Squish Version 1.1
Reference Manual

Created May 6th, 1994

Documentation produced by Scott Dudley, with Don Dawson

TABLE OF CONTENTS

You can contact the author at any of the addresses listed below:

FidoNet:	1:249/106
Internet:	sjd@f106.n249.z1.fidonet.org
CompuServe:	>INTERNET:sjd@f106.n249.z1.fidonet.org
BBS:	(613) 634-3058 (V.32bis)

Surface mail:

777 Downing St.
Kingston, Ont.
Canada K7M 5N3

The author can also be reached through the FidoNet EchoMail conferences called MUFFIN (Maximus support) and TUB (Squish support).

Sending correspondence via electronic mail is strongly preferred, and the use of surface mail is discouraged. However, if you really have to send paper mail (and expect to receive a reply), please enclose a self-addressed, stamped envelope. (Users outside of Canada should include an international postal reply coupon instead of a stamp.)

DO NOT ATTEMPT TO CONTACT THE AUTHOR BY TELEPHONE! VOICE SUPPORT WILL NOT BE PROVIDED FOR NONCOMMERCIAL USERS!

Please feel free to contact the author at any time to share your comments about this software and/or licensing policies.

Our thanks to the Free Software Foundation for most of the wording of this licence.

INTRODUCTION

About Squish

Squish is a multi-featured, FidoNet-compatible EchoMail processor. Squish incorporates most of the common EchoMail functions into one integrated package, including tossing, scanning, packing, point remapping and topic linking.

Although Squish was designed to be used with Maximus 2.0 or above, Squish is compatible with other software which supports either the Squish or the *.MSG message base standards. Squish is not merely a "giveaway" utility; rather, Squish is a full-featured conference manager, making it highly competitive with most stand-alone packages on the market today.

To install Squish, your first task is to PRINT OUT AND READ THIS DOCUMENT. At the very least, you should print the section on installation. The "Quick Installation" section can guide you through the standard installation procedure.

Features

Some of the features in Squish version 1.1 include:

* One-pass tossing and scanning, with full support for "passthru" areas. Outbound messages can be built directly from the inbound *.PKT files, without needing to stop over in the message areas.

* A 32-bit "industrial-strength" version of Squish is available (under both DOS and OS/2) for systems which carry many areas and which have large numbers of downlinks.

* Support for "feature DLLs" under OS/2. Third-party developers can write dynamic link libraries which perform actions while Squish is tossing or packing mail.

* Support for MSGID-based message linking and dupe checking.

* Internal support for both BinkleyTerm and FrontDoor-style routing.

* Squish supports both the standard *.MSG format and the proprietary, flat-file *.SQ? format on an area-by-area basis.

* Superior multi-zone operation. Primary addresses can be selected on an area-by-area basis, as can SEEN-Bys and

numerous other features. It's now easily possible (and practical!) to use a single configuration file for multiple, unrelated FidoNet-technology networks.

* True support for BinkleyTerm and InterMail "busy flags". Instead of remaining blocked while the mailer is transmitting to another node (and therefore holding up processing), packets are simply queued for later use.

  All processing is performed in a separate working directory, and packets are only transferred to the mailer's outbound area as necessary. Since Squish stays out of the way of other mail-handling tasks, Squish offers a significant performance advantage over other mail processors.

* Areas can be defined in AREAS.BBS for compatibility with other programs, but areas can also be defined in the main Squish configuration file.

* Support for all archiving and dearchiving programs, past and future, through the use of a flexible archiver control file.

* Verbose binary statistical information (optional). Squish provides enough information for external utilities to provide a 100% accurate billing report for NECs or hubs, based on mail volume.

* Point support, running as either a bossnode or a point, for both 4D and "fakenet" points.

* Support for the "2+" 4D packet header proposal, including zone and point numbers.

* "Point directory" support for BinkleyTerm 2.50 and above. Squish is the first publicly-available program to support the Binkley point directories both conveniently and efficiently.

* Squish also features a built-in node remapper and message linker for BinkleyTerm-style systems. The remapper is not limited to points; it also supports wildcards and soundex name matching. The message linker supports both *.MSG and *.SQ? areas, so no external reply linker is required.

* Squish can optionally swap to XMS, EMS or disk when running external programs. Squish can therefore be used in many tight-memory situations, since Squish will only occupy 3K of memory when swapped out. (DOS only.)

*   Squish runs under OS/2 in protected mode, in addition to running under DOS in real mode. The OS/2 version of Squish includes a special serialization feature to allow Squish to be run conveniently in a multi-line environment.

System Requirements

Although Squish was designed to be as generic as possible, the following system configuration is required as a minimum:

* An IBM PC, XT, AT or PS/2, or a 100% compatible. To run the 32-bit version of Squish, an 80386SX, 80386DX, 80486SX, 80486DX, Pentium, or compatible processor is required.

* A hard drive, with at least two megabytes of free space for the installation, plus space to hold the inbound packet files, local message areas, and generated packets.

* Software supporting either the *.MSG or *.SQ? message formats.

* A front-end which is compatible with either BinkleyTerm or FrontDoor.

In addition, Squish-DOS requires MS/PC-DOS 3.0 or above, and Squish-OS/2 requires OS/2 (IBM or MS) 1.2 or above. If you wish to handle compressed mail, Squish will also require an external archiving program.

Squish, Money and You

In general, Squish is a freeware program. If you are a noncommercial user and you abide by the terms given in the licence, there is NO CHARGE for running Squish. NONCOMMERCIAL USERS ARE STILL WELCOME TO RUN SQUISH WITHOUT PAYING A CENT.

Unlike other software, this program has no crippled features, extra bells'n'whistles or "registration incentives". There is one simple difference between the commercial and noncommercial versions of Squish: the commercial version entitles you to legally use Squish in a commercial environment.

If you are a COMMERCIAL USER as defined in point 8) of the licence above, you must obtain a licence before using Squish. Please see ORDER.FRM for more information. If you did not receive a copy of the order form with this document, contact the author at one of the addresses listed in the licence for more information.

If you are a NONCOMMERCIAL user as defined in the licence above, you are welcome to use Squish for free. However, Squish is an extremely large and complex program, and simply maintaining the existing code is consuming a large percentage of my time. Donations of any value (or even just a postcard) will be gladly accepted. However, noncommercial donations are on a VOLUNTARY basis and are NOT REQUIRED.

However, BOTH commercial and noncommercial users must abide by several restrictions before using Squish. The main points in the licence are:

* You may not distribute Squish on a CD-ROM, WORM, or as any other form of a "value added" good in a sales transaction. You may not bundle Squish with other software without prior written permission of the author.

* You may not incorporate parts of Squish into another program which is not completely free for all users.

Other than the above, there are few restrictions on the use of Squish. Please read the licence agreement carefully, since additional restrictions or qualifications may apply to you.

Ordering Information

Please see the ORDER.FRM file (included in the Squish distribution package) for information on ordering a copy of Squish for commercial use.

NETWORK PRIMER

This section is intended as a primer for SysOps who are new to FidoNet or a FidoNet Technology Network (FTN). This section covers many of the terms and concepts which are required for everyday FidoNet operations. Those who are familiar with EchoMail and mail routing should feel free to skip on to the next section, entitled "Installation".

The Basics

The term "FidoNet" refers to an amateur electronic mail network, run collectively by a group of system operators. In the beginning, FidoNet started out as a simple system for exchanging private messages between different bulletin boards. Since then, FidoNet has grown into a full-fledged electronic mail and conferencing network which has members in most countries of the world.

FidoNet itself is organized into a numerical hierarchy of "zones", "regions", "nets", "nodes" and "points". Each member of FidoNet, individually known as a "node", can be uniquely identified by that system's zone, net, node and point numbers. To define each term:

Zones are wide geographical areas, usually covering one or more continents. At the time of writing, FidoNet currently has six zones: zone 1 (North America), zone 2 (Europe), zone 3 (Oceania), zone 4 (South America), zone 5 (Africa) and zone 6 (Asia).

Nets cover a much smaller area than zones; a net usually encompasses a large city and the surrounding area. There are usually many nets within each zone, each of which represents a small geographical area within that zone.

Nodes are individual systems. Most nodes consist of bulletin board systems, although a few nodes are devoted exclusively to handling mail. If you wish to become a member of FidoNet and you are running a BBS, this is probably where you will start.

Points are users on an individual system. Normally, points do not run full-time systems, since they simply send and receive mail through their "bossnodes" (the nodes where the points pick up their mail). As the size of the network grows, points are becoming increasingly popular. If you don't wish to run a full-time system, this is probably where you'll start.

These four terms can be combined to give a "network address" which identifies any one node in the network. The format of a FidoNet address is as follows:

```
zone:net/node[.point]
```

For example, given a user in zone 1, in net 249, with a node number of 106, and a point number of 2, that user's full address would be '1:249/106.2'. The point number is optional, so both 1:249/106 and 1:249/106.0 refer to the bossnode of 1:249/106.2.

This mode of addressing is sometimes referred to as "4D" or four-dimensional, since it includes the four basic elements of a network address.

The Outside World

Like other electronic mail systems, it's possible to enter a private message on a FidoNet system and have that message be delivered to its final destination in a short period of time. FidoNet systems "talk" with each other over telephone lines, using one or more sophisticated handshaking protocols. To get a message (known in this context as "NetMail") from system "A" to system "B", the following sequence of events has to occur:

* The message is created. Most Fido-compatible software packages can be used to generate a private message to a user on another node. The destination address is entered, using the standard 4D addressing scheme.

* The on-disk message is then converted to packet (or *.PKT) form. If you are running BinkleyTerm, this will be performed by Squish after a user logs off. If you are running FrontDoor or a similar mailer type, this will be performed by the mailer itself on startup, or while your mailer is connected to other systems.

  There are four reasons for converting a message into a packet:

  1) The packet structure is much more flexible than the local message structure. All of the fields (such as the To:, From: and Subject: fields) in a packet are variable length, whereas the fields in stored messages are fixed-length.

  2) Packets are the "compatibility layer". Since all systems convert messages to the *.PKT format before sending them to another system, there are few

compatibility problems. This means that systems can store their local message bases in different formats, but still be able to exchange messages easily. In addition, more than one message can be stored in a single packet. Sometimes hundreds (or even thousands) of messages can be stored in a single packet.

3) Messages in a packet can have a different address from the packet itself. The packet itself has a destination (the system where you'll be sending that packet directly to), but each message has an individual destination address. This is useful, for example, when a long-distance call is required to connect with certain parts of the network. The message's final destination always stays the same, but by sending the packet to someone who is local to you (and then having that someone send it to another local system, and so on), costs can be controlled quite effectively. Since the interim destination of a packet does not need to be the same as the final destination of the message, routing of messages via the lowest-cost route can be performed.

4) Packets can be given a "flavour". A "flavour" (or a behaviour characteristic) helps your system decide what to do with an individual message. For example, the "hold" flavour instructs your system to hold the message and wait for the destination system to call and pick it up. Other flavours include "crash" (send a message directly to the destination), "direct" (same as crash), and "normal" (wait for later routing commands).

Packets always have an extension of *.PKT. (Qualifier: if you are running a BinkleyTerm system, they may have an extension of *.HUT, *.OUT, *.CUT, or *.DUT on your local system, but Binkley always changes them to *.PKT files before they are sent to another system.)

* After the packet is created, it can be optionally archived using a file compression utility. Compression is useful when transferring large volumes of mail or sending to long-distance sites, since compressing mail saves both time and money.

* The system which created the message then tries to call the destination system. Obviously, if both systems are fairly busy, this may take a while. Messages are sent back and forth between systems through the use of mailers, also known as "front ends". Mailers call out to deliver waiting mail,

handle incoming messages and files, and in general, supervise the entire file transfer.

*   After the two mailers connect (using one of several FidoNet protocols), waiting mail and files are transferred between the two systems.

*   After the transfer completes, the receiving system then tries to import the message with Squish (or any other mail processor). If the packet was compressed by the sender, it will be decompressed. The *.PKT files will then be imported (otherwise known as "tossed") into the local message base, ready for the recipient to read.

Although transferring NetMail can involve much more than just what is given above, this should give you a grasp on NetMail fundamentals.

Is There an Echo In Here?

In the beginning, FidoNet consisted solely of nodes exchanging NetMail. The only way to get a message from "here" to "there" was to send a private NetMail message. However, a technology called "EchoMail" was developed in late 1985; EchoMail is analogous to a public message area or conference, but EchoMail areas (sometimes known as "echoes") are shared among several other systems.

EchoMail is organized into different groups of echoes, each with a different topic. For example, the topics of FidoNet echoes range from Maximus Support to deep-sea fishing and many more special-interest groups. To facilitate topic-oriented EchoMail, each echo must given a tag (or area name). This tag is used to uniquely identify that EchoMail area when transferring messages with other systems. (It doesn't matter what you call the echo on YOUR system, as long as you are using the same tag as everyone else.) Area tags are one word only, although they can include periods and underscores. To start receiving an echo area, you need to know the tag of that area. For example, the area tag for the echo dealing with hardware and other technical issues is "TECH".

EchoMail messages are normally public, and they are entered in a message area just like a normal message. EchoMail messages also look like normal, locally-entered messages, but with some special control information at the bottom of each message.

After an EchoMail message is saved, an EchoMail utility (such as Squish) is invoked to "scan" that message out to the rest of the network. Unlike NetMail, EchoMail areas have an electronic topology. Some echoes are very large, and as such, the cost to directly send a message to each system which carried that echo would be prohibitive. Instead, each system carrying that echo only transfers EchoMail messages to neighbouring systems. (The neighbour you receive an echo from is also known as your "feed".) EchoMail messages get sent from the originating system to its neighbours, and from those systems to their neighbours, and so on. Despite this "hoppity-hop" method of transferring messages, EchoMail is fairly quick; it can often take less than three days for a message to travel from the USA to Australia and back.

Just like NetMail, echoes are sent to other systems in packets. EchoMail messages are almost always compressed, since most of the popular echoes have a daily throughput anywhere from 20 to 200 messages per day.

Squish handles EchoMail automatically, just like NetMail. However, you have to tell Squish the names of the areas that you wish to carry, in addition to who your "neighbours" are for each echo. (Information on doing this is covered in greater detail in the installation section.)

There is much more to both NetMail and EchoMail than mentioned above; however, you should now be comfortable enough with FidoNet terminology to start installing Squish.

INSTALLATION

Assumptions

Before proceeding any further, you should already have a FidoNet-compatible front end installed, in addition to a software package which reads either *.MSG or *.SQ? format message areas. In addition, you should have a set of working batch files for your system.

This installation guide only covers the bare essentials, and it glosses over what is required to run Squish as an EchoMail "leaf node" (a system which only transfers mail with one other system).

