on monochrome screens, so you risk sending messages which are unreadable.

You can also combine font changes and size changes into your zsigs by using these same formatting commands when you set the zsig. You should be especially cautious when using colors in your zsig, since, as mentioned above, some colors may be unreadable on certain displays. If you use color formatting without the extra parentheses, your username and message would also be the same possibly invisible color, and other people wouldn't be able to tell who was zwriting them. If you decide to still use color zsigs and risk them being invisible, you should use the format with the additional parentheses, which we will repeat here. You would use something like

```
@(@color(blue)I like blue zsigs)
```

and make this your zsig with `zctl set zwrite-signature`.

It is important to emphasize that color in zsigs can cause legibility problems on monochrome screens or screens that have colorful backgrounds. In general it is best to use special colors and fonts as little as possible, for these reasons, and because it tends to annoy some users.

5.4 Langwrite

You can send Zephygrams with accents and other symbols used in some foreign languages with the help of a program called `langwrite`. Langwrite is in the `potluck` locker, so in order to access it you would need to type

```
add potluck
```

To find exact details on how langwrite works, you should type

```
man langwrite
```

Then, to use langwrite with Zephyr, you would type

```
langwrite | zwrite username
```

to send the output of the langwrite program over Zephyr as the message. On some remote connections characters produced by langwrite may display incorrectly or not display at all.

6 Recieving Zephyrgrams

Receiving zephyrgrams is rather simple. Zephyrgrams are displayed to your screen as long as you have zwgc running and your exposure set properly. It is possible, however, to customize your account such that zephyrgrams are sorted and appear in different places, colors, and sizes. It is also possible to display more information than the default settings show, and to ignore specific users. Another option is to log (save to a file) all the zephyrgrams that you receive. This can be especially useful when logged in remotely over a poor connection, or in conditions where zephyrgrams may dissapear before you have a chance to read them.

6.1 Colors and Locations

The colors and locations of your incoming zephyrgrams can be changed in two ways: through your `.Xresources` file, which can be used to customize all of your X applications, and through your `.zwgc.desc` file, which describes only your zephyrgrams. People often change the color and locations of zephyrgrams from different Zephyr classes and instances, including things like class **login** (for `znol` login and logout announcements), and class **mail** (for announcements of new mail).

To change where zephyrs appear, it is useful to first understand the way that geometry is specified in X. This is best explained by example. The top left corner of the screen is +0+0. The first number of this is the X coordinate and the second the Y coordinate. The right edge of the screen corresponds to an X coordinate of -0, and the bottom edge to a Y coordinate of -0. The lowercase letter c refers to the center of either the x or y axis. You can also use any integer to refer to a location on the screen. +0-20, for example, is near the bottom left corner of the screen, 20 pixels up from the bottom, while -100+100 is 100 pixels away from the right side of the screen and 100 pixels down from the top of the screen. In summary, positive numbers measure from the left and from the top, and negative numbers measure from the right and from the bottom; the horizontal position is the first number and the vertical position is the second number.

The basic format of an entry in your `.Xresources` file to change the geometry (location) of incoming zephyrgrams is:

`zwgc*style.`*class.instance*`*geometry:` *+x-y*

where *+x-y* is any combination of integers, as described above. Some examples follow:

```
zwgc*style.message.personal*geometry:          -0+0
zwgc*style.message.white-magic*geometry:       +c-0
zwgc*style.login*geometry:                     +0+0
zwgc*style.mail*geometry:                      +0+30
```

These four lines place personal zephyrgrams in the top right corner of the screen, zephyrgrams on instance **white-magic** in the bottom center of the screen, login and logout zephyrgrams (from `znol`) in the top left corner of the screen, and new mail notification zephyrgrams just below login and logout zephyrgrams in the top left corner of the screen.

To do something similar by editing your `.zwgc.desc` file you would need to have lines something like this (we have included comments in this code with #'s, to make clearer what it is actually doing since you will need to alter this significantly to meet your needs. Any line beginning with a # is a comment, and your zwgc will ignore it (whereas you should pay attention to it):

```
# looks at the name of the instance after converting it to lowercase
# since .zwgc.desc is case sensitive
case downcase($instance)
# things to do if instance name is white-magic
match "white-magic"
        set X_geometry = "+c-0"
# things to do if instance name is personal
match "personal"
        set X_geometry = "-0+0"
# stops looking at instances
endcase

# looks at the name of the class after converting it to lowercase
# since .zwgc.desc is case sensitive
case downcase($class)
# things to do if class name is login
match "login"
        set X_geometry = "+0+0"
```

```
# things to do if class name is mail
match "mail"
        set X_geometry = "+0+30"
# stops looking at classes
endcase
```

This code has the same effects as the `.Xresources` entries listed before. Although it is somewhat longer, some people opt to use the `.zwgc.desc` method because they can include additional customizations (other than position) to `.zwgc.desc` for those particular classes and instances at the same time. For example, you can change the way a message is formatted. The best way to learn how to do this is to look at other people's `.zwgc.desc` files and examine what the various commands are doing. Some examples can be found in the `dotfiles` locker. One particular `.zwgc.desc` file you might want to look at is rjbarbal's, which you can find in `/mit/dotfiles/Zwgc.desc/rjbarbal.zwgc.desc` (you need to type `attach dotfiles` first).

You can change the colors of incoming zephyrgrams by modifying your `.Xresources` file. The basic format of `.Xresources` entries for foreground and background colors for zephyrgrams is:

`zwgc.style.`*class.instance*`*background:` *color*

`zwgc.style.`*class.instance*`*foreground:` *color*

To avoid problems when you don't have a color screen, you can place your color preferences after a line reading

`#ifdef COLOR`

and follow them with a line reading

`#endif`

An example follows:

```
#ifdef COLOR

zwgc.style.login*background:                          LawnGreen
zwgc.style.login*foreground:                          MidnightBlue
zwgc.style.message.white-magic*background:            purple
zwgc.style.message.white-magic*foreground:            white
zwgc.style.message.personal*background:               blue
zwgc.style.message.personal*foreground:               yellow
zwgc.style.mail*background:                           red
zwgc.style.mail*foreground:                           black

#endif
```

These lines would cause login announcements to show up as MidnightBlue on LawnGreen, zephyrgrams in instance**white-magic** to show up as white on purple, personal messages to show up as yellow on blue, and mail notifications to show up as black on red.

6.2 Showing Pings

When someone sends a Zephyr to you a "ping" is sent to check whether you are logged in. As a default these pings are not shown on your screen, but it is possible to have them displayed by making alterations to your .zwgc.desc file. Information on how to do this is in section 9.2.

6.3 How To Ignore People

People often express interest in hiding from a specific user, so that *eviljoe* can't zwrite you and doesn't know that you are logged in, but everyone else can zwrite you. Zephyr is not actually capable of this exact functionality, but it comes close. You can not set it up so that Zephyr will give a *yourusername:* `not logged in or not subscribing to messages` error message to *eviljoe* and not to everyone else. However, there is a way to edit your .zwgc.desc to filter out zephyrgrams from any users you specify and to send them an automatic message informing them that you are not receiving their messages. Again, the relevant `.zwgc.desc` lines will be explained with comments in the code. Here is the code:

```
# punting a user
# zctl set username punt
# zctl unset username

# take the sender's username, and assign it to a variable called test
        set test = zvar($sender)

#if the sender's username has been set to punt
        if ($test == "punt") then

# if it is also a personal Zephyr sent only to you
        if (upcase($instance) == "PERSONAL" && upcase($class)=="MESSAGE") then

# send them a message, and don't print the Zephyr
# you should not have a return between the -m and the message
        exec "zwrite" "-q" "-n" $sender "-m"
        "Your Zephyr is not being received at this time."
                        exit
                endif
                exit
        endif
# exit from the .zwgc.desc without printing any zephyrgrams
```

For your changes to `.zwgc.desc` to take effect without logging out and logging in, you will neeed to type `zctl wg_read` to re-read the configuration file. Then, in order to "punt" a particular user's zephyrgrams, you can type

`zctl set` *username* `"punt"`

and to "unpunt" them (see their zephyrgrams again and stop ignoring them), you can type

`zctl unset` *username*

The basic methods used in this code sample allow you to make your own Zephyr variables and then set them from your `athena%` prompt using `zctl set`. If you want a customization to only happen when you explicity say so, you could make up the name of a variable, and use `zvar` to make it a variable. You could have `if` statements in your `.zwgc.desc` that will execute the customization for certain values of that variable.

In your `.zwgc.desc`, before you use a variable, you need to declare it, thusly:

`set` *variable* `= zvar("value")`

Then, to change its value, you would type

`zctl set` *variable value*

Note that if you want to set variables to non-numerical values like "punt" or "true," you will need to quote the word both when you set it with `zctl set` and when you test it in `.zwgc.desc`.

6.4 Logging your Zephyrgrams

You can log zephyrgrams to a file as well as receive them on your screen, with only a few simple changes to your `.zwgc.desc` file.

If you want to save the complete zephyrgram (everything you see on your screen), you should add the following things to your `.zwgc.desc` file. First, near the beginning of the file, before any of the code that actually places and formats zephyrgrams, you need to add the line

```
appendport "outf" getenv("HOME")+"/.zlog"
```

which tells `.zwgc.desc` that "outf" refers to a file in your home directory called `.zlog`.

Then, wherever you want a particular kind of zephyrgram (like personal zephyrgrams, or all zephyrgrams to a particular instance) to be recorded in the file, you would need to find where those zephyrgrams are specified in the `.zwgc.desc` file and add the command

```
put "outf"
```

after the line (that should already be there) that says just `put`.

Another thing that can easily done using this infrastructure is to log Zephygrams only if a certain variable is set. To do this, instead of just adding the `put "outf"` line mentioned above, you would add several lines of code. Near the beginning of your .zwgc desc, you would put the line:

```
set log = zvar("log")
```

Then, in the locations where the zephyrgrams you want to log are mentioned, instead of just putting `put "outf"`, you would add the lines

```
if ($log == "true") then
put "outf"
endif
```

Then, to start logging zephyrgrams, you would need to type

```
zctl set log "true"
```

To stop logging them, you would need to type

```
zctl unset log
```

7 Zephyr Classes and Instances

Zephyr classes and instances allow groups of people to have conversations via Zephyr. Zephyr classes are slightly more private than instances since you must know the name of a Zephyr class to be able to receive zephyrgrams directed to it. Zephyr classes have sub-channels called instances. When someone talks about a Zephyr instance they usually mean an instance of the Zephyr class `message`, although all Zephyr classes have instances. Since class `message` is widely known, and it is possible to subscribe to class `message`, the instances of class `message` are not very private. Instances of class `message` are good for public discussions that are of interest to many people. For (slightly) more private conversations amongst a group of friends, a Zephyr class of some other name is probably preferable.