Before reading this section, you should be at least somewhat familiar with network terminology and operations; if not, read over the prior section entitled "NETWORK PRIMER". A minimal knowledge of batch files is helpful.

The quick installation also assumes that you will be using the *.MSG format for storing messages. If you wish to use the Squish format, please see the section entitled "USING SQUISH-FORMAT MESSAGE AREAS".

This installation procedure doesn't deal with any advanced topics, so most of the command examples have been simplified to make the installation process easier. For full descriptions of each command, please consult the reference material contained later in this manual.

16-bit or 32-bit? SQ386 and SQ386P

The main Squish executable currently comes in four flavours. You may want to use a different version of Squish depending on your operating system and computer type.

The examples in this documentation assume that "SQUISH.EXE" is the name of the Squish executable being used.

The DOS distribution includes both SQUISH.EXE and SQ386.EXE:

> SQUISH is the 16-bit version of Squish that runs on all computers. SQUISH requires at least an 8088 processor.
>
> SQUISH can address up to 640 kilobytes of memory.
>
> SQ386 is the 32-bit, DOS-extended version of Squish. SQ386 requires at least an 80386 processor. (SQ386 is optimized to run the fastest on an 80486, but it will still work on an 80386.)
>
> SQ386 can address up to 4 gigabytes of memory. SQ386 usually runs faster than SQUISH.

The OS/2 distribution includes both SQUISHP.EXE and SQ386P.EXE:

> SQUISHP is the 16-bit version of Squish that runs under all versions of OS/2. SQUISHP requires at least an 80286 processor.
>
> SQUISHP can address up to 16 megabytes of memory.
>
> SQ386P is the 32-bit version of Squish that requires OS/2 2.0 or above. SQ386P requires at least an 80386 processor. (SQ386P is optimized to run the fastest on an 80486, but it will still work on an 80386.)
>
> SQ386P can address up to 512 megabytes of memory. SQ386P usually runs faster than SQUISHP.

All versions of Squish support the same basic feature set, but the 32-bit versions have some extra options (such as larger buffer settings) that allow for faster execution.

SQUISH, SQUISHP and SQ386P are native applications that run without extra support from the operating system.

SQ386 is a DOS-extended application that requires a DOS extender to operate. The DOS distribution of Squish includes a copy of the DOS/4GW DOS extender. While this DOS extender should work with other operating environments (such as Windows, DESQview, and the OS/2 DOS box), not all combinations have been thoroughly tested.

Running SQ386 may not be a simple "plug-and-play" exercise. DOS extenders are kludges to get around the limitations of DOS, but this means that they may not be compatible with all environments.

For example, DOS-extended applications may not work properly on a non-dedicated DOS fileserver. DOS-extended applications may also steal system interrupts for longer than the usual amount of time, possibly causing errors if another task is performing an upload or download while SQ386 is running.

These are limitations of the DOS extender, and from the point of view of the application developer, very little that can be done about it. DOS extenders are rickety by nature, and they may not be able to coexist with certain hardware or software configurations.

WHILE SQ386 WAS TESTED AS THOROUGHLY AS POSSIBLE, THERE IS NO GUARANTEE THAT IT WILL WORK PROPERLY IN ALL ENVIRONMENTS. IF SQ386 DOES NOT WORK FOR YOU, GO BACK TO USING THE STANDARD SQUISH EXECUTABLE.

While we welcome problem reports regarding SQ386, remember that it may not be possible to make SQ386 work for you.

For more information on using SQ386, please see the section entitled "SQ386 Usage Notes".

Quick Installation

After reading the installation section at least once, you should decompress all of the Squish files from the distribution archive into a separate directory. You can place Squish anywhere you desire, although all of the examples in this manual use C:\Squish as the base directory.

After everything has been decompressed, you will need to load an ASCII text editor to modify the Squish configuration files. Any plain text editor will do, including Qedit, DOS 5.0's EDIT, any word processor in "non-document" mode, or even EDLIN.

The four main Squish configuration files are:

SQUISH.CFG

> This is the main configuration file. Information about your system is kept in here, including your system addresses, passwords, run-time options, and (optionally) EchoMail area definitions. Squish will first look for a "SQUISH" environment variable. If found, it will try to use the named file as the Squish configuration file. Otherwise, Squish will look for a file called SQUISH.CFG in the current directory.

ROUTE.CFG

> This is the control file used for mail routing and schedules. This file is used for both FrontDoor and BinkleyTerm-style systems, although it plays only a minimal role when used with FD.

COMPRESS.CFG

> This holds information about all of the archiving programs on your system. Squish uses this information to automatically identify the type of incoming archives, and it uses the commands within to add to and extract from archives. This file is compatible with the compression configuration file used by Maximus 2.0 or above.

AREAS.BBS (optional)

> AREAS.BBS has historically been used to define EchoMail conference information, including the name of the conference, where to store the local message base, where to send the messages to, and so on. Squish fully supports the AREAS.BBS format; however, Squish also supports message area

definition in SQUISH.CFG. For flexibility, areas can be declared in both AREAS.BBS and SQUISH.CFG, which provides for complete compatibility with all existing software.

Modifying CONFIG.SYS

This section applies only to DOS users. OS/2 users can skip this step and go on to the next section, "Customizing SQUISH.CFG".

Before modifying the Squish configuration files, you must first make sure that your system has been configured properly. Squish is a disk-intensive program, and it keeps a number of files open at the same time. To make sure that your system is set up to allow this, use an ASCII text editor to edit C:\CONFIG.SYS.

Inside CONFIG.SYS, there should be a line which reads 'FILES=n', where 'n' is a number from 8 through 255. If this line doesn't exist, it should be added. For Squish, you must make sure that 'n' is no less than 30. If you are running a multitasking system, even more file handles (50 or 60) may be necessary. Having more than 30 file handles is allowable, but having less than 30 will certainly cause problems.

After you have changed your FILES statement, save CONFIG.SYS. Since CONFIG.SYS is only read once when the computer starts up, you must reboot to make sure that your changes are recognized by the operating system.

WARNING! IF YOU INTEND TO USE SQUISH-FORMAT MESSAGE AREAS IN A MULTITASKING OR NETWORK ENVIRONMENT, YOU *MUST* INSTALL SHARE.EXE! Squish uses SHARE for file and record locking, and if two programs are accessing the flat-file Squish format without SHARE loaded, message base corruption will occur.

To install SHARE.EXE, either add this line:

```
     INSTALL=C:\DOS\SHARE.EXE
```

to your CONFIG.SYS, or add the following line:

```
     SHARE
```

to the end of your AUTOEXEC.BAT. Both methods have the same effect. (Note: DOS 5.0 users must install SHARE by the CONFIG.SYS method.)

Customizing SQUISH.CFG

After modifying CONFIG.SYS and rebooting your machine, you can now start configuring Squish itself. For starters, SQUISH.CFG needs to be modified to suit your system. Although you will see many options in the configuration file, only a few need to be modified to get a minimal Squish system up and running. When performing a new installation, most of the options in SQUISH.CFG should be left alone, since the defaults are acceptable in most cases. However, you will need to modify the following keywords:

Address

This keyword must be modified to match your system's actual network address. If you are already a member of FidoNet or some other network, include your full 4D address here, including your zone, net and node number. If you do not have a node number, set your address to "1:-1/-1" until you receive an official address.

If you are running a point system, please see the SQUISH.CFG reference for more information on configuring Squish for use in a point environment.

NetFile

This keyword tells Squish where to find inbound files. This is commonly referred to as a "NetFile path" or an "inbound directory", since this is where your front end places inbound *.PKT files and compressed archives.

AreasBBS

This keyword points Squish to the location of an AREAS.BBS file. This file is optional, so if you do NOT have any software which uses an AREAS.BBS, this statement can be commented out. If you are new to FidoNet, please see the section entitled "Configuring EchoMail Areas" to decide which format is best for you.

ArcmailAttach

If you are running FrontDoor, InterMail, D'Bridge, or any other front-end which requires a "NetMail attach message" to send files, then this keyword should be enabled.

Otherwise, if you are running BinkleyTerm or any other program that uses the "outbound area" concept, this statement should be commented out (disabled).

Compress

The "Compress" keyword specifies the location of the archiver configuration file. If you wish to put COMPRESS.CFG somewhere else (or if you wish to use a compatible COMPRESS.CFG, as used by Maximus), you can specify an alternate path and filename here. Otherwise, this option should be left alone.

Routing

The "Routing" keyword gives the location of your routing control file. If you are using the default configuration, you can leave this as is.

Outbound

This keyword specifies a directory to use for building packets and file attaches. This keyword is required for both FrontDoor and Binkley-style systems.

In a Binkley environment, this should be the "root name" of the BinkleyTerm outbound area.

In a FrontDoor environment, this will be the base directory used for building packets and compressed mail archives.

No directory extensions should be given for this keyword. For multi-zone systems, Binkley adds an extension to the base directory (to separate the mail for each zone). However, Squish will add the OUTBOUND.### zone extensions AUTOMATICALLY, you should just specify the path (without an extension). Squish will also create a private work area (with a .SQ extension) for both Binkley and FrontDoor-style systems.

LogFile

The LogFile keyword specifies the path and filename of the Squish log file. This log file uses a Binkley and Maximus-compatible log format.

Origin

The Origin keyword is only required if you are not using AREAS.BBS. (AREAS.BBS includes a default origin line, so you can skip this keyword if you are using AREAS.BBS.) Squish needs a default origin line to place at the end of messages, just in case a message being scanned didn't already contain one. Generally, this line should contain the name of your system, your location, and your phone number. The text in your origin line text should be no more than 60 characters (letters/numbers) long.

For a normal system, the above changes should be enough to get your system up and running. Later, once your system is running reliably, the control files can be modified to suit your personal preferences.

Configuring NETMAIL and BAD_MSGS

After customizing the main configuration file, you need to declare at least three messages areas. For starters, a NetMail area is required. This is where private messages addressed to specific users on other systems are stored. Squish requires at least one netmail area. Secondly, a bad messages area is also required, since Squish needs a place to store invalid messages.

Netmail areas are declared by placing a 'NetArea' line in SQUISH.CFG. A sample NetArea declaration might look like this:

```
     NetArea   NETMAIL   C:\Max\Msg\Net
```

'NetArea' is the area type. This tells Squish that the area contains netmail.

The next part of the line is a one-word 'area tag', which is simply a short form for the area name. In the case above, 'NETMAIL' is the area tag. All areas must be given a unique area tag, but aside from that, the area tag you give to a NetMail area (whether it be 'NETMAIL' or 'WOMBAT') is of little importance.

Finally, the last item in the line is the path to the area itself. For *.MSG format areas, this should be a separate directory on your hard drive. Squish will create nonexistent directories when importing messages, so you don't have to create the directory right away.

*   Tip for advanced users: A Squish-format netmail area can be used instead of *.MSG. For more information, see the documentation for 'NetArea' in the SQUISH.CFG reference.

Next, an area to hold bad messages must be created. This area will be used to store insecure messages, messages destined for unknown areas, and other messages that Squish can't process. The format for a 'BadArea' line is very similar to that of netmail areas:

```
     BadArea   BAD_MSGS  C:\Max\Msg\Bad
```

'BadArea' is the area type, and it tells Squish that the area will be used to hold bad messages. WARNING! If you are using MsgTrack or some other message-bouncing utility, ensure that each program has its own "bad messages" area. Otherwise, Squish will erroneously attempt to toss bounced messages.

The next part of the line is the area tag, which has the same restrictions as the tag for netmail areas. 'BAD_MSGS" is suggested for the area tag, but any other single word will do.

As with netmail areas, the last part of the line contains the path to your bad messages directory. Again, the bad messages area defaults to the *.MSG format, so this should be the name of a separate directory on your hard drive.

Finally, an area for storing duplicate messages is also required. This area will be used to store messages that were erroneously sent to your system more than once. The format for a 'DupeArea' line is similar to that of netmail and bad message areas:

```
     DupeArea  DUPES     C:\Max\Msg\Dupes
```

'DUPES' is the suggested area tag for the dupes area.

Configuring EchoMail areas

After creating the definitions for both NETMAIL and BAD_MSGS, the next task is to define one or more EchoMail areas. Squish supports two different methods of declaring echoes; the best method for you depends on your system configuration.

First of all, EchoMail areas can be declared in SQUISH.CFG. The format for defining echoes is almost identical to the formats for NETMAIL and BAD_MSGS, so it's easy to remember. In addition, several Squish-specific flags can only be used in SQUISH.CFG.

Secondly, areas can also be declared in AREAS.BBS. This file is the "standard" for EchoMail area definitions, and many other programs support AREAS.BBS. The format of AREAS.BBS is less flexible than the format of SQUISH.CFG, so several Squish-specific options are not available when using AREAS.BBS. However, if you are converting from another program and already have an AREAS.BBS, this is probably the wisest option.

* Tip for advanced users: Squish can handle areas defined in both SQUISH.CFG and AREAS.BBS. You can even declare one area in both places, if you need compatibility with other programs and also Squish-specific features. For more details, please see the EchoArea portion of the SQUISH.CFG reference.

Declaring Areas in SQUISH.CFG

If you have decided to declare your EchoMail areas in SQUISH.CFG, then read on. Otherwise, skip ahead to "Elementary Routing". Remember, if you are not using AREAS.BBS to define your echoes, you must enable the 'Origin' statement in SQUISH.CFG. (See the "Customizing SQUISH.CFG" section for more details.)

Defining an EchoMail area in SQUISH.CFG is similar to defining netmail and bad message areas. A sample echo definition might look like this:

```
     EchoArea  MUFFIN    E:\Msg\Muffin  1:123/456
```

'EchoArea' tells Squish that the area we are defining is an EchoMail area.

'MUFFIN' is the tag for this area. Unlike the tags defined for NetMail and bad message areas, the tag that you specify for EchoMail areas is important, since the tag is used as the area name when sending EchoMail to other systems. Area tags are usually short, and spaces are not allowed. You must ensure that your system is using the same echo tag as your feed, so it's best to call your feed and get the required tag information in advance.

'E:\Msg\Muffin' is the path to the EchoMail area. By default, echoes use the *.MSG format, so a separate directory is required for each echo. If this directory does not exist, Squish will create it automatically.

Finally, '1:123/456' is the address of the system from which you receive the echo. This tells Squish that it's okay to accept mail from that address, and it also tells Squish to send locally-entered messages to that system. As many addresses can be listed as desired, with a space between each address. If you are a leaf node, you'll only need to list the address of your feed. Full 4-dimensional addresses (zone, net, node and point) are acceptable.

In addition, a number of special flags can follow the addresses. These flags can be used for many purposes, including declaring the area to be Squish format, defining a primary address, or declaring an area as 'passthru'. For more information, please see the EchoArea portion of the SQUISH.CFG reference.

Any number of EchoMail areas can be defined, limited by available memory. However, each area must be defined on a separate line in the configuration file.

Declaring Areas in AREAS.BBS

If you have already defined your EchoMail areas in SQUISH.CFG, then skip ahead to "Elementary Routing". Otherwise, read on.

AREAS.BBS is the de facto standard for defining EchoMail areas, so you'll have the highest degree of compatibility if you declare your areas using this method. AREAS.BBS contains two separate components: a default origin line, and also a number of echo definitions.

Default Origin Line

The very first line of AREAS.BBS is interpreted in a special manner. The line should have the following format:

```
     <default_origin>! <sysop_name>
```

<default_origin> is the default origin line to use for all of your echoes. Normally, this should include your system name, location, and phone number, or anything else that will fit in under 60 characters.

<sysop_name> should be your name. For historic purposes, this should be separated from the default origin line by an exclamation point. Squish doesn't use the name you specify here, but it should be included for compatibility with other programs.

For example, the first line in AREAS.BBS might look like this:

```
MyBBS * Anytown, Anystate * (123) 456-7890! Joe SysOp
```

In this example, the default origin line would be "MyBBS * Anytown, Anystate * (123) 456-7890", and the SysOp name would be "Joe SysOp".

EchoMail Areas

Following the default origin line can be any number of EchoMail areas. All echoes must each be defined on a separate line, using the following format:

```
     <path>  <tag>  <nodes>
```

<path> is the name of the directory in which the *.MSG files are to be kept. As with other *.MSG-type areas, each echo should have its own separate directory. If the directory does not exist, it will be automatically created.

<tag> specifies the area tag for this area.

<nodes> is a list of zero or more nodes to which will be sending the specified echo to you. Normally, for a leaf node, you will only be sending the echo to one place: to your feed. All of the addresses go after the area tag, with at least one space between each address.

For example, the following line could be used as an entry for the MUFFIN echo:

```
C:\MAX\MSG\MUFFIN   MUFFIN    1:123/999 888 777
```

Elementary Routing

After you have configured the EchoMail areas available on your system, your attention should be turned to mail routing. In Squish, routing is based on the idea of schedules and control files. A schedule is simply a set of routing commands which can be performed as a single unit. Schedules can be run either all day, during certain times of the day only, or on manual request. Most nodes will only need one schedule, since the majority of systems use the same set of routing commands 24 hours a day.

Commands in ROUTE.CFG have three purposes:

* Firstly, routing can direct NetMail from one system to another. (Normally, EchoMail is not routed.) For example, routing commands could redirect all NetMail destined for 1:123/456 to 1:987/654. If 1:987/654 is a local call, but 1:123/456 is not, then it's obviously useful to send mail via the system which is local to you. For ArcmailAttach systems, this feature is usually handled by your mailer.

* Secondly, routing can control whether or not mail is compressed. When sending large volumes of EchoMail, external programs such as ARC and ZIP can be used to compress mail packets. This reduces the amount of time it takes to transmit mail, which in turn reduces long-distance phone charges.

* Finally, routing also controls the 'flavour' of outbound mail. All outbound mail has a flavour (or priority) which controls when the mail gets sent. By default, Squish creates all outbound packets with a "normal" flavour. However, the routing commands can be used to change this flavour to "crash" (send this mail immediately), "direct" (a synonym for crash), or "hold" (do not call out; hold mail for pick-up). The flavour of messages can also be changed by your mailer (depending on when phone rates are the cheapest, for example), but ROUTE.CFG is used to assign a default flavour to each outbound packet.

In general, ROUTE.CFG starts off with a section of global routing commands (which are run every time Squish scans the netmail area), followed by a set of zero or more schedules. The commands within the global section and each schedule all use the same format; the only difference is when the commands are executed.

No matter what, commands in the global section of the routing control file are ALWAYS executed. Even when explicitly running a different schedule, the global commands are still run first.

Therefore, the global section should contain commands that you want to run every time that Squish is executed. Everything between the first line of ROUTE.CFG and the first 'Sched' statement is considered to be a global command and is treated accordingly.

Since most nodes will not require schedules, only basic routing information is described here. Most nodes will have all of their routing commands in the global section of the routing file, which is what this quick installation describes.

Routing commands in ROUTE.CFG are executed from top to bottom in sequence. In other words, if you place a certain routing command before another, you can be assured that Squish will process each command in the order you specified.

By default, unless routing commands are used for a given node, mail will be sent directly to that node, uncompressed, using the normal message flavour. However, various routing commands can be used to modify this behaviour.

The Send Command

The most basic form of routing is the 'Send' command. This command instructs Squish to compress mail for the specified nodes, and to give the resulting mail a flavour. The Send command does NOT perform any readdressing; it simply modifies the flavour and compression of the mail, without changing that mail's destination.

The format for the Send command is as follows:

```
Send <flavour> <node> [<nodes>...]
```

<flavour> specifies the flavour to use for the resulting compressed mail archive. Valid flavours are normal, crash, hold and direct.

Following the flavour comes a list of one or more nodes. Squish will search for normal-flavoured, uncompressed mail destined to any of the specified nodes, and compress and flavour that mail accordingly. For example, given the following command:

```
     Send Crash 1:123/456
```

Squish would take all normal-flavoured packets for 123/456, compress them using the default archiver, and send the compressed mail archive to 123/456 using the crash flavour.

You can specify more than one node for a Send command, but Squish behaves exactly as if each node were in a separate Send command of its own. If you wish to route mail for one system through another, then the Route command must be used instead.

In other words, the following command:

```
     Send Crash 123/456 234/567 345/678
```

is completely identical to this:

```
     Send Crash 123/456
     Send Crash 234/567
     Send Crash 345/678
```

Both of these commands would compress normal-flavoured packets for 123/456, 234/567 and 345/678, give the compressed mail the crash flavour, and send the resulting archives to 123/456, 234/567 and 345/678 (respectively).

If you are using an ArcmailAttach mailer, then the 'Send' command is probably the only one you'll need. Dynamic routing is performed by your mailer, so the only reason for using ROUTE.CFG is to compress and flavour mail, which is exactly what the Send keyword does.

The Route Command

The Route command is similar to the Send command; however, Route can be used to change the destination address of mail (otherwise known as routing that mail), whereas Send only sends mail directly to its destination. The Route command is normally NOT required for systems using the ArcmailAttach keyword. If you are running an ArcmailAttach mailer, most messages will be routed on-the-fly, so you can safely skip this section.

The format of the Route command is as follows:

```
     Route <flavour> <target> [<nodes>...]
```

As with the Send command, <flavour> specifies the message flavour to give the resulting archive. Valid flavours are normal, crash, direct, and hold.

<target> specifies the address of the routing target. Mail for all of the other nodes will be packaged up, given the specified flavour, and sent to this address. In other words, this is where all of the routed mail will be sent. In addition, mail addressed to the target itself will also be flavoured and sent accordingly.

Make sure that you have the permission of the target node before routing mail through his/her system.

<nodes> is the optional list of network addresses for which Squish should readdress mail. Squish will look for normal-flavoured, uncompressed packets for these nodes, and then route them through the specified target.

For example, the following route command:

```
     Route Crash 123/456 234/567 345/678
```

would take mail for 123/456, 234/567 and 345/678, compress it using the default archiver, and send it all to 123/456 using the crash flavour. Note the difference between Route and Send: whereas Send will send the mail to each node individually, Route will send all of the mail to the first node specified. The difference between Route and Send is a fundamental concept in Squish routing.

Wildcards

The Route and the Send commands provide the framework for a very flexible routing system. However, the Route and Send commands are often not enough to accomplish a particular task. For example, to route mail for all nodes in net 123 through another node, is it necessary to explicitly give the node number for each system? Fortunately, the answer is no.

The All and World Wildcards

In both SQUISH.CFG and ROUTE.CFG, Squish supports a form of wildcards. These wildcards can be used to specify a particular node or range of nodes for a particular routing command. The most basic form of wildcard is 'All'. 'All' can be used in place of a zone, net, node or point number, and it instructs Squish to process mail for all nodes which match the rest of the address.

For example, given the following command:

```
     Send Crash 106/All
```

Squish would then scan for normal packets destined to any node in net 106, compress the packet, give it the crash flavour, and send the resulting archive directly to its destination.

Wildcards can also be used with the 'Route' command. To use the same example, but to have all of net 106's mail routed through one node, the following could be used:

```
     Route Crash 106/123 106/All
```

As above, this would compress normal-flavoured packets for nodes in net 106, and give the resulting archives the crash flavour. However, all of the archives would be sent to 106/123, where they could be handled by routing commands on 106/123's system.

Squish also supports zone wildcards. For example, the following command:

```
     Route Crash 2:123/456 2:All
```

would take mail for all addresses in zone 2, compress it, and send it all through 2:123/456.

Finally, the 'World' wildcard can be used to specify all uncompressed and normal-flavoured packets, no matter where they are addressed. This is typically useful for "clean-up" situations and for taking care of mail that has no applicable routing commands. 'World' is equivalent to 'All:All'. For example, the following command:

```
     Route Crash 1:987/654 World
```

would cause all remaining, normal-flavoured mail to be compressed and sent to 987/654.

As before, all of these wildcards can be used for both the Send and Route commands, in addition to most of the other commands in ROUTE.CFG and SQUISH.CFG.

IMPORTANT NOTE FOR ARCMAILATTACH SYSTEMS:

By default, Squish leaves outbound packets in an uncompressed form. However, ArcmailAttach mailers only recognize compressed archives, so you MUST ensure that all packets that Squish creates are compressed. To do this, you must add the following statement to the end of your ROUTE.CFG:

```
     Send Normal World
```

This instructs Squish to archive all remaining packets, and to give the resulting archives a normal flavour. This makes sure that your mailer can see all of the packets that Squish generates, even if you have not added specific routing commands for an individual node.

Address Abbreviations

Although full 4D addresses can be specified almost everywhere, Squish also permits several shortcuts to save on typing. When Squish encounters a node number, it will save a copy of that address. Later, if Squish comes across an incomplete address, it will use that saved copy to fill in missing information. Squish can use short forms to handle zone and net numbers; however, a node number must always be specified.

For example, the following Route command:

```
     Route Crash 1:12/34 1:12/45 1:23/45 2:23/56 2:23/67 2:34/67
```

could be rewritten as follows:

```
     Route Crash 1:12/34 45 23/45 2:23/56 67 34/67
```

which has the same effect. In this case, Squish assumes zone 1 and net 12 when it comes to the lone "45", based on the previous address. The "23/45" address is also assumed to be in zone 1, again because of the earlier zone 1 address. The "2:23/56" is used to start a new set of zone and net defaults, so the full address is required. As with "45, the "67" assumes the zone and net numbers of the last address. Finally, a zone number of 2 is assumed for the last address because of the earlier "2:".

Examples

This section contains several simple routing files, including comments, which should demonstrate the uses of the various routing commands. In ROUTE.CFG, comment lines start with either a semi-colon (;) or a percent sign (%). (Everything on the line following either of those characters will be ignored.)

Example 1: Simple ArcmailAttach routing

```
% Sample routing file #1.  This example demonstrates a minimal
% system configuration.  This type of routing file is the most
% useful when using the ArcmailAttach keyword:  since all
% routing is performed by your mailer, the only purpose for
% ROUTE.CFG is to tell Squish which messages to compress.
%
% This routing file simply compresses all mail and sends it
% to the mail's final destination.  (However, these messages
% can be routed by your mailer using dynamic routing.)

     Send Normal World

% The above statement tells Squish to take mail for everyone
% (ie. 'World'), place it into an archive, and give it the normal
% flavour.  Unless you have a special configuration, this is all
% of the routing that you have to do for ArcmailAttach systems.
```

Example 2: Simple BinkleyTerm Routing

```
% Sample routing file #2.  This example demonstrates a minimal
% system configuration for a BinkleyTerm mailer.  This example
% assumes several things:
%
% 1) You will be connecting with another node on a fairly
%    frequent basis, and you'll be routing most of your
%    long-distance netmail through this system.  (Most
%    Net Echo Coordinators are willing to route netmail
%    for little or no charge, but you should ask before
%    doing so.)
%
% 2) You want to send mail directly to nodes in your own
%    net.  Since those nodes are local to your system, sending
%    netmail directly is usually much faster.
%
% 3) You may also be sending netmail to selected long-distance
%    nodes, and you want EchoMail for those systems to be sent
%    directly, as opposed to being sent through your local
%    coordinator.
%
% This example also uses a fictitious network address of
% 123/456.  Obviously, your own net and node number should
% be substituted for this address, or -1/-1 if you have no
% network address.

% The first command instructs Squish to send netmail directly
% to all systems in net 123.  Mail will be archived, given
% the "crash" flavour, and sent directly to its destination.
%
% Most of your mail will probably be to and from local systems,
% so sending it directly is usually the best route.

    Send Crash 1:123/All

% The next command also tells Squish to send mail directly
% to nodes 111/222, 222/333 and 333/444.  These addresses
% are presumably long-distance nodes with which you talk
% frequently, and you therefore want to send your
% NetMail directly, as opposed to routing it.  If you
% don't communicate with any long-distance nodes, this
% command can be commented out.
%
% However, if you are running any EchoMail conferences of
% your own, and you are feeding the conference to other nodes
% from your system, you should include the addresses of those
% nodes here.  It is considered to be impolite and against
% FidoNet policy to route unsolicited EchoMail through
% other systems, so you should usually send EchoMail
% directly.  All of the routing commands apply to both
% NetMail and EchoMail, so you should make sure that you have
% added the appropriate commands to deal with systems that
% receive EchoMail from you.

    Send Crash 1:111/222 333/444 555/666

% The third and last command tells Squish to send mail for
% everywhere else to 1:123/3, which is presumably your area
% coordinator.
%
% Notice how this statement is positioned below all of the
% other routing commands.  Since Route and Send only operate
% on uncompressed packets with a normal flavour, this command
% ignores mail which has been processed by the two other
% commands; it will only route mail which has not been already
% processed.
%
```

```
% However, if this command was mistakenly placed above the
% other two commands, you would find that ALL of your mail
% was being sent to 123/3 regardless.  Since the mail
% is archived and changed to a crash flavour by this
% command, the following Send commands would not find any
% mail to process, which was probably not your original
% intent.

     Route Crash 1:123/3 World
```

Batch Files

After adding the appropriate routing commands, the final step in the installation is to modify the batch files for your mailer and your BBS. At a bare minimum, Squish must be run in two different situations:

1) After receiving mail. When your mailer receives mail from another system, Squish needs to be executed to decompress the mail and import it into your own message base. Optionally, Squish can also scan out or export mail at the same time, if you are sending an echo to someone else.