There are three parameters that control who will get a zephyrgram: a class, an instance, and a recipient. When you send a zephyrgram to a friend, using `zwrite` *username*, the class is **message**, the

instance is **personal**, and the recipient is your friend's username. These parameters can be changed to allow conversations among a group of people via Zephrgrams. What people call a Zephyr class is a particular set of zephyrgrams with the class parameter set to something other than **message**. What people call a Zephyr instance, or public instance, refers to the set of zephyrgrams in class **message** with a particular value for the instance parameter. In the next few sections, we will explain how to change the class, instance, and recipient parameters, to send and receive zephyrgrams to groups of people through different classes and instances.

7.1 Subscribing to classes and instances

When you are subscribed to a class or instance, you will see messages sent to that class or instance. There are two commands to subscribe to a class or instance:

`zctl sub` *class instance recipient*

which lasts for your current login session, and

`zctl add` *class instance recipient*

which adds this subscription permanently.

To reverse `zctl sub`, you can use `zctl unsub`. If you want to reverse `zctl add`, you can do it permanantly using `zctl delete`. To unsubscribe temporarily to a class or instance that you are currently subscribed to using, you can use `zctl unsub`

One additional way to subscribe and unsubscribe to Zephyr classes and instances is to modify your `.zephyr.subs` file directly. The commands `zctl add` and `zctl delete` make permanent changes because they alter the list of subscriptions in this file. The format of a subscription in the `.zephyr.subs` file is *class,instance,recipient* (note that there are no spaces between words). To update Zephyr subscriptions directly from this file, you must use the command `zctl load`. If you want to list other Zephyr subscriptions in a file other than `.zephyr.subs`, you can load those subscriptions using `zctl load ~/`*filename*. Only the subscriptions in your `.zephyr.subs` file are read when you first log in and run Zephyr; if you want to read subscriptions from another file automatically, you would need to put the `zctl load ~/`*filename* command in one of your startup files.

7.2 What to Subscribe to

Now that you know the various ways to set up subscriptions, some specific examples of subscribing to instances and classes will be described.

Classes

Zephyr Classes are considered to be private because there is no * wildcard option for the class field. You must subscribe to a class explicitly. If you try to subscribe to a class using `zctl sub \* \* \*`, you will be subscribed to all the instances on the class named *****. However, the lack of this wildcard does not guarantee absolute privacy on a Zephyr class. Listening in on a Zephyr class is as easy as figuring out what the name of the class is and then subscribing to it.

Classes are a good to use for things like having a group of friends chat, or to send zephyrgrams to a number of people, perhaps about a group project. If you and your friends wanted to use a Zephyr class to chat, you could make up a name like **myfriends** (you would want to choose something private, that other people would probably not be using already) and then type

```
zctl sub myfriends \* \*
```

(Again, you could use `zctl add` instead.)

The * for the instance means you will be subscribed to all instances on class **myfriends**. This means that your friends could write to a specific instance within class **myfriends**, perhaps to distinguish what you are talking about, or just to be silly, and you would receive the message regardless of which instance they decided to use.

One class you might want to subscribe to allows you to receive zephyrgrams from the post office servers when you get new mail. To subscribe to this class, you should use `zctl add mail inbox`. You probably want to use `add` rather than `sub` so that you will receive these zephyrgrams whenever you are logged in. In your `.zephyr.subs` file, this will appear as the line `mail,inbox,%me%`, indicating that you are only subscribed to messages on this class with your username as the recipient, and on the instance *inbox*.

Another class that you might be interested in subscribing to is class help. As its name suggests, it is commonly used for the purpose of asking questions about various topics. To subscribe to it, type at the athena prompt:

```
zctl sub help \* \*
```

Topics on this zephyr class are threaded by instance, using the topic name. After subscribing, in order to send a message, you can type (at an Athena prompt:)

`zwrite -c help -i` *topicname*

Instances – Class Message

Class **message** is special. When someone talks about a Zephyr instance they probably mean an instance of class **message**. Just like other classes, it is possible to subscribe to all of the instances of class message (many people do). This means that zephyr instances are very public and are not a good place to discuss anything you do not want made public..

To subscribe to the particular instance **help**, you would type

```
zctl sub message help \*
```

In any of these examples, you can also use `zctl add`. This would give you a permanent subscription, in this case to the instance **help**. A permanent subscription to the instance **help** would show in your `.zephyr.subs` file as the line `message,help,*`

Looking at the above example of a subscription to instance **help**, the * in the recipient field indicates a broadcast message. (You need to type a \ before the * so that it is interpreted correctly by the shell.) In other words, the * means that means that you are receiving zephyrgrams that have been sent to everyone who subscribes to that instance. The other option for recipient in a Zephyr subscription is `%me%`, which indicates that you are subscribed to zephyrgrams sent specifically to you. This is generally not useful on an instance such as **help**, which is intended to send messages to a large number of people, rather than a specific user. The **help** instance is used by people who want to ask a question, about anything ranging from using Athena, to progamming, to random facts about MIT, Boston, history, etc. If you just want to ask a question, you probably want to use `zctl sub` to ask it and see the answer, since there are a lot of zephyrgrams on this instance, and you may not want them distracting you when you don't have a question to ask. Many knowledgeable people are subscribed to the instance **help**.

Public instances are not limited to the instances people often write to. Any public instance (i.e. on the class **message**) that someone chooses to write to exists when they decide to write to it. Obviously, you can't explicitly subscribe to each one of these, but you can explicity subscribe to instances that you know of. Some others besides **help** are **white-magic**, which is an instance for random chatting,

b5, which is an instance where people chat about Babylon 5, and instances like **6.170**, where people talk about that particular subject.

You can also subscribe to every possible public instance by typing `zctl sub message \* \*`. In this case, the `*` for the instance name acts as a wildcard which subscribes you to all instances in the class **message**. The `*` wildcard for instances is the reason that instances on class **message** are called public—anyone who chooses to subscribe to `message \* \*` will get zephyrgrams sent to any instance you might send to, so there is no way to keep people from finding out about the instance and jumping into a conversation on it. There are actually a number of people who are always subscribed to "star star," as it is often called.

If you do want to subscribe to `message \* \*`, there are a few useful things you might want to know. First of all, a word of warning: you will receive a **lot** of zephyrgrams, which can be very confusing if you do not customize your dotfiles to make zephyrgrams on instances look different from personal zephyrgrams, and change where they appear on your screen. It is also rather distracting. To decrease the volume of zephyrgrams somewhat, you can filter out zephyrgrams on certain instances that you have no interest in reading by creating "negative subscriptions" in your `.Zephyr.subs` file. For each instance you want to filter out, you would add a line reading `-message,`*silly-instance*`,*` (where *silly-instance* is the name of the instance you want to filter out). You would then be subscribed to all public instances except those listed with a minus sign in your `.zephyr.subs` file. Also, there is a command called `zpunt` in the `sipb` locker which you can use to unsubscribe from a particular instance temporarily. The `zpunt` command is necessary for people who are subscribed to "star star" because using the `zctl unsub` command will not work in this case. It works in the same way that the "negative subscriptions" in your `.zephyr.subs` file do. To use `zpunt`, you would type

`add sipb; zpunt` *annoying_instance*

If you want to resubscribe to that instance before you log out, you can type

`zunpunt` *annoying_instance*

7.3 Writing to instances and classes

`zwrite -i` *instancename*

changes the instance you are sending to from **personal** to *instancename*, and by default sends to the class **message** and the recipient *****, which was explained above. For example, `zwrite -i help` would send a zephyrgram to the instance **help**, and anyone who had subscribed would receive the zephyrgram.

`zwrite -c` *classname*

changes the class you are sending to from **message** to *classname*. By default, it will send to the instance **personal**, and the recipient *****. This means that everyone subscribed to that class would receive the zephyrgram. Using the example class **myfriends**, `zwrite -c myfriends` would send a zephyr to the class **myfriends** and the instance **personal**.

`zwrite -c` *classname* `-i` *instancename*

sends a zephyrgram to an instance within the class you are writing toinstead of to the instance **personal.** Generally, all people subscribed to a class subscribe to all the instances within that class, so sending to the instance **silly-name** in the class **myfriends** would not go to any different people than just sending to the class **myfriends**.

One thing to note about sending to various Zephyr instances and classes is that the capitalization you use does not matter. For example, `zwrite -i white-magic` is the same as `zwrite -i`

`WHITE-MAGIC`, which is the same as `zwrite -i WhIte-mAGiC`, etc. Anyone subscribed to **white-magic** will receive messages sent any of these ways.

7.4 Etiquette on Zephyr Instances

When sending zephyrgrams to instances, it is best to avoid doing things that might annoy other users who are subscribed to those instances. You should generally avoid things like sending excessively long messages, having excessively long zsigs, using the font `@huge`, using color. Using ALL CAPS is considered shouting and should be used as little as possible. Also, it is customary to send messages that are not really relevant to the discussion on an instance (for example, a joke about something said on the instance) to the instance *instancename***.d**. This is used particularly for instance **help** and instances for discussion about classwork.

7.5 Can I see that last zephyr again?

Zephyrs sent to a few commonly used zephyr classes are logged in a couple of Athena lockers. Instances of class message are logged in the zlog locker in the file `/mit/zlog/`*instance* . For example, to look at the last bunch of zephyrs sent to instance 6.170, you can type

```
attach zlog
tail -100 /mit/zlog/6.170
```

Zephyrs sent to a couple of other classes (including class help) are logged in the *zlogs* locker. All the zephyrs sent to class help since the last 3AM are located in `/mit/zlogs/help/help` . Zephyrs sent to class help on the previous day can be found in the same directory in help.0 , those from two days can be found in help.1 , etc. Additionally, class help zephyrs are also logged by instance under `/mit/zlogs/help-by-instance/`*instance* . To look at the last few zephyrs sent to class help, instance boston, you can type:

```
attach zlogs
tail -100 /mit/zlog/help-by-instance/help
```

8 Other Useful Commands

There are two four-letter words that you may notice get used a lot when talking about Zephyr. One of these, `zctl`, has appeared in this document many times already, as part of the commands to change Zephyr subscriptions, Zephyr exposure levels, and other things. The other one is `zwgc`. In this section, both the `zwgc` and `zctl` programs will be introduced, so that you can better understand how Zephyr actually works.