2) After entering messages locally. When a message is entered, either through a BBS or an external editor, Squish must be called to export the locally-entered messages. There are several variations on this theme (after entering EchoMail, after entering NetMail, or both), but the same general process is followed for each variation.

Configuring your Mailer

First of all, your mailer must be configured to either run an external program when mail has been received, or else to drop back to the calling batch file with an errorlevel. If you don't know how errorlevels work, you should refer to the Maximus Operations Manual for a hand-holding tutorial. Section eight of the Maximus installation covers errorlevels and batch files. This gives a general overview of concepts and terminology, which will be helpful when setting up Squish.

Assuming that you have batch file basics covered, the first step is to find out which errorlevel your mailer uses when mail is received. For BinkleyTerm, this errorlevel is specified by the 'E2' and 'E3' flags in your BINKLEY.EVT configuration file. For FrontDoor, this errorlevel is specified in the Mailer / Errorlevels / ReceivedMail option in SETUP.EXE. For other systems, please consult your mailer's documentation.

The next step is to invoke Squish when the specified errorlevel is found. Assuming that your mailer exits with an errorlevel of 100 after receiving mail, the following should be placed in your batch file, immediately after running your mailer:

```
:Loop
bt unattended                       ; Other commands here
if errorlevel 100   goto SquishIn   ; Toss messages
if errorlevel  96   goto BBS        ; Call a BBS program
if errorlevel  48   goto BBS         ; Call a BBS program
... and so on ...
```

Only the lines labelled ':Loop' and 'SquishIn' are required by Squish itself. The other lines should have be added so that your BBS is run for human callers; they are not mandatory for Squish's operation, and the errorlevels may be different (depending on which BBS package you run). The errorlevel procedure is the same for FrontDoor, except that 'fd' should be substituted for 'bt unattended'. Please consult your mailer's documentation to learn how to start up a different type of mailer.

The ':Loop' label should be placed just before your mailer's command-line. After tossing and scanning mail, Squish will jump back up to this label and restart your mailer.

The 'if errorlevel 100' statement checks for the "received mail" errorlevel, and if found, it jumps to the part of the batch file which starts up Squish.

Keep in mind that all errorlevels must be given in DESCENDING ORDER. When interpreting each 'if errorlevel' statement, the operating system performs a check to see if the current errorlevel is GREATER THAN OR EQUAL TO the specified errorlevel. To ensure that each line is executed only when you want it to, all of the errorlevels have to be in descending order.

After adding the check for receiving mail, you should now create the portion of the batch file which actually invokes Squish. Near the end of your batch file, but before any final 'exit' or ':end' statements, add the following:

```
:SquishIn
cd\Squish
squish in out squash link
cd\Bink
goto Loop
```

This does five things:

1) The ':SquishIn' label is referenced by the earlier 'if errorlevel' statement. This is where your batch file will jump to if mail is received.

2) The 'cd\Squish' changes the current directory to \Squish,

which is presumably where SQUISH.EXE and its associated configuration files are kept.

3) The 'squish in out squash link' command starts Squish in one-pass mode. This command will cause Squish to check for incoming mail, decompress it, toss it to the local message bases, scan it out to other nodes (if necessary), pack messages and create ARCmail attaches in your netmail area, and finally, to link reply chains. For more information on Squish's command-line parameters, please see the section entitled "Squish Command Line Parameters and Syntax".

4) The 'cd\Bink' command tells DOS to change back to your mailer's directory. (If you are running FrontDoor, this will probably be 'cd\FD' or something similar.) Unless this line is included, your mailer may not operate properly after Squish runs.

5) The 'goto Loop' command causes your batch file to loop back to the top again, and to restart your mailer. Unless this command is included, Squish would simply drop back to the operating system, or possibly execute other unwanted commands in your batch file. To make sure that this looping works, ensure that there's a ':Loop' label right above the line that calls your mailer. (See above for more details.)

Configuring your BBS

After making the above modifications, Squish should be fully operational when receiving mail. The only other modification you need to make are to your BBS's or off-line reader's batch files. The procedure is similar: determine which errorlevel is used when mail is entered, and jump to the appropriate portion of the batch file.

For example, Maximus uses the following errorlevels:

Errorlevel 12:	Caller entered EchoMail (and maybe NetMail)
Errorlevel 11:	Caller entered NetMail (but not EchoMail)
Others:	Caller entered neither NetMail nor EchoMail

The two separate cases -- NetMail/EchoMail, and NetMail only -- must be treated differently. When EchoMail is entered (and possibly NetMail as well), the echoes must be scanned for messages to export, and the netmail area must be packed (or scanned for ARCmail attaches). However, when only NetMail is entered, only the packing/scanning needs to be performed. What follows is a simple batch file which demonstrates how to implement Squish on a Maximus system. This assumes that you have configured Max's "Log EchoMail" feature and that it points to the file C:\MAX\ECHOTOSS.LOG.

```
max -p%1 -b%2 -t%3      ; Other parameters here
if errorlevel  12   goto SquishOut
if errorlevel  11   goto SquishSquash
if errorlevel   5   goto Recycle
... and so forth ...
```

As before, only the 'SquishOut' and 'SquishSquash' lines are required by Squish.

After taking care of the other errorlevels, you should insert the following two sections in your BBS's batch file:

```
:SquishOut
cd\Squish                                ; Chg to Squish dir.
squish out squash -fc:\max\echotoss.log  ; Export messages
del c:\max\echotoss.log
goto End

:SquishSquash
cd\Squish
squish squash                      ; Pack msgs and do ArcAttaches
goto End
```

The section marked ':SquishOut' invokes Squish and instructs it to scan the EchoMail areas listed in ECHOTOSS.LOG. If your BBS program or off-line reader is not capable of generating an ECHOTOSS.LOG, then simply omit the '-fc:\max\echotoss.log'. Omitting that command-line option will cause Squish to scan all EchoMail areas every time it is run, which is usually much slower than using ECHOTOSS.LOG for scanning. (Please see the section entitled "Squish Command Line Parameters and Syntax" for information on the format of ECHOTOSS.LOG.)

After exporting messages, the 'squash' portion of the command line tells Squish to pack netmail messages, create ARCmail attaches, and execute the commands in ROUTE.CFG.

The ':SquishSquash' portion of the batch file does the same thing, except that it skips scanning the EchoMail areas, and simply runs Squish in a mode which packs netmail messages and executes ROUTE.CFG.

This concludes the Squish installation procedure. If you have followed all of these steps, you should have a working version of Squish that tosses, scans, packs, links, slices and dices. For advanced tricks and tips on operating Squish, please read through the rest of this manual at your leisure.

OPERATION

Squish Command Line Parameters and Syntax

Squish operates in several modes, all of which are controllable from the command line. The command line itself is broken down into a series of "actions" and "switches". Actions are single words which control Squish's processing modes, such as tossing and scanning messages. Switches are used to modify the behaviour of those modes, such as only scanning certain areas, not displaying any output, and so on. The format of the command line is as follows:

```
     SQUISH <switches> <mode>
```

Modes

<mode> consists of one or more of the following keywords:

IN — Toss (import) messages. This option causes Squish to check for packets and compressed mail archives in all of the NetFile directories, and to import any messages that it finds.

OUT — Scan (export) messages. This option causes Squish to scan all EchoMail areas for new messages. If Squish finds a new message, that message will be exported to all of the nodes listed in that area's definition. Squish uses the SEEN-BYs to keep track of which nodes the message has already been sent to, so the message may not be exported to all of the nodes listed.

If both the IN and OUT actions are specified on the same command line, Squish will function in "single pass" mode. "Single pass" means that Squish will both toss and scan messages at the same time (which yields higher performance). Single pass mode is especially useful for systems running many "passthru" areas, since messages will be written directly to the output *.PKT files, as opposed to making a temporary stop in a message area. Consequently, using passthru message areas in multipass mode is MUCH slower than using passthru areas in single-pass mode.

However, you should ONLY use the "IN" parameter when there are messages to be tossed. When running with "IN" and "OUT", Squish will only scan those areas to which messages are being tossed. Therefore, when you wish to only scan messages, the `IN' parameter should NOT be specified.

When the OUT command is specified alone, Squish will scan all EchoMail areas on the system. When used in conjunction with the IN command, Squish will scan messages as it tosses. In addition, if Squish detects that unsent messages exist in an EchoMail area, Squish will invoke a full scan of that area before tossing to it.

SQUASH — For a BinkleyTerm system, the SQUASH command packs messages from the NetMail areas, scans the outbound areas, performs mail routing, and compresses packets.

For a FrontDoor-style system, the SQUASH command compresses packets, scans the NetMail area, creates ARCmail attach messages, and optionally kills any orphaned archives.

If the SQUASH command is specified on the same command line as IN and OUT, the single pass MaxMsgs mode can also be used to limit outbound packet sizes.

LINK — Relink reply chains. The LINK command will read all messages in an EchoMail area and create reply links for each message (based on the subject fields). Relinking allows other software to perform "threaded reading" on message bases, which is an easy and convenient way of viewing messages.

When combined with the IN and OUT commands, only those areas which received messages will be linked. By default, the LINK command only processes EchoMail areas. However, by using the "-f" parameter to specify one explicit area, a link can be forced for any type of area. (For example, "squish link -fNETMAIL" will force the NetMail area to be linked.)

RESCAN — Rescan one or more EchoMail areas. The RESCAN command provides a convenient way to rescan EchoMail areas from the command line, instead of fiddling with 1.MSG and other system files.

Unlike the other modes, RESCAN cannot be combined with other switches or actions. The format for the RESCAN command is:

```
SQUISH RESCAN <echo_or_tosslog> <node>
```

<echo_or_tosslog> is either the tag of an EchoMail area, or the name of an ECHOTOSS.LOG-format file (containing a list of area tags).

<node> specifies the node number for which the specified EchoMail areas should be rescanned.

After processing the command line, Squish will immediately rescan ALL of the messages in the specified areas. The SEEN-BYs will not be updated in the on-disk message base, the high water mark will be left untouched, and the messages will not be sent to anyone except the listed node.

This command was designed for NECs and other EchoMail hubs who wish to rescan entire message bases for nodes who have just connected to a particular echo.

Warning! If you wish to specify command line switches in conjunction with the RESCAN command, those switches must precede the rescan command on the command line. In other words, to leave all packets in the OUTBOUND.SQ area, use `SQUISH -L RESCAN AREANAME NET/NODE'.

GET — This command instructs Squish to request a file from a node given on the command line. THIS COMMAND IS NOT SUPPORTED FOR ARCMAILATTACH SYSTEMS.

The format of the GET command is shown below:

```
SQUISH GET <filename>[!<pwd>] [FROM] <node> [flavour]
```

<filename> is the name of the file to be requested. The filename can contain wildcards.

<pwd> can be used to specify the password for a password-protected file. A single exclamation mark must separate the filename and password.

The optional literal "FROM" can be used to make the Squish command line more readable.

<node> specifies the node number to which the file request should be sent.

<flavour> specifies the optional flavour of the request. It can be one of CRASH, DIRECT, HOLD, or NORMAL (the default).

Examples:

```
SQUISH GET FILES FROM 249/106 CRASH
```

Request "files" from 249/106 using a "crash" priority.

```
SQUISH GET TEST.ZIP!MYPWD 2:254/1
```

Get "test.zip" from 2:254/1 using a password of "mypwd".

UPDATE This command instructs Squish to perform an update request for the node given on the command line. THIS COMMAND IS NOT SUPPORTED FOR ARCMAILATTACH SYSTEM.

The format of the UPDATE command is shown below:

```
SQUISH UPDATE <filespec>[!pwd] [FROM] <node> [flavour]
```

The command-line syntax of the UPDATE command is identical to that of the GET command. However, this command instructs Squish to perform an update request instead of a simple file request. In addition, a fully-qualified pathname must be provided, instead of just a filename.

SEND This command instructs Squish to send a file to a node given on the command line. THIS COMMAND IS NOT SUPPORTED FOR ARCMAILATTACH SYSTEMS.

The format of the SEND command is shown below:

```
SQUISH SEND [^|#]<filename> [to] <address> [flavour]
```

<filename> gives the name of the file that is to be transmitted to the remote system.

If a "#" is placed in front of the filename to send, the file will be truncated after it is sent.

If a "^" is placed in front of the filename to send, Squish will delete the file after it is sent.

IMPORTANT NOTE FOR OS/2 AND 4DOS USERS: Since the "^" character has special significance on the OS/2 and 4DOS command lines, one must enclose the entire filename (including the "^") in quotes.

Examples:

```
SQUISH SEND C:\CONFIG.SYS TO 249/1 CRASH
```

Send c:\config.sys to 249/1 using a priority of CRASH.

```
SQUISH SEND "^E:\Max\Max.Exe" to 106/114 Hold
```

Send e:\max\max.exe to 106/114 using a priority of HOLD. The file will be deleted after it is sent.

```
SQUISH SEND #e:\*.* 273/703
```

Send all of the files in the root directory of E: to 273/703, using a priority of NORMAL. The files will be truncated after they are sent.

POLL This command instructs Squish to poll a specified node.

The format of the POLL command is as follows:

```
SQUISH POLL <address> [flavour]
```

<address> is the node which is to be polled.

Examples:

```
SQUISH POLL 1:249/106 Crash
```

Generate a crash poll to 1:249/106.

```
SQUISH POLL 2:253/68
```

Generates a normal poll to 2:253/68.

With the exception of RESCAN, SEND, GET, UPDATE and POLL, all of these modes can be specified at the same time for optimum performance. The command "SQUISH IN OUT SQUASH LINK" instructs Squish to toss and scan messages in single pass mode, to pack messages from the NetMail area, to route and compress packets, and to relink reply chains.

Certain combinations of the Squish command line parameters are useful only in certain situations. Namely, the "IN" command must only be used when there are packets to be tossed, so care must be taken to use "IN" only when necessary.