8.1 zwgc

`zwgc` stands for **Z**ephyr **W**indow**g**ram **C**lient. It is generally pronounced "zwig-see" or "zwigka". `zwgc` is the program that actually makes zephyrgrams appear on your screen. By default, `zwgc` is started automatically when you log into Athena on a workstation running X windows. If you log in through dialup, or for some other reason you do not have an X display, `zwgc` will not be able to start properly and you will get the error message: `zwgc: Unable to open X display -- disabling X driver`. In order to enable you to receive zephyrgrams in these cases, you can use the `-ttymode` option to `zwgc`:

```
zwgc -ttymode
```

will start the zwgc program so that you can receive zephyrgrams, but instead of trying to place them in windowgrams that require an X display, it will print them to the screen that you are typing in as text. (This text is treated only as output, i.e. it will not be read in as part of the command you were typing when you received the zephyrgram, or otherwise interfere with your terminal input.) You can also cause zwgc -ttymode to start automatically when you log in through dialup using the following command:

```
zctl set fallback true
```

This basically tells zwgc that if it can't startup in X, it should automatically fall back to zwgc -ttymode. Some people prefer to place the line

```
zwgc -ttymode
```

in their .startup.tty file, but this is less efficient, because it means that the zwgc process will try to start up in X, fail, and then start up in ttymode, each time you log in on dialup. In any case, using **both** of these things together is a mistake, as you will wind up running the zwgc -ttymode process twice and getting double zephyrgrams.

Since zwgc is the program that causes the appearance of zephyrgrams on your screen, if you ever want to completely stop receiving zephyrgrams, one way you can do this is to stop running zwgc by killing the process. You could do this by first getting a list of processes using some form of the command ps, and then searching for the process id that zwgc was started with. Also, there is a simple command in the consult locker that kills a process. To kill zwgc with it, you would type

```
add consult; punt zwgc
```

The zwgc program is also responsible for setting the appearance of zephyrgrams on your screen. When started, zwgc looks at a file called .zwgc.desc, which was discussed briefly in Section 6, to find the descriptions of how the zephyrgrams should behave. The default .zwgc.desc, which will be read if you do not create a .zwgc.desc with your own customizations, can be found in /usr/athena/lib/zephyr/zwgc.desc. An excellent example of a.zwgc.desc file can be found in /mit/dotfiles/Zwgc.desc/rjbarbal.zwgc.desc; you may need to attach dotfiles before you can access it.

8.2 zctl

zctl stands for **Z**ephyr **C**on**t**ro**l** program. It is usually pronounced "zeekootle" or "z'ctul" (Try reading zctl as a word, and however you pronounce it will probably be about right, as long as you still have your tongue.) As we have mentioned in previous sections, you can use zctl to modify your Zephyr subscriptions and to change your Zephyr exposure level. The zctl program can be called in two ways: either by typing the arguments to zctl at the command line (i.e. zctl add mail pop, or by just typing zctl at the prompt, and then typing the rest of the command. The second method allows you to type ? at the zctl: prompt and get a list of possible commands to type if you are unsure of the exact command you want to use, among other things. Other useful options in the zctl command include:

```
zctl flush_locs
```

This removes "ghost" zlocates you might have left over from being on a dialup machine. It removes all zlocate information, so you need to use zctl unhide to make yourself zlocateable again after this.

```
zctl new_server
```

This allows you to change Zephyr servers, which might be necessary if you know someone is logged in but for some reason you seem unable to zwrite them. Often this means that the Zephyr server you are using (one of a few machines that control the transmission of all the Zephyr messages on Athena)

is having some problems. Changing servers will hopefully help you work around the problems you are experiencing.

`zctl retrieve`

This lists all of your current Zephyr subscriptions, so you can tell if you have temporarily lost any of them because of problems with the Zephyr servers. Many of the subscriptions listed by this command are not zephyrgram subscriptions, but subscriptions made by the file system, which can be ignored for all practical purposes.

`zctl wg_read`

This rereads the information from the `.zwgc.desc` file and updates the behavior of the windowgrams according to it.

`zctl wg_shutdown`

This shuts `zwgc` down, so that you no longer receive any zephyrgrams.

`zctl wg_startup`

This reverses `zctl wg_shutdown` and starts `zwgc` again.

`zctl add` *class instance recipent*

This subscribes you to the *class*, *instance*, *recipient* triplet and adds it to your `.zephyr.subs` file. so that in future login sessions you will be subscribed to that triplet.

`zctl delete` *class instance recipient*

This unsubscribes you from the *class*, *instance*, *recipient* triplet and removes it to your `.zephyr.subs` file. so that in future login sessions you will not be subscribed to that triplet.

`zctl sub` *class instance recipient*

This subscribes you to the *class*, *instance*, *recipient* triplet, but only for the current login session.

`zctl unsub` *class instance recipient*

This unsubscribes you from the *class*, *instance*, *recipient* triplet, but only for the current login session.

`zctl load`

This is used to reread your Zephyr subscriptions from your `.zephyr.subs` file.

8.3 zhm

Although it is a three-letter word, `zhm`, the **Z**ephyr **H**ost **M**anager, is also important in understanding how Zephyr works. `zhm` is the program that is run on a machine you are logged into that allows your machine to communicate with the Zephyr servers. All zephyrgrams, and error messages about Zephyr, are transmitted through `zhm`. Once in a while, you may get an error message when you are logging in saying something about `Hostmanager not responding`. This generally means that for some reason, the `zhm` process on your machine is not running or is too slow to be useful. Therefore, when your other Zephyr processes try to start, they can't communicate with the Zephyr servers. If the problem persists, you can fix this by starting a new `zhm` process on your machine by typing `su` and then the root password, and then typing

```
zhm
exit
```

to exit from the root shell. (You would also need to load your Zephyr subscriptions again using `zctl load` after starting a new `zhm`).