The following command lines are recommended for various situations:

After receiving EchoMail or NetMail from another system:

```
     SQUISH IN OUT SQUASH LINK (with an optional -fECHOTOSS.LOG)
```

After EchoMail and/or NetMail has been entered locally:

```
     SQUISH OUT SQUASH LINK (with an optional -fECHOTOSS.LOG)
```

After NetMail has been entered locally:

```
     SQUISH SQUASH
```

Command Line Switches

In addition to <mode>, Squish accepts any of the following command line switches:

-a<file>	The -a switch instructs Squish to use the specified AREAS.BBS-like file, overriding the filename specified in SQUISH.CFG.
-c<file>	The -c switch instructs Squish to use a configuration file with the specified path and filename, as opposed to looking for SQUISH.CFG in the current directory. This switch will override the config name specified by the SQUISH environment variable (if any).
-f<file>	The -f switch specifies the name of an ECHOTOSS.LOG-type file which contains a list of echo tags to process. When used in conjunction with the IN command, Squish will create a log of all non-passthru echoes that received messages during the current session. When used in conjunction with the OUT or LINK commands, Squish will only process echoes listed in the ECHOTOSS.LOG file. Although the -f switch was designed to be used with a tosslog filename, it can also specify the name of a single EchoMail area to be processed. For example, "SQUISH OUT -fMUFFIN" would instruct Squish to scan the MUFFIN echo for new messages. WARNING! Squish will never delete ECHOTOSS.LOG. When tossing, Squish will simply append to the log you specify (to fully supporting multi-line systems). It is assumed that your batch files will delete

ECHOTOSS.LOG after performing all necessary processing; this must always be done, or else ECHOTOSS.LOG will grow and grow after each successive invocation of Squish.

-l	The -l switch (BinkleyTerm systems only) instructs Squish to leave *.OUT packets in the OUTBOUND.SQ holding area. Before Squish terminates, it normally moves all unrouted packets from OUTBOUND.SQ to the proper zoned outbound area. However, if you are running Squish in a multipass environment, you may wish to leave the packets in the OUTBOUND.SQ area between passes to shield the packets from other programs on multitasking or multi-line systems.
-n<file>	The -n switch instructs Squish to use a different log file for the current run only. This switch overrides the LogFile keyword in SQUISH.CFG.
-o	The -o switch (BinkleyTerm systems only) instructs Squish NOT to pack messages in the NetMail area when performing a SQUASH command. Squish will still scan the outbound area and archive packets, but no NetMail messages will be processed. If you need to run an external utility over your NetMail area (such as a message tracker or readdresser), this option will allow you to use MaxMsgs with a one-pass IN OUT SQUASH command line. Squish can be run with IN OUT SQUASH and the -o switch, and then your external utility can be run, followed by a separate SQUISH SQUASH without the -o switch.
-q	The -q switch enables "quiet mode". Quiet mode suppresses most of the information displayed while a packet is tossing, although Squish will still display the name and origination addresses for each packet. Although this option makes the Squish screen display less informative, the -q makes Squish run slightly faster.
-s<sched>	The -s switch instructs Squish to process the specified schedule when performing the SQUASH command. If no schedule is specified, Squish will execute the global commands, plus any schedule which is within the current time period. If an explicit schedule is specified on the command line, only that schedule and the global commands will be executed.
-t	The -t switch toggles the status of Squish's secure

mode. In other words, if you have `Secure' turned ON in SQUISH.CFG, using -t will temporarily turn OFF Secure more. Likewise, if you have Secure turned OFF in SQUISH.CFG, using -t will temporarily turn ON secure mode.

-u	The -u switch toggles the use of the TossBadMsgs keyword. If TossBadMsgs is enabled in SQUISH.CFG, this switch will disable TossBadMsgs processing. If TossBadMsgs is not enabled in SQUISH.CFG, this switch will enable TossBadMsgs processing.

-v	The -v switch toggles the use of "Statistics Mode". If "Statistics" is turned ON in the configuration file, this switch will temporarily disable statistics mode. Similarly, if "Statistics" is turned OFF in the configuration file, -v will temporarily turn it back on. This switch can be used to create statistics information only when mail is received from a certain node.

-z	The -z switch instructs Squish to scan non-passthru areas only. Normally, when Squish is invoked with the OUT command, it will scan all of the areas defined in SQUISH.CFG and/or AREAS.BBS. However, if you normally do not keep any messages in your passthru areas, this switch will cause Squish to skip those areas and marginally speed up processing.

Running Squish in Multipass Mode

The easiest and most efficient way to run Squish is in "single pass" mode, meaning that Squish will toss, scan, and pack messages all at the same time. However, Squish can also be run in multipass mode, meaning that tossing, scanning and packing can be separated into two or three different passes. Some of the reasons for using multipass mode are:

* Decreased memory requirements. When running in single pass mode, Squish needs enough memory to buffer the packet being tossed, to hold the incoming messages, and buffers for outbound messages. When running in multipass mode, Squish only needs to keep one part of the message in memory at a time, thereby reducing memory requirements of the DOS version by 60K or more.

* Some preprocessing utilities may need to be run between passes. For example, several utilities may need to be run over the message bases after a message has been tossed, but before that message is scanned out to other systems. Multipass mode allows for this type of external utility.

The key to multipass mode is the use of ECHOTOSS.LOG. The -f switch should be used to keep a log of tossed messages, such that Squish scans and links only those areas which received messages.

Invoking Squish in multipass mode is usually done in the following manner:

```
SQUISH IN -fECHOTOSS.LOG

rem * Other external utilities here

SQUISH OUT -fECHOTOSS.LOG

rem * Other external utilities here

SQUISH SQUASH

rem * Other external utilities here

SQUISH LINK -fECHOTOSS.LOG

del ECHOTOSS.LOG
```

Depending on your processing requirements, you may wish to run two passes at the same time (such as SQUASH and LINK, or IN and OUT), but the general format remains the same. Squish will be somewhat slower when running in multipass mode, especially when the IN and OUT passes are separated. However, this method of operation allows for flexibility and allows other external utilities to be inserted into the tossing/scanning process.

SQ386 Usage Notes

The 32-bit DOS version of Squish uses the DOS/4GW extender from Rational Systems, Inc. This file is included in the Squish distribution as DOS4GW.EXE. The DOS extender is automatically loaded whenever SQ386.EXE is run. DOS4GW.EXE must be in the same directory as SQ386.EXE, and it must be on your path.

To operate properly, SQ386 requires either an XMS or a DPMI memory driver:

Under DOS, this means that you should have HIMEM.SYS installed, or an equivalent DOS memory manager (such as QEMM or 386^MAX).

Under OS/2 or Windows, nothing needs to be done. Both OS/2 and Windows act as DPMI hosts.

The 32-bit version of Squish is also memory-hungry. While SQ386 does not require much conventional memory, A MINIMUM OF TWO MEGABYTES OF XMS/DPMI MEMORY IS REQUIRED. In general, SQ386's performance will increase as you give it more memory.

For more information on the 32-bit version of Squish, please see the installation subsection titled, "16-bit or 32-bit? SQ386 and SQ386P".

AREAS.BBS REFERENCE

Squish supports the definition of EchoMail areas in the standard AREAS.BBS file. However, this format was designed for use with older systems, and as such, it doesn't support many of the new features that Squish offers. Regardless, some users may find it advantageous to declare areas in AREAS.BBS for compatibility reasons.

The first line of AREAS.BBS is always your default origin line; this text will be added to messages which do not already have an origin line. After the text of the origin line, you should place an exclamation mark, followed by your name. Squish doesn't use the SysOp name, but other programs may require it.

For example, the first line in AREAS.BBS might look like this:

```
Fowl Weather Post * Kingston, Ont., Canada !Scott Dudley
```

Following the default origin line should be one or more EchoMail area definitions. An EchoMail area definition has the following format:

```
     [#][$]<path> <tag> [nodes...]
```

Both the '#' and '$' characters are optional. If a '#' is placed before the path, that area will be treated as a passthru area. (For more information on passthru areas, please see the section entitled "Area Definitions" in the SQUISH.CFG reference.)

If a '$' is placed before the path, that area will be treated as a Squish (*.SQ?) format message area. (For more information, please see the section entitled "USING SQUISH-FORMAT MESSAGE AREAS".)

Both the '#' and '$' characters should come immediately before the first character of the area's path. No spaces are permitted.

<path> should specify the path of the EchoMail area. For a *.MSG area, this should be the name of directory in which the *.MSG files are kept. For a Squish (*.SQ?) area, this should specify the path and root filename of the message area.

<tag> should be the one-word area tag to be used for that area.

[nodes] is the optional list of nodes to whom you send that echo.

For examples, please see the distribution copy of AREAS.BBS.

SQUISH.CFG REFERENCE

SQUISH.CFG is the main Squish configuration file. SQUISH.CFG controls most aspects of Squish's operation, including the message areas it uses, the directories where it places files, and even how to remap messages addressed to certain users.

SQUISH.CFG is divided into two logical sections: firstly, there is the main configuration section. This section contains information about your system and everything else that Squish needs to know. Secondly, there is an optional area configuration section. This part of the configuration file allows you to define NetMail, EchoMail, dupe storage and bad message areas for your system.

SQUISH.CFG is an ASCII file, so it can be edited using a text editor (or a word processor in non-document mode). Lines which begin with a semicolon (;) are treated as comments and are ignored.

Configuration Options

This section describes all of the commands and options available in the system information section of SQUISH.CFG. For information on defining EchoMail, NetMail, bad message or dupe storage areas, please see the next section.

The following is an alphabetical listing of keywords in SQUISH.CFG. Descriptions are given for all keywords, and examples are given where appropriate.

AddMode (BinkleyTerm systems only)

> The AddMode keyword enables the "add mode" functionality of the outbound directory manager. Add mode causes new file attaches to be merged into existing attaches of different flavours. When add mode is turned on, Squish will perform some special processing before creating a file attach. Instead of simply creating an attach with the specified flavour, it will first scan the outbound directory to see if any other attaches exist for the same node. If a non-normal-flavoured attach is found, add mode will cause Squish to add the current file to that attach, regardless of the flavour specified for the "Route" or "Send" commands.
>
> This verb is useful in many situations, such as when a node goes down for a short period of time. With add mode turned on, simply change the flavour of the existing attaches in the outbound area to "hold". When Squish scans out more

mail for that node, it will notice the hold attach and it will hold all new mail as well. This means that no modifications to your routing control files are necessary, even if you are using "Route Crash" or "Send Crash" for that particular node.

By default, with add mode disabled, Squish will simply use the given flavour, without attempting to check for other existing attaches.

Address <node> [<node>...]

The Address keyword defines one or more network addresses used by your system. The first address specified in SQUISH.CFG is considered to be your "primary address", and this address will be used for all outbound mail (unless told otherwise -- see the documentation on the EchoArea keyword in the "Area Definitions" section of the SQUISH.CFG reference).

Squish handles full 4D addresses and is capable of running as either a bossnode or a point.

For a normal network node, only one address statement is required. Your primary address should be declared in this format:

```
Address 1:123/456
```

where "1:123/456" is your full network address, including zone number.

For fakenet points, two addresses are required. The first address declared in SQUISH.CFG should be your fakenet address, and the second address should be your full 4D point address. Both addresses must include the proper zone number. For example, a fakenet point at 1:123/456.1 (using a fakenet of 12345) might use the following set of address statements:

```
Address 1:12345/1
Address 1:123/456.1
```

In addition, fakenet points must also use the "PointNet" verb, below.

For 4D points, only one address is required. This address should be your full 4D point address, and it should be declared as the first address in your configuration file.

For example, a 4D point of 1:123/456 might use the following address statement:

```
Address 1:123/456.1
```

However, to use 4D points, all of your software must be able to handle 4D addresses, and your mailer and EchoMail processors must compliant with the "2+" packet type.

After you have defined your primary address, any number of AKA (Also-Known-As) addresses may be listed. These addresses won't be used for outgoing mail, but Squish will use these AKAs to determine whether or not a packet is destined for your system.

Any number of AKAs may be specified, up to the limit of available memory.

AddToSeen <node> [<node>...]

The AddToSeen keyword globally adds a node number to the SEEN-BYs of all EchoMail areas which are processed by your system. The AddToSeen keyword specifies one or more two-dimensional address (net/node only) which are to be added. This keyword adds these addresses to ALL echoes, so it should be only used in special situations. By default, your primary address is added to the SEEN-BYs for all areas. For information on changing your primary address, or for adding to the SEEN-BYs on an area-by-area basis, please see the "Flags" section of the SQUISH.CFG reference.

ArcmailAttach

The ArcmailAttach keyword informs Squish that your mailer requires "ARCmail file attaches" to transmit EchoMail to other systems. Mailers such as FrontDoor, D'Bridge, and InterMail require this keyword, whereas others (such as BinkleyTerm) do not. If you are not running any of the above pieces of software, please consult your mailer's documentation to determine whether or not it requires ARCmail attaches.

When this keyword is enabled, Squish operates differently in several ways:

* Packing of the NetMail area is disabled. Mailers which require ARCmail attaches perform dynamic packing on their own, so this portion of the Squish code is automatically disabled.

* ARCmail attach messages will be generated for outbound EchoMail. This support is internal to Squish, so no external utilities are required.

* Most of the routing code for the SQUISH SQUASH command is disabled. Again, since ARCmail attach mailers perform dynamic routing, these features are not required.

* Squish will only write packets to the OUTBOUND.SQ area, and it will disable the use of BinkleyTerm-style "zoned outbound directories".

* Squish will enable 4D point support, since most ARCmail attach mailers are capable of handling 4D points.

By default, ARCmail attach support is disabled. This means that:

* Squish will perform packing of the NetMail area. Squish will gateroute messages, handle file attaches, file requests and update requests, in addition to the optional deletion and truncation of attached files. Messages will be packed from all areas declared using the 'NetArea' keyword.

* Squish will perform static routing and packing in the outbound areas. If the BinkPoint verb is implemented, Squish will enable support for 4D points on a BinkleyTerm-style system.

* Squish will directly process BinkleyTerm-style zoned outbound directories and enable full support for all commands in ROUTE.CFG.

Unless the ArcmailAttach verb is set correctly, Squish may produce unpredictable and unreliable results on your system. Make sure that this verb is set appropriately before running Squish.

AreasBBS <filespec>

The AreasBBS keyword specifies the path and name of a ConfMail-compatible AREAS.BBS file, which is one way of defining EchoMail areas. Please see the section entitled "AREAS.BBS REFERENCE" for more information on using this command. Note: the '-a' command-line switch can also be used to override the filename given for this command.

BatchUnarc

The BatchUnarc keyword instructs Squish to decompress all compressed mail bundles at the same time. By default, Squish will decompress one mail bundle, toss all of the packets in it, decompress the next bundle, toss those packets, and so on. Although this method makes efficient use of disk space, there is a slight chance that messages may get out-of-order, depending on the order in which the compressed mail archives arrived on your system.

The BatchUnarc keyword instructs Squish to decompress all of the compressed mail archives at once, and to then toss the *.PKT files. Squish always sorts packets by date before tossing, so this ensures that messages will not get out of order.

BinkPoint (BinkleyTerm systems only)

The BinkPoint keyword enables the BinkleyTerm 2.50+ point directory support. When this keyword is enabled, Squish will turn on full 4D address support, and Squish can then communicate freely with 4D points running BinkleyTerm.

If you are using 4D points with Binkley 2.50 or above, BOTH the boss and the point must have this keyword enabled.

Buffers <type>
   or
Buffers <writebuf> <outbuf> <msgbuf>

The Buffers keyword can be used to limit Squish's memory usage. Two different forms of the keyword can be used:

With "Buffers <type>":

<type> is any of "Small", "Medium" or "Large". These types are just short-hand for certain predefined buffer sizes. (See below for more information.)

With "Buffers <writebuf> <outbuf> <msgbuf>":

<writebuf> controls the amount of memory used to buffer writes to disk. (This parameter does not provide for much of a speed increase beyond 16k.)

<outbuf> controls the amount of memory used to buffer messages for various nodes. This parameter controls how often Squish has to stop tossing and place packets in the outbound area. For systems which forward mail

to many nodes, this parameter will have a significant impact on processing speed.

<msgbuf> controls the maximum message size that Squish is able to process, for either inbound or outbound mail. Under the 16-bit versions of Squish, this parameter is limited to 63K.

The defaults for the 16-bit versions of Squish (SQUISH and SQUISHP) are:

BUFFERS	writebuf	outbuf	msgbuf
Small	16k	16k	16k
Medium	26k	40k	32k
Large	57k	57k	63k

The defaults for the 32-bit versions of Squish (SQ386 and SQ386P) are:

BUFFERS	writebuf	outbuf	msgbuf
Small	64k	64k	64k
Medium	76k	128k	128k
Large	128k	512k	256k

The buffer sizes shown for "Small" are the lowest values that can be specified. In addition, buffer sizes for the 16-bit versions of Squish cannot be larger than 63K.

Note that the small, medium and large buffer sizes have changed from Squish 1.01. To use the same buffer sizes as the "large" mode of Squish 1.01, use:

```
Buffers 57 57 16
```

If no "Buffers" statement is specified, Squish will use the LARGE setting for writebuf and outbound and the MEDIUM setting for msgbuf.

Keep in mind that while reducing the level of buffering will provide the DOS versions of Squish with more memory, reducing the buffer sizes will also have an impact on speed. If you want Squish to run as fast as possible, use large buffer values. If you want Squish to run in as little memory as possible, use small buffers. If you want a compromise, try using medium buffers.

Other tips for reducing memory usage:

* Disable the "Statistics" keyword. Keeping statistics almost doubles Squish's memory requirements for storing AREAS.BBS information.

* Reduce the "MaxAttach" and "MaxPkt" settings. A value of 64 for each setting works for most systems.

* Reduce the "Duplicates" setting. However, since this impairs Squish's ability to identify duplicate messages, doing so is not recommended.

* Run SQUISH SQUASH in a separate pass from SQUISH IN OUT. If memory is really tight, run everything in its own separate pass. (See the section entitled "Running Squish in Multipass Mode" in the OPERATION section for more information.)

BusyFlags

The BusyFlags keyword enables "busy flag" support for BinkleyTerm and InterMail systems. This option is required when using more than one copy of your mailer with the same outbound area; the busy flags provide for file locking, which prevent collisions when two different nodes attempt to send mail at the same time.

Squish has full support for the busy flag mechanism. To further help throughput, Squish also performs most of its operations in a separate holding directory, outside of the main zoned outbound directories. Files are only transferred to the outbound directories as necessary, which minimizes the "down time" for any given node.

In addition, if Squish attempts to send mail to a node which is busy due to mailer activity on another line, the packet will be simply queued for later use, as opposed to waiting for that node to become free again.

Unlike other mail processors, this means that Squish never has to wait because of mail activity occurring on other lines of a multi-line system, which makes Squish the ideal choice for a multi-line installation.

If ArcmailAttach mode is enabled, Squish uses a busy flag format that is compatible with InterMail (and certain other ArcmailAttach mailers).

If ArcmailAttach mode is not enabled, Squish uses BinkleyTerm-style busy flags.

When using BinkleyTerm, make sure to turn on busy flag support by enabling the "Flags" directory and setting a non-zero task number.

CheckZones

The CheckZones keyword instructs Squish to check the zone numbers on all incoming packets. This option is enabled in the default configuration file; however, it may cause problems in secure mode with older, non-zone-aware systems. Commenting out this option tells Squish not to check the zone number when performing security checks on inbound packets, which may help if you have to deal with such older software.

Compress <filespec>

The Compress keyword gives the path and filename of the compression configuration file. Squish can use almost any external program to compress mail, including ARC, PKArc, PKZip, LHarc (both 1.xx and 2.xx), ZOO, PAK, ARJ, and more. Squish's compression support is completely user-definable, so new archivers can be added to the configuration file with ease. For more information, please see the documentation on the "Pack" and "DefaultPacker" keywords, and also the section entitled "ARCHIVES AND MESSAGE COMPRESSION".

DefaultPacker <packer>

The DefaultPacker keyword instructs Squish to use <packer> as the default packer for compressed mail. <packer> must be the name of a packer defined in the compression configuration file.

NOTE! The official standard for compressed mail in FidoNet is the version 5 ARC format. Unless you have a good reason for changing the default, ARC should be left as the default packer. Packer types can be changed on a node-by-node basis through the "Pack" verb, so DefaultPacker should only be used when necessary.

DupeCheck [<type>...]

The DupeCheck keyword controls the dupe-checking algorithm used by Squish:

<type> can be either or both of "Header" or "MSGID".

"Header" instructs Squish to check the message header to determine whether or not a message is a dupe. Squish will hash the "To", "From" and "Subject" fields into a 32-bit identifier. It will append the message date to this, resulting in a 64-bit duplicate identifier.

"MSGID" instructs Squish to check the MSGID kludge to determine whether or not a message is a dupe. Squish will hash the text of the MSGID "address" field into a 32-bit identifier. It will append the MSGID serial number to this, resulting in a 64-bit duplicate identifier.

If only one of the above settings is enabled, Squish will only use that method when determining whether or not a message is a dupe.

However, if both MSGID and Header are specified, Squish will perform both checks. If EITHER the MSGID or the header is duplicated, Squish will declare the message to be a dupe.

Duplicates <number>

The Duplicates keyword instructs Squish to keep up to <number> duplicate message IDs for each message area. By default, Squish stores the IDs of the past 1000 messages. Unlike other mail processors, Squish uses a highly-accurate 64-bit duplicate identifier, so increasing <number> will NOT significantly increase the chances of a duplicate message being falsely detected.

Feature <name>   (OS/2 only)
Feature32 <name>   (OS/2 only)

The Feature and Feature32 keywords instruct Squish to load a third-party DLL file. <name> should be the root name of the file to load, without the ".DLL" suffix.

The Feature keyword should only be used with 16-bit DLLs. The Feature32 keyword should only be used with 32-bit DLLs. (By convention, the name of the feature DLL will include a "32" if it is a 32-bit DLL.)

See the Squish Developers Kit (to be made available shortly after the release of Squish 1.1) for information on writing feature DLLs.

ForwardFrom [FILE] <node> [<nodes>...]

The ForwardFrom keyword tells Squish that the forwarding of in-transit NetMail messages is allowed, as long as the messages originated FROM any of the specified nodes. If the "FILE" keyword is added before the first node number, Squish will also properly forward in-transit files. Any number of nodes may be listed, and wildcards may be used.

NOTE! The ForwardFrom verb allows the specified nodes to forward messages ANYWHERE. If you have your routing set up to crash all messages to their destinations, this should only be enabled for systems that you trust. For a more conservative approach, see the ForwardTo keyword.

ForwardTo [FILE] <node> [<nodes>...]

The ForwardTo keyword tells Squish that the forwarding of in-transit NetMail messages is allowed, as long as the messages are destined TO any of the specified nodes. If the "FILE" keyword is added before the first node number, Squish will also properly forward in-transit files. Any number of nodes may be listed, and wildcards may be used.

This option is appropriate for most systems, since it allows anyone to forward mail to the specified systems, but to those systems only. Unlike ForwardFrom, this means that you have control over where the messages end up, as opposed to the person who sent the message.

GateRoute <flavour> <gate> <node> [<nodes>...] [EXCEPT <node> [<nodes>...]]

The GateRoute keyword causes Squish to perform gaterouting on NetMail messages addressed to the specified gate or any of the following nodes. Squish follows the FTS-0001 standard for gaterouting, so the messages produced by this command should be acceptable to SEAdog and other non-zone-aware mailers.

Gaterouting will only be performed on NetMail messages with a flavour of normal. Messages marked as crash or hold will never be gaterouted; instead, they will be classified as direct interzone messages and will be treated accordingly.

<flavour> specifies the flavour to which messages will be converted after gaterouting has taken place. The gaterouted flavour should usually be normal (so that other commands in ROUTE.CFG can compress and route the mail normally), but you can specify an alternate flavour if you so desire.

<gate> specifies the address of the system to which messages should be gaterouted. This should be the address of an official network zonegate.

<node> and <nodes> specify the list of nodes for which gaterouting is to be performed, including wildcards. All NetMail messages which are addressed to these nodes will be gaterouted through the specified gate system. (EchoMail should NEVER be gaterouted.)

Optionally, a list of exceptions can be given for each zonegate. Simply add the 'EXCEPT' keyword to the GateRoute line, and for the current GateRoute keyword, Squish will ignore all messages addressed to those nodes. Wildcards are also supported for gaterouting exceptions.

For example, to gateroute all messages going from zone 1 to zone 2, with the exception of messages destined for 2:123/456, the following gateroute statement could be used:

```
GateRoute 1:1/2  2:All Except 2:123/456
```

In zone 1 of FidoNet, the standard set of gaterouting statements for other zones looks like this:

```
GateRoute 1:1/2  2:All
GateRoute 1:1/3  3:All
GateRoute 1:1/4  4:All
GateRoute 1:1/5  5:All
GateRoute 1:1/6  6:All
```

Include <filename>

This keyword instructs Squish to read in another file as part of the standard SQUISH.CFG processing. The contents of <filename> will be processed just as if they were placed directly in SQUISH.CFG. Include files can be nested, but no more than 16 levels of nesting are supported.

KillBlank

The KillBlank verb instructs Squish to delete blank NetMail messages. Blank netmail messages are commonly generated by ARCmail attach systems, manual file requests, file attaches, and update requests. By definition, a "blank message" is a message which consists only of a message header, kludge lines and blank lines.

KillDupes

The KillDupes verb instructs Squish to delete duplicate messages as they are received, as opposed to placing the dupes in the DUPES message area. If you are encountering a large number of dupes and you already know where the problem is, this keyword can then be enabled to delete the dupes as they are tossed. However, having a copy of the dupes tossed into the DUPES area is instrumental to determining which system originated the duplicate messages, so this keyword should not be used for normal operation.

KillIntransit

The KillIntransit verb instructs Squish to delete in-transit NetMail messages. Normally, Squish leaves in-transit messages in the NetMail area for review by the SysOp, but this keyword will cause Squish to delete in-transit messages as soon as they have been packed up and sent to their destination.

KillIntransitFile (non-ArcmailAttach systems only)

The KillIntransitFile verb instructs Squish to delete in-transmit files. Squish will add a "delete-after-sending" flag when writing the *.?LO file for the forwarded file. This option is only valid for non-ArcmailAttach systems.

LinkMsgid

The LinkMsgid keyword instructs Squish to perform reply linking on the basis of MSGID and REPLY keywords. This style of linking ensures that the "read original" and "read reply" fields in the message header always point to the correct message. However, MSGID-based linking requires more memory than the default subject-based reply linking.

LogFile <filespec>

The LogFile keyword instructs Squish to keep a log of message information in the specified file. The log file is Maximus and Binkley-compatible, so it's safe to keep all three logs inside one file.

The Squish log file includes information on archives, packet headers, areas which received mail, the amount of mail that was sent from your system, and to whom. The log file provides one way of obtaining EchoMail statistics; the other way is through analysis of the binary statistics file. For more information on binary statistics, please see the "Statistics" keyword.

LogLevel <number>

The LogLevel keyword instructs Squish to include only a certain subset of messages in the log file. <number> must be a log level number from 0 to 6. (The default log level is 6.)

The types of lines logged for each log level are shown below:

Level	Flags
0	!
1	!+
2	!+*
3	!+*-
4	!+*-#
5	!+*-#:
6	!+*-#: (and a space)

MaxArchive <k>

When used in conjunction with MaxMsgs, this keyword helps to control the maximum size of archive files that Squish generates.

<k> specifies the approximate size, in kilobytes, of the largest ARCmail file that Squish should create.

The MaxArchive value controls Squish's selection of archive files when compressing messages. With MaxArchive set to 'x', Squish will not ADD any packets to archives which exceed a size of 'x' kilobytes. Instead, it will create a new archive for the same node, assuming that all of the *.MO0 to *.MO9 filename extensions have not been used. If all *.??0 through *.??9 have been used, Squish will add packets to the *.??9 archive, regardless of its size, to ensure that packets remain in order.

This keyword does not provide a strict upper limit on the size of archives, but it does help to control archive size. For example, assume that a value of "MaxArchive 100" is used, and assume that a 99k file already exists. This file is less than 100k, so Squish will add more mail to that archive. Depending on your MaxMsgs setting, the mail to be added could range from a few kilobytes to a few megabytes.

However, a relatively low value for MaxMsgs (100-200) allows Squish to control the size of generated archives within 50-75k of the value specified by MaxArchive.

MaxAttach <number> (ArcmailAttach systems only)

The MaxAttach keyword specifies the maximum number of ARCmail attach messages which can be in your NetMail directory at any given time. For internal reasons, Squish needs to know how many attaches you'll have concurrently. The default, 256, is more than enough for most systems. However, if you transfer an extremely large volume of mail with an ArcmailAttach mailer, you may need to increase this to 512 attaches or more.

MaxMsgs <msgs>

The MaxMsgs keyword instructs Squish to create a new set of outbound packets after processing every <msg> messages. The MaxMsgs keyword is useful in NEC situations, especially when a large volume of mail is processed at the same time. This verb helps to limit the size of packets that Squish generates, by making it taking a break from tossing and scanning after sending every <msgs> messages, and to then go and archive the packets generated so far.

This keyword is completely automatic when running in single pass mode; as long as IN, OUT and SQUASH are specified on the command line, Squish will interleave the tossing, scanning and packing automatically. However, when running Squish in multipass mode, some batch file modifications are required.