9 A Few Customizations and Programs

9.1 zaway

`zaway` is a command that you can use to send an automatic message to everyone who zwrites you saying that you aren't there to read the message, acting as a Zephyr answering machine. Just typing `zaway` in an xterm will give the default zaway message, with a zsig of `Automated Reply:`, to anyone who tries to zwrite you while you have `zaway` running. `zaway` will only respond to zephyrgrams sent directly to you, as opposed to on instances or classes, and will not respond to other `zaway` messages. People generally run `zaway` when they are not at their workstations and want to let friends know that they are not ignoring them. Upon returning to your workstation, you can stop running `zaway` by typing Control-C in the window you started it in.

You caN customize the `zaway` messages by create a file called `.away` containing the messages you want sent to different users. Here is an example `.away` file:

```
>joeuser
Hi, Joe!  I'm not here.  Catch ya later.
>friend1
>friend2
Hello, my friend.  As you can see, I'm not at this machine right now.
>%
Hi there.  I don't seem to be at this machine,
but I'll read your message and
figure out who you are when I return.
>*
I'll get back to you when I return!
```

The line(s) following `>joeuser` will be sent in the text of a reply to joeuser. The line(s) following `>friend1` and `>friend2` will be sent in a reply to either friend1 or friend2. The line(s) following `>%` will be sent in a reply to anyone who is neither friend1, friend2, or joeuser, and the line following `>*` will be sent in a reply to everyone, including friend1, friend2, and joeuser.

Another common thing that people do with zaway is to start it automatically when running `xscreensaver`. `xscreensaver` can be found in the sipb locker; you can run it by typing:

```
add sipb; xscreensaver
```

To start `zaway` with `xscreensaver`, you can add the following in your `.Xresources` file:

```
xscreensaver*lockCommand:  zaway
```

Then whenever you leave your screen and run `xscreensaver`, the `zaway` messages will automatically inform anyone who zwrites you that you are not there.

9.2 Pings

When you send someone a message using the normal `zwrite` command, the first thing that the `zwrite` does is to send a "ping" to the person you are zwriting. This is used to check whether they are logged in. If they are not logged in, Zephyr gives you the message *username*: `Not logged in or not subscribing to messages.` To zwrite people without sending pings, you can use the option `zwrite -n`, which saves a bit of time and lessens the load on the Zephyr servers.

When people use `zwrite` without the `-n` to write to you, you can intercept these "pings" and have them displayed as small zephyrgrams, so you know who is about to send messages to you. To do this, you will need to modify your `.zwgc.desc` file, which describes how you want your zephyrgrams to

appear on your screen, to tell it to display the pings somewhere on your screen. Then you will know when people are about to zwrite you.

You can check to see if you already have a `.zwgc.desc` by typing `ls -a ~/.zwgc.desc`. If you don't, you should first copy the default `.zwgc.desc` file into your home directory, from the file:

`/usr/athena/lib/zephyr/zwgc.desc`

You should be sure that you copy it to the file `.zwgc.desc`, as the default file does not have the initial period. You probably want to have some basic understanding of how the `.zwgc.desc` file works so that you can figure out where to place modifications you want so that they will have the desired effects. Sample code to receive pings for personal zephyrgrams follows. A few modifications you might want to make might be to change the X_geometry (or leave it out altogether), which determines where to place the zephyrgram on the screen, but also leaves that setting for other zephyrgrams until you change it in your .zwgc.desc file for those cases, and to change the message printed in the ping. You might need to read through your `.zwgc.desc` file to figure out where the proper location for these lines are; `.zwgc.desc` hacking is not easy to do by "formula". Here is what you would need to include:

```
match "message"
if (upcase($opcode) == "PING") then
if (upcase($instance) == "PERSONAL" && upcase($class) == "MESSAGE") then
set X_geometry = "+0+20"
print "@b("+$sender+")"
print "@small( is about to send you a message)"
print "@beep()"
put
exit
endif
exit
endif
```

You may also need to remove a line stating

9.3 Random Zsigs

Many people have used their copious free time to devise methods of sending zephyrgrams through some sort of program that chooses a zsig randomly from a list of quotes they have chosen. In this section, I will suggest three programs that work as zsig randomizers. The first is a csh script, written by Ross Lippert, `<ripper>`. In order to use it, you would need to put the program into a file (calling it `.zrandom` would work) and add your zsigs. Then, after saving the file, you would need to make it executable using the command `chmod 755 .zrandom`, and then make an alias such as

```
alias zr '~/.zrandom'
```

in your `.cshrc.mine` file which you would use to zwrite with random zsigs. The program follows:

```
#!/bin/csh -f
set zsigs = (\
"Your zsig number 1." \
"Your zsig can not easily deal with apostrophes and quotes." \
"Multi-line zsigs are not possible." \
"Exclamation points must be backlashed like this \!" \
)
```

```
set index = `jot -r 1 1 $#zsigs`
if ($index <1 || \$index>$#zsigs) then
        if ($index<0) then
        @ index = - $index
        endif
@ index = $index % $#zsigs
@ index = $index + 1
endif
zwrite -s "$zsigs[$index]" $argv:q
echo "$zsigs[$index]"
```

Another program you can use is a Perl randomizer was written by SIPB member Matthew Gray, <mkgray>. In order to use it, you need to create a file of your zsigs with one zsig on each line, which must be called ~/.zsigs, and then follow a few steps. You would put the perl code into a file, calling it something like .zrandom, make the file execut able using chmod 755 .zrandom, and then alias the execution of the file with a line like

```
alias zr '~/.zrandom'
```

in your .cshrc.mine file. The code follows:

```
#!/usr/athena/bin/perl

srand;
open(ZSIGS, "/mit/$ENV{'USER'}/.zsigs")|| die("No ~/.zsigs file");
$ops = join(' ', @ARGV);
     while(<ZSIGS>){
        chop;
        $sig[$i++]=$_;
    }
    $x = rand($i-1);
    print("Zsig: $sig[$x]\n");
    exec("zwrite", "-d", "-s", $sig[$x], split(' ',$ops));
```