For multipass mode, after Squish finishes scanning every <msgs> messages, it will stop execution and exit with an errorlevel of 5. This errorlevel should be trapped by your batch file, and SQUISH SQUASH should be called at that point. After packing the NetMail area, your batch file should call SQUISH IN OUT again. For example, the following batch file segment could be used to run Squish in multipass mode:

```
:TossIt
rem * Perform the main toss/scan cycle

SQUISH IN OUT -fECHOTOSS.LOG
if errorlevel 5 goto PackIt
goto DoneToss

:PackIt
```

```
rem * If we got here, it's because we exceeded MaxMsgs.
rem * We use -o, since we don't want to pack the netmail
rem * area right now.

SQUISH SQUASH -o
goto TossIt

:DoneToss
rem * Now, perform a full pack to take care of remaining
rem * archives, and also to handle any in-transit NetMail.
SQUISH SQUASH
```

When using the MaxMsgs keyword, Squish will create a file called MAX_MSGS.DAT in the current directory. This file is used to maintain information that is required between passes, such as the current location in the packet file and information about the packet itself.

MaxPkt <num>

The MaxPkt keyword informs Squish that there may be up to <num> packets queued in the OUTBOUND.SQ directory. For internal reasons, Squish needs to know the maximum number of packets that you'll be keeping in OUTBOUND.SQ. MaxPkt defaults to 256 packets, which should be more than enough for most systems. Unless you run a very busy system, you won't need to use this keyword.

NetFile [NoPkt] [NoArc] [NoEcho] <path>

The NetFile keyword specifies the path to one of your inbound directories. When tossing mail, Squish will look in all of the NetFile directories you specify in the configuration file, and it will attempt to toss from each one.

If the 'NoPkt' modifier is added before the directory name, Squish will never toss plain *.PKT files from that directory.

If the 'NoArc' modifier is added before the directory name, Squish will never toss compressed mail files from that directory.

If the 'NoEcho' modifier is added before the directory name, Squish will never toss EchoMail messages from that directory. NOTE! The NoEcho modifier cannot be used if "BatchUnarc" is enabled. Squish must be able to toss one incoming archive at a time to properly apply NoEcho processing.

These modifiers can be used in conjunction with BinkleyTerm's three-tiered inbound areas to make your system slightly more secure.

Nuke (ArcmailAttach systems only)

The Nuke keyword instructs Squish to delete any orphaned compressed mail files in the OUTBOUND.SQ directory. Before performing a SQUISH SQUASH, Squish will scan the NetMail area for ARCmail attach messages. It will then scan OUTBOUND.SQ and delete any compressed mail file for which there is no attach.

WARNING! This keyword is dangerous, since Squish will delete compressed mail bundles if an ARCmail attach message is accidentally deleted.

Most systems do NOT require this keyword. If your mailer supports the ^aFLAGS kludge for truncating files, this is not necessary.

As of this writing, the only mailers which is known to require the Nuke keyword is D'Bridge. Some users have also reported that the Fido mailer also requires this keyword.

NoSoundex

The NoSoundex keyword instructs Squish to disable the automatic "Soundex" feature for remapping point mail. In certain cases, the Soundex feature may incorrectly map the name of a point, so this keyword can be used to ensure that Squish uses exact name matching when processing remapped mail.

NoStomp

The NoStomp keyword instructs Squish to leave packet headers alone when routing mail with the "Route" keyword in SQUISH.CFG. (Packet stomping is normally required to ensure that the packet is addressed properly, including the packet password and destination address.)

OldArcmailExts

This keyword instructs Squish to use the old file extension format when creating compressed mail archives. Normally, Squish creates files with extensions of .MO?, .TU?, .WE?, and so on. If OldArcmailExts is enabled, Squish will use the .MO? extension only, for compatibility with older systems such as Opus 1.03.

Origin <text>

The Origin keyword specifies a default origin line to use for EchoMail messages. If no origin can be found in the body of a message which was originated locally, Squish will use this text as the default origin line.

If you are using AREAS.BBS, this keyword is not required. The AREAS.BBS format includes a default origin line, and that origin will override the one given in SQUISH.CFG.

Outbound <path> [zone]

This keyword is required for both ArcmailAttach and BinkleyTerm systems.

When running Binkley, this keyword gives the path to the base outbound directory. Specify only the base path of your outbound directory, such as "D:\Bt\Outbound". If you are using a BinkleyTerm system, Squish will add the zoned directory extensions automatically, so only one outbound path needs to be declared for most systems. If a particular outbound directory does not exist, Squish will create it on the fly.

Squish also creates a directory with an extension of .SQ for both BinkleyTerm and ArcmailAttach systems. This is used by Squish as a temporary work directory for packets.

With the optional [zone] parameter, Squish supports BinkleyTerm's domain outbound directories, using zone-based domain resolution.

The first "Outbound" keyword in SQUISH.CFG must not contain a zone number. This keyword will be used as the default outbound directory for your primary domain.

After the first default "Outbound" keyword, any number of domain-specific "Outbound" keywords can follow. The [zone] parameter instructs Squish that mail for the given zone belongs to a different domain and should be placed into a different directory.

For example, given the following definitions:

```
Outbound  \squish\outbound
Outbound  \squish\alternet    7
Outbound  \squish\foonet      123
```

Mail for zone 7 will be placed into \squish\alternet, since the zone number matches the second "Outbound" keyword.

Mail for zone 123 will be placed into \squish\foonet, since the zone number matches the third "Outbound" keyword.

Mail for your primary zone number will be placed into \squish\outbound. The zone number does not match any of the zone-specific outbound directories, so the mail goes into the default directory.

Mail for zone 9 will be automatically placed into \squish\outbound.009. The zone number does not match any of the zone-specific outbound directories, so it goes into a zoned version of the default directory.

Pack <packer> <node> [<nodes>...]

The Pack keyword instructs Squish to use the specified mail compression program when compressing mail for the listed nodes. By default, if a node is not listed after any Pack keywords, Squish will use the compressor defined with the DefaultPacker keyword. If no default packer is declared, Squish will simply use the first packer listed in SQUISH.CFG. Since the FidoNet standard for mail compression is ARC, you are urged not to change the default. Instead, the Pack keyword can be used to change the compression method on a node-by-node basis.

<packer> specifies the name of a packer, as specified in COMPRESS.CFG. Please see the section entitled "ARCHIVERS AND MESSAGE COMPRESSION" for information on COMPRESS.CFG.

<node> and <nodes> specify the nodes for which this packer should be used. Any number of nodes may be listed here, including wildcards. When packing mail for any of the nodes listed after a particular packer, Squish will attempt to run that packer using the information given in COMPRESS.CFG.

You can specify as many nodes as you like, and you can also use as many packers and Pack lines as desired. When unpacking compressed archives, Squish will determine the archive type automatically, again using the information given in COMPRESS.CFG.

WARNING! Squish will not attempt to convert archives from one type to another. Before changing the compression type for a certain node, make sure that no other compressed mail archives are waiting to be sent to that system. If there was, Squish would attempt to add a packet to that archive using the wrong archiver, which would certainly cause problems.

Password <node> <password>

The Password keyword allows passwords to be assigned to individual systems. For more information on using passwords, please see the section entitled "SECURITY".

<node> should specify a single node address. No wildcards are permitted.

<password> is the case-insensitive password to use for the specified node. Passwords must be eight characters or fewer, with no spaces allowed.

PointNet <net>

PointNet specifies the "fakenet" number to use for non-4D points. The fakenet point scheme was designed to work with systems which were unable to support true points, such as BinkleyTerm 2.40 and lower.

When this keyword is used on the bossnode system, it causes the specified net number to be stripped from the SEEN-BYs for messages being exported to non-point systems. The fakenet number will also be used on the bossnode for remapping point addresses.

If used on a point system, this keyword ensures that the right address is used on the point's origin line, if Squish is actually forced to add an origin line to a message.

NOTE! The point software should be responsible for placing the correct origin line in messages generated by that point. Squish will use the correct address if Squish itself inserts the origin, but Squish will never modify an existing origin line.

QuietArc

The QuietArc keyword instructs Squish to suppress the screen output of external compression utilities. This option makes the screen look tidier, but it doesn't make any functional changes.

Remap <node> <name> (non-ArcmailAttach systems only)

The Remap keyword is used to readdress mail with specified addresses in the "To:" field. Unlike other mail processors, Squish's remapping facility is a full node remapper. Messages can be remapped anywhere, from a point across the street to another system across the world.

<node> specifies the network address of the system for which mail is to be remapped. This address must be EXPLICIT; if you are remapping mail to a 4D point, then you must specify the address in a 4D format. If you are remapping mail to a fakenet point, you must specify the node's fakenet address.

<name> should be the name to check for in the "To:" field of inbound messages. When a NetMail message arrives that is destined for your system, Squish will scan the "To:" field for all of the names listed in Remap statements. If it finds a match, it will readdress the mail to the appropriate address and write the message back to disk.

The <name> field has two special features:

* Name wildcards. By placing a "*" at the end of a name, Squish will automatically remap messages which begin with that text. For example, using a remap name of "Jes*" would remap messages which were to "Jesse Hollington", "Jesse", and so forth.

* Soundex support. If Squish can't find an exact match, it will perform a soundex compare of both the "To:" field and the Remap names. If a misspelled match is found, the message will still be readdressed (and noted in the log). To disable Soundex support, see the "NoSoundex" keyword.

Remap statements are processed in sequential order, from top to bottom. Therefore, if you wish to have a "clean up" statement to remap messages which were not caught by other Remap statements, include the command "Remap <node> *" after all other Remap statements. This will cause Squish to remap all remaining messages to the specified address, even if it couldn't find a match using any of the other Remap statements.

Routing <filespec>

This keyword specifies the location of the routing configuration file. By default, Squish will look for ROUTE.CFG in the current directory.

SaveControlInfo (Squish message areas only)

When this keyword is enabled (as it is in the default configuration file), Squish will preserve SEEN-BY and path information in *.SQ?-style message areas. If this keyword is commented out, Squish will strip SEEN-BY information from *.SQ? areas (when running in single pass mode ONLY).

If you are really tight on disk space, stripping the SEEN-BYs is one way to reduce disk requirements. As long as you are running in one-pass mode, the *.SQ? format is safe to use without having SEEN-BYs written to the local message base. However, Squish will always write SEEN-BY information for *.MSG areas, as it will for *.SQ? areas when running in multipass mode.

Secure

The Secure keyword enables Squish's packet security features. For more information, please see the section entitled "SECURITY".

Statistics [filename]

The Statistics keyword turns on a binary statistics file, SQUISH.STT, which is used to hold extremely verbose statistics information about inbound and outbound messages.

The Squish statistics file includes enough information to create a 100% accurate billing report for NECs and other EchoMail hubs. A minimal program (SSTAT) is included to parse this statistics file and display a billing report.

[filename], if specified, overrides the default filename for the statistics file. For example, to instruct Squish to place the statistics file in d:\squish\squish.stt, the following keyword should be used:

```
Statistics d:\squish\squish.stt
```

Note that the "Statistics" keyword will approximately double the memory requirements for reading the SQUISH.CFG and AREAS.BBS files.

StripAttributes [node [node...] [EXCEPT node [node...]]]]

The StripAttributes keyword instructs Squish to strip the crash, hold and direct bits from incoming messages.

This keyword is enabled in the default configuration file, since it prevents other systems from overriding your routing by sending a crash-flavoured message through your system.

However, some systems run "power points" which have file attach privileges, so this keyword may need to be disabled to give full control of the system to the points (and everyone else).

If no [node] parameter is specified, Squish will strip attributes from all incoming mail by default. If nodes are specified, Squish will strip attributes on incoming mail from those nodes only.

In addition, the EXCEPT keyword can be used to exclude certain nodes from StripAttributes processing.

For example, the following line:

```
StripAttributes 1:All 2:All EXCEPT 1:123/456
```

instructs Squish to strip attributes from all messages from zone 1 and zone 2, except those which originate from 1:123/456.

Swap [filespec]   (DOS version only)

The Swap keyword instructs Squish to swap itself out of memory before calling external archiving programs. If you are running Squish in a restricted environment, swapping can save 200K of conventional memory (or more).

By default, Squish will first try to swap itself to XMS and then EMS. If unsuccessful, Squish will try to swap itself to disk. When swapping to disk, Squish will use a default swap filename of __SQUISH.~~~ in the current directory. However, if you wish to specify an alternate path and filename (such as a file on a RAMdisk), you can do so here.

THIS KEYWORD MUST SPECIFY A FILENAME, NOT JUST A PATH!

TinySeenBys <node> [<nodes>...] [EXCEPT <nodes>...]

The TinySeenBys keyword instructs Squish to strip down the SEEN-BYs to a bare minimum when exporting or scanning messages to the specified nodes. The SEEN-BYs in the messages sent to those nodes will contain only the addresses of those systems which are listed for the EchoMail area in question. Any number of nodes may be specified, and wildcards can be used.

The "EXCEPT" keyword can be used to except certain nodes from tiny SEEN-BY processing.

For example, the following line

```
TinySeenbys 1:249/All 1:12/All Except 1:12/12
```

instructs Squish to generate tiny SEEN-BYs for all nodes in nets 249 and 12, with the exception of node 12/12.

TossBadMsgs

Uncommenting the TossBadMsgs keyword indicates that it's safe to toss messages from your BAD_MSGS area. Normally, when Squish finds a message with an unknown area tag, or if it finds an insecure message, Squish will place that message in the BAD_MSGS area. However, if the message later becomes valid (such as when the SysOp adds an EchoArea definition for a previously-unknown area), Squish can automatically toss messages from that area. This feature can save you from manually moving dozens of misdirected messages.

Note that the -u command-line parameter can be used to toggle this mode.

Track <filespec>

The Track keyword specifies a filename to use as a separate log for in-transit NetMail. If this keyword is enabled, Squish will log the To:, From: and Subject: lines of all in-transit netmail, in addition to the messages' origination and destination addresses.

This keyword is useful for determining exactly who is forwarding messages through your system, and where the final destinations of those messages were. In addition, Squish will also place a note in the log if an AREA: line was found in the forwarded message, which indicates that the message in question was a routed EchoMail message.

WARNING! This keyword must point to a separate log file. You cannot use the same log for both Track and LogFile.

ZoneGate <target> <node> [<nodes>...]

The ZoneGate keyword is used to strip the SEEN-BYs on messages destined for other zones. According to the FidoNet Technical Standards Committee, SEEN-BYs should be stripped whenever a message crosses a zone boundary. The procedure of stripping the SEEN-BYs on such a message is called "zonegating".

<target> specifies the node for which you are zonegating mail. All EchoMail messages destined to this node will have their SEEN-BYs stripped.

<node> and <nodes> specify one or more nodes to add to the SEEN-BYs of the zonegated messages, AFTER the original SEEN-BYs have been stripped. These addresses should be two-dimensional (net and node only), since there are no provisions for zone or point numbers in SEEN-BYs.

WARNING! Unlike other mail processors, Squish does not automatically add any addresses to the SEEN-BYs of zonegated messages. As a bare minimum, you should add your own address and the address of the zonegate.