Yet another randomizer was written by SIPB member Aaron Ucko <amu>. Like the previous randomizer, it uses a file called ~/.zsigs, but with signatures seperated by blank lines. This randomizer allows multiple-line signatures, but please bear in mind that sigs over a certain length are rude.

```
#!/usr/athena/bin/perl
# -*- perl -*-
die "no arguments" unless @ARGV;
$/ = '';
open(ZSIGS, "$ENV{HOME}/.zsigs");
@zsigs = <ZSIGS>;
$sig = $zsigs[rand(@zsigs)];
chomp $sig;
```

```
# $sig .= "\n" if (($sig =~ /.\n./) || (length($sig) >= 60));
exec('/usr/athena/bin/zwrite', '-s', $sig, @ARGV);
% There are a lot of funny comments in this document. That is because
% emacs font lock mode can't handle verbatim modes and other random
% programming languages thrown into the mix
% $
```

9.4 Other ways of using znol and zwrite

One of the simplest hacks you can do with `znol` is to have your znol list of who is logged on be sent to you as a zephyrgram instead of being in the xterm where you typed the command. This is done with UNIX redirection, sending the output of the znol program to zwrite as the message. To do this, you would type

`znol | zwrite` *yourusername*

Another popular program which many people use to replace both `znol` and `zwrite` is called `xzewd`. This program is **not** supported by SIPB. It was written by a user, Eddie Kohler, `<eddietwo>`. It is an X application that gives you lots of windows with silly graphics for your znol list and for sending zephyrgrams. In order to use xzewd, you need to type

```
add outland; xzewd &
```

If you have problems with xzewd, you should bring them to `eddietwo@mit.edu`. There are a few important things to note about xzewd. While it is easy to use, it makes dotfiles and changes some of your dotfiles for you. It also uses up computational resources constantly because of the graphics.

Other programs that some people use include `xzul` and `xzwrite`. `xzwrite` can be found in the standard athena release. To read more about it, type

```
man xzwrite
```

`xzul` is written by a SIPB member, Yonah Schmeidler. It is an update of `xznol`. For more information about it, type

```
add outland; man xzul
```

All of these programs are graphical applications that do similar things to `znol` and `zwrite`.

9.5 zcrypt

`zcrypt` is a program written by Philip Lisiecki. He is not affilited with SIPB and the program is not supported by SIPB. The program is designed to allow encryption over Zephyr. It has some security holes, but is an option if you feel the need for a more secure Zephyr conversation. For information about this program

```
add outland ; man zcrypt
```

10 Other Zephyr Clients

There are a few other commonly used zephyr clients out there.

10.1 Owl

Owl is a text-mode, curses-based zephyr client. Unlike the other zephyr clients used on Athena, owl is a monolithic program - sending and receiving of zephyrs is all done in a scrollable, searchable text

window. It has a configuration language in perl, though no configuration files are necessary to run it. You can find out more about owl from its official homepage at `http://www.ktools.org/owl.html`.

The following documentation was adapted from the owl introduction found at `http://web.mit.edu/ktools/src/owl/dev/doc/intro.txt`.

Running owl

Owl will run happily without a configuration file, so to get started just run the program. Owl will take over the terminal window it is started in, so you may wish to have another terminal window available at the same time.

On Athena, you can run the latest version of owl by typing at the athena prompt:

```
add ktools; owl-beta
```

Why owl?

Why use owl instead of another zephyr client?

1. Owl allows you to "thread" zephyr conversations and read them one thread at a time. This is convenient if you return from a lunch break to hundreds of zephyrs in owl.
2. Owl is configurable in perl
3. Owl is popular! More of your friends can help you learn to use it
4. Owl provides easy logging and filtering of messages
5. Owl can be run within screen and then accessed from any computer

The Screen Layout

There are three main parts to the owl screen. The large top portion of the screen is where messages are displayed. The status bar separates this area from the one below and displays owl status information, such as the total number of messages. The space below that is used to type messages and is also used by owl to give warnings and information to the user.

Getting help

Owl has a full on-line help system. Pressing the `h` key will bring up the basic help screen. Further help can be obtained using the help command, described later.

Sending zephyrs

To send a zephyr press the `z` key. This will start a `zwrite` command, which you can finish by typing the name of the user you wish to send to, followed by enter. Begin typing your message. When you are ready to send the message type `Control-D` or a dot (`.`) on a line by itself. If instead you wish to cancel the message type `Control-C`.

If you wish to send to a class/instance pair simply supply -c and -i arguments to the zwrite command as you normally would.

Getting around in owl

Owl indicates the **current** message with an arrow that looks like this: `->`. To move up and down from message to message, use the up and down arrows. To scroll the screen from side to side, use the right and left arrow keys.

Replying to a message is simple: simply select the message you want to reply to using the up or down commands, then type `r`. Type your message, then send it with Control-D or a dot (`.`) on a line by itself.

To delete a message, mark with with the `d` key, and a `D` will appear to the left of the message. To undelete a message, select it with the arrow keys and type `u`. Messages will not be removed from owl until you perform an expunge. `x` expunges all messages marked for deletion.

Here are some more message manipulation controls. Note that in Unix, "C-*" is a shorthand way of saying "Control-*" or "press and release the control key then this key." Similarly "M-*" is shorthand for "press and release Meta then this key". If you don't have a Meta key on your keyboard, the Esc will work the same way. Holding down Alt, then the specified key, then releasing both also replicates the Meta functionality.

Moving between messages

```
n   move to the next non-deleted message
p   move to the previous non-deleted message
C-n or down  move to the next message
C-p or up   move to the previous message
<  move to the first message
>  move to the last message
C-v          page down
M-v          page up
right        scroll the screen to the right
left         scroll the screen to the left
P            move to the next personal message
M-P          move to the previous personal message
```

You can also use the up and down arrow keys to move up and down from message to message.

Deleting, undeleting, and expunging messages

```
d   mark a message for deletion
u   unmark a message for deletion
x   expunge deleted messages
T   mark all 'trash' messages for deletion
M-D          mark all messages in the view for deletion
M-u          unmark all messages in the view for deletion
```

Replying to messages

Beyond your basic "reply" command, owl has some other reply functions.

```
r            Reply.  Personal messages get a personal reply,
                  group messages get a group reply.
R            Reply to sender.  Always replies personally
                  to the sender.
```

```
M-r          Reply but allow editing of the command line.
M-R          Reply to sender but allow editing of the
                  command line.
```

Manipulating windows within owl

Pressing `M` on a selected message brings it into a separate popup window. Within that window, you can further navigate the message with these commands.

```
SPACE        page down
b            page up
RETURN       line down
BACKSPACE    line up
```

(The message pointer will change to indicate that the message is not starting at the first line.)

To get more detailed information about a particular message (such as the host the message was sent from, or a more precise time stamp), type `i` with the message selected instead of `M`.

Command Mode

To enter owl's command mode press the colon (`:`) key:

```
:   begin command mode
```

Owl will give you a command prompt and you can begin typing your command. Type `Enter` to execute the command, `Control-C` to cancel. The basic commands are listed on the basic help screen (by pressing `h`).

For detailed information on the syntax and use of a command you can use:

`help` *command*

For example `help zwrite` will display all the options available when using the zwrite command.

AOL Instant Messenger (AIM) and Owl

In addition to sending and receiving zephyrs, owl also can send and receive AIM messages. Before sending an AIM message you must login to AOL Instant Messenger, by using the `aimlogin` command, with your screenname as an argument:

`aimlogin` *screenname*

You will be prompted for your password, which you then enter. Once you are successfully logged in you can send an AIM message by pressing the `a` key, which will bring up an `aimwrite` command:

`aimwrite` *screenname*

Supply the screen name you wish to write to as an argument and then send the message just as you would send a zephyr, as described above.

10.2 VT

VT is another text-mode zephyr client, originally designed as a MUD client and then later on (several years ago) was given as set of configuration files to support zephyr. The configuration is done in a C-like scripting language called VTC, and many people have diverging configuration files. For more information, you can go to http://web.mit.edu/cat/project/vt/ivt (information can be found on athena by looking in the directory /mit/cat/project/vt (make sure you type `attach cat` first).

10.3 WinZephyr

WinZephyr is a Windows based zephyr client. It has two display modes: Windowgrams which behave similarly to zwgc on Athena, and also a scrolling buffer. One can send zephyrs from this client by clicking on a zephyr sender icon, or by typing zwrite from a Command Prompt. More information is available from http://web.mit.edu/is/help/winzephyr

10.4 MacZephyr

MacZephyr is a MacOS classic based zephyr client. It has three display modes: As entries in a list (the default); As a TTY window; or as popup windows. More information is available from http://web.mit.edu/is/help/maczephyr

10.5 Gaim

Gaim is a GTK2 based multi-protocol IM client, supporting, in addition to Zephyr, AIM, MSN Messenger, Yahoo Messenger, ICQ, Jabber, IRC, and Gadu-Gadu. It supports both personal zephyrs and chats (subscription entries from your /.zephyr.subs), in per-person windows, or with a tabbed interface.

The version currently installed on Athena is 1.5.0 with a couple of fixes for connecting to MIT's jabber server. For more information, on gaim, you can go to http://gaim.sourceforge.net

To run it from any Athena cluster workstation, type

```
gaim
```

On some older Athena workstations, gaim is not installed locally, and you will have to run an older version from the im locker. To do that, type

```
add im; gaim
```

10.6 Emacs Zephyr

10.7 Emacs Zwgc

11 More Information

To start with, you can read manual pages on various things related to Zephyr. Some helpful ones include `man zwgc`, `man zwrite`, and `man zctl`. Also, looking in the `dotfiles` locker and at various people's world-readable `.zwgc.desc` files can give you more ideas about Zephyr customizations. You can also read through the On-Line Consultants' stock answers on Zephyr, by typing `olc_answers` and choosing the menu item on Zephyr. On-Line Help (OLH) also has information on Zephyr, which can be found by typing `help &` and choosing the menu item "Communicating with other Users."

You can also feel free to visit the SIPB office in W20-557, right next to the Athena Cluster in the Student Center, call us at 253-7788, or email us at `sipb@mit.edu` and ask us whatever you are interested in finding out. The Student Information Processing Board (SIPB) is a student volunteer organization that helps computer users at MIT and provides many services to the user community. You can stop by and ask us anything you'd like to know, about any computing-related topic, and we'll try our best to answer.