For example, to zonegate messages from 1:123/456 to 2:987/654, the following ZoneGate line might be used:

```
ZoneGate 2:987/654      123/456 987/654
```

Notice that both our address (123/456) and the zonegate's address (987/654) were included in the <nodes> section of the command. Failing to add both of these nodes may cause dupes.

Area Definitions

Squish supports a flexible method for defining message areas in the Squish configuration file. The same mechanism is used for defining EchoMail, NetMail, BAD_MSGS and duplicate message areas. The general format of an area declaration is:

```
     <type> <tag> <path> [flags] [nodes]
```

Area Types

<type> specifies the type of area to define. A type must be specified for all areas. This type should be one of the following keywords:

NetArea	This area is defined as a NetMail area. For BinkleyTerm systems, Squish will pack messages from all NetAreas specified. For ArcmailAttach systems, Squish will use the first NetArea declared for creating ARCmail file attaches.
EchoArea	This area is defined as an EchoMail area. Messages can be tossed to and scanned from this area using the standard Squish commands.
BadArea	This area is defined as a "bad messages" (or BAD_MSGS) area. This will be used to store messages with security violations, messages for unknown areas, and other messages for which Squish couldn't find a proper home.
DupeArea	This area is defined as a duplicate message area. If Squish receives duplicate messages from another system (and if KillDupes is NOT enabled), those messages will be placed into this area for review by the SysOp.

Area Tags and Paths

<tag> specifies a short, one-word area tag (name) to use for the area being defined. For EchoMail areas, this tag should be the name of the echo. For other area types, such as NetAreas and BadAreas, any unique tag can be used.

<path> specifies the directory and/or filename to use for this message area. If using the *.MSG format, the name of a separate directory should be given for each area. If using the Squish format (*.SQ?), a path and root filename (up to eight characters

with no extension) should be given. Squish will automatically create areas which do not exist.

WARNING! The <tag> and <path> fields in AREAS.BBS are "reversed", meaning that the path comes before the tag. If you are used to defining areas in AREAS.BBS, make sure that you use the proper order when defining areas in SQUISH.CFG (tag and then path).

Flags

[flags] specifies an optional set of flags and options. A flag consists of a single dash followed by a letter, plus an optional modifier. The flags currently supported by Squish are:

-f

This flag tells Squish that the area is stored in FTS-0001 (*.MSG) format. *.MSG is the default storage format, so this flag usually is not required. When the *.MSG format is being used, the name of the *.MSG directory should be given for <path>.

-h

This flag tells Squish to ensure that all incoming messages have the PRIVATE bit enabled. This is useful for NetMail areas or other area types where confidential information may be exchanged.

-p<node>

This flag specifies an alternate primary address for the current area. <node> should be a full 4D address, including zone and point numbers (if appropriate).

Using an alternate primary address will affect Squish's operation in the following ways:

* The alternate address will be added to the SEEN-BYs for the specified area, as opposed to adding the first address declared in SQUISH.CFG.

* The alternate address will be added to the ^aPATH line instead of your normal address.

* The alternate address will be used when creating packets with messages from the specified area.

	This flag permits "clean" multi-zone operation with only one configuration file, even when using a diverse range of addresses and message areas.
-s	This flag strips the private bit from all messages which are tossed to this area. Unlike other EchoMail processors, handling of private messages can be controlled on an area-by-area basis.
-u<node>	This flag instructs Squish to accept broadcast modify/delete messages from the node in question. See the section entitled "BROADCAST MODIFY/DELETE" for more information.
-x<node>	This flag instructs Squish NOT to accept any inbound messages in this echo from the specified address. This flag can be used to make a certain node "read only" for one area, since messages coming from that node will trigger a security violation and be placed in BAD_MSGS.
-0	(That's a zero, not the letter "o".) This flag tells Squish that the current area is a "passthru" area. This means that messages from this area don't stay on your system; they are simply scanned out to the other systems listed for this area and then deleted. If you are running Squish in single pass mode, messages will be scanned directly from the inbound *.PKT files and will never touch your message areas.
-+<node>	This flag adds a node to the SEEN-BYs for the current area only. <node> should be a two-dimensional address (net/node only). Note! If you are using an alternate primary address, that address will be added to the SEEN-BYs automatically.
-$	This flag tells Squish that the area is stored in the Squish (*.SQ?) format. When using this flag, <path> should specify the full path and root filename of the Squish area. (A root filename is the path and the first eight characters of a DOS filename, with no extensions permitted.)
-$d<num>	This flag specifies that no more than <num> days

worth of messages should be kept in a Squish-style area. This causes SQPACK to delete all messages which are more than <num> days old. NOTE! Unlike -$m, this flag does NOT cause Squish to purge messages on the fly. If you choose to delete message by date, you must run SQPACK once a day. Otherwise, if you choose to delete by message number, you can allow Squish to renumber and purge the area as it tosses.

-$m<msgs>	This flag specifies the maximum number of messages to keep in a Squish-style area. Normally, Squish leaves the maximum message counter alone in a *.SQ? area. However, if Squish has to create the area for one reason or another, the maximum message counter will be set to a value of <msgs> messages. Using the -$m switch also implicitly activates the -$ switch.
-$s<msgs>	This flag specifies the number of messages to skip killing <msgs> messages at the beginning of a Squish-style area. This flag must be used in conjunction with the -$m flag. See the section entitled "USING SQUISH-FORMAT MESSAGE AREAS" for more details.

Split Area Definitions

Squish allows EchoMail areas to be defined in SQUISH.CFG, AREAS.BBS, and it even allows area definitions to be split between both. Such splitting is necessary, especially if you want to use Squish-specific features and also wish to remain compatible with the AREAS.BBS file format.

When declaring a split area in either SQUISH.CFG or AREAS.BBS, Squish will check to see if that area exists in the list of existing areas. If it does, Squish will simply add to that area's definition, as opposed to creating a new logical area. Nodes and flags specified in one file will be automatically applied to the other; in other words, the following two definitions:

AREAS.BBS:

```
     $C:\Msg\Magic     XXYZY          1:123/456 789
```

SQUISH.CFG:

```
     EchoArea XXYZY   C:\Msg\Magic -$ -p1:234/567
```

would be equivalent to the following definition in SQUISH.CFG only:

```
     EchoArea XXYZY  C:\Msg\Magic -$ -p1:234/567 1:123/456 789
```

Declaring areas using this method enables all of the new Squish features, but it also allows for compatibility with utilities which use AREAS.BBS for node information, including Areafix and many others.

ARCHIVERS AND MESSAGE COMPRESSION

Through the use of external file archiving utilities, Squish is capable of compressing mail that it sends to other systems. If you are using the Send and Route keywords in ROUTE.CFG, Squish will automatically compress mail for the specified nodes. Instead of sending *.PKT files, Squish will then send *.MO?, *.TU?, *.WE?, *.TH?, *.FR?, *.SA? and *.SU? files. (The last part of the file extension will be a single digit; this digit is incremented after creating each archive, which ensures that compressed mail files do not get overwritten once an archive arrives at its destination.)

In addition, Squish allows you to define a separate compression program for individual nodes. Some of the systems you connect with may wish to use ARC, while others may elect to use ZIP and LZH. The Pack and DefaultPacker statements in SQUISH.CFG can be used to select different archivers for each node.

However, Squish takes a new approach to message compression. Instead of hardcoding support for all of the archivers currently available, Squish uses a "compression configuration file". Using such a file allows Squish to support as many archive types as possible. The flexibility of this file even allows Squish to use archiving programs which have not been invented at the time of this writing.

In SQUISH.CFG, the Compress keyword controls the location of the compression configuration file. As distributed, COMPRESS.CFG comes configured for use with ARC, ZIP, PAK, ZOO, ARJ, LHarc 1.13 and LHarc 2.xx. However, new programs can be added to Squish's repertoire on a moment's notice.

COMPRESS.CFG is divided into a number of entries, one per archiver. (As many archivers can be defined as you like, memory limiting.) Each entry in COMPRESS.CFG looks like this:

```
     Archiver <name>
          Extension  <ext>
          Ident      <pos>,<hexstr>
          Add        <cmd>
          Extract    <cmd>
          View       <cmd>
     End Archiver
```

The "Archiver" keyword starts off the definition of an archiving program. <name> specifies the short-form name of the archiver; this short form should be a single word, since Squish uses it to select compression types with the Pack statement in SQUISH.CFG.

<ext> is the file extension normally used by the specified archiver. At present, this information is not used by Squish, but it may be used by other programs which share the same COMPRESS.CFG file.

Next comes the Ident keyword: the information provided by this keyword allows Squish to identify archives of unknown types. Most archiving programs place a special signature at the beginning of a compressed file, so when Squish encounters such a file, these special signatures are used to determine how to unpack the archive.

The <pos> number tells Squish where to look for the archiver's signature. If <pos> is greater than or equal to zero, it is interpreted as an offset from the start of the file, with the first byte being offset 0, the second byte being offset 1, and so on. If <pos> is negative, Squish will interpret it as an offset from the END of the file, with -2 being the last byte in the file, -3 being the second-last byte in the file, and so on.

<hexstr> is an ASCII representation of the archiver's signature. Every byte in the archiver's signature should be converted to hexadecimal, and then entered into the compression configuration entered as two hexadecimal digits. Since most archivers use special control characters in their signatures, entering the signatures directly is not possible. However, by representing each byte as two hexadecimal digits in the range 0-9 and A-F, even the simplest of editors can read the compression configuration file.

The Add keyword specifies the command which should be used to add files to the specified type of archive. Similarly, the Extract and View keywords specify the commands to use for extracting from and viewing archives (respectively).

Before executing the archiver, Squish will make two special translations: all occurrences of the string "%a" will be changed to the name of the archive being processed. Similarly, all occurrences of the string "%f" will be changed to the name of the file(s) to be added or extracted. These tokens allow the utmost in flexibility when handling multiple archiving programs.

The "End Archiver" keyword signifies the end of an archiver definition. Each "Archiver" keyword must be followed by an "End Archiver" keyword later in the file.

In addition, if Squish encounters a line prefixed with the "DOS" or the "OS2" keywords, it will only parse that line when running under the indicated operating system. This allows different archiving programs to be used under DOS and OS/2.

ROUTING

Squish includes a complete routing control system for both BinkleyTerm and ArcmailAttach mailers. Squish is capable of generating ARCmail attach messages for systems with dynamic routing, and it also includes a full complement of commands for handling static routing.

All of Squish's routing is performed via the ROUTE.CFG file. Whether you are using dynamic (ArcmailAttach) or static (Binkley) packing, all of the Squish routing commands must be placed in this file.

As distributed, all of the commands in your routing control file are executed every time a SQUISH SQUASH is performed. However, Squish's routing system can be used to implement "schedules", which cause certain routing commands to be performed at certain times. (Schedules can be run on the basis of the time of day, or they can be simply run on demand.)

Commands which are to be always run, regardless of schedules, must be placed in the "global section", or before the first "Sched" statement in ROUTE.CFG. The only difference between schedules and the global section is when the commands are performed; otherwise, after Squish has started processing a section of the control file (whether it be the global section or a schedule), all routing commands are treated the same.

Generic Routing Commands

Several routing commands apply to both static and dynamic packing. Namely, all of following commands can be used equally well on both BinkleyTerm and ArcmailAttach systems:

Send <flavour> [NoArc] <node> [<nodes>...]

> The Send command scans for uncompressed mail with a normal flavour, compresses it, and changes the mail's flavour. The Send command changes the attributes of the mail, but it never changes the mail's destination address. (If you wish to change the destination address, please see "Route", below.)
>
> <flavour> specifies the flavour to use for the resulting compressed mail file. Squish will assign this flavour to the compressed mail archive after archiving all of the packets for this node.

The optional NoArc modifier stops Squish from archiving the packet that is created for the specified system. Squish will simply change all normal-flavoured packets to the specified flavour, placing all of the original packets into one large packet file. The command "Send <flavour> NoArc <node>" has the same net effect as "Change Normal <flavour> <node>". This modifier apples to BinkleyTerm systems only.

<node> and <nodes> are simply a list of nodes to which mail should be sent. Squish will expand all node number wildcards and process all normal-flavoured packets for those nodes.

In the BinkleyTerm outbound area, using Send (without the NoArc modifier) has the following result:

```
xxxxyyyy.OUT   -->  xxxxyyyy.MO? + xxxxyyyy.?LO
```

With the NoArc modifier, the result will be as follows:

```
xxxxyyyy.OUT   -->  xxxxyyyy.?UT
```

Route <flavour> [NoArc] [File] <target> [<nodes>...]

The Route command scans for uncompressed mail with a normal flavour, compresses it, changes the mail's flavour, and (unlike Send) readdresses the mail. The Route command scans for mail addressed to any of the specified nodes, but readdresses that mail to the target. In other words, Route scans for uncompressed mail destined to <nodes>, archives the mail, and finally sends it to <target>.

<flavour> specifies the flavour to use for the resulting compressed file.

The NoArc modifier (optional) stops Squish from archiving the resulting mail. Squish will simply combine all of the specified packets into one, give it the appropriate flavour, and send it to the target without any form of compression. This modifier applies to BinkleyTerm systems only. WARNING! The NoArc modifier causes all 4D zone and point information to be lost! Only net and node numbers will be maintained for mail routed with the NoArc keyword.

The File modifier (optional) instructs Squish to route file attaches in addition to messages. By default, all file attaches are sent to the specified destination, regardless of flavour. However, if the File modifier is used, Squish will also scan for and route file attaches. This modifier applies to BinkleyTerm systems only.

<target> is the address of the system which will receive the routed packets. Make sure that you have made prior arrangements with the target to route your mail, since it's discourteous to route mail through someone else's system without asking permission. Wildcards should not be used when specifying the target address.

<nodes> are the addresses for which mail will be routed. All of the normal-flavoured mail for these addresses will be packaged up and sent to the target.

In the BinkleyTerm outbound area, using Route (without any modifiers) has the following result:

```
     xxxxyyyy.OUT   -->  XXXXYYYY.MO? + XXXXYYYY.?LO
```

When using the NoArc modifier, the Route command has the following result:

```
     xxxxyyyy.OUT   -->  XXXXYYYY.?UT
```

The File modifier causes the following changes to take place, in addition to the above:

```
     xxxxyyyy.FLO   -->  XXXXYYYY.?LO
```

Define <token> <text>

The Define token is used to provide a short-form for routing commands. The Define command acts as a macro facility, since it causes all further occurrences of <token> to be replaced by <text>. As an example, the Define command is useful if there is a common set of nodes for which different routing commands must be applied over and over again. Given the following Define statement:

```
     Define Hub300  300 301 302 303 304 305 306
```

Squish would translate the word "Hub300" to "300 301 302 303 304 305 306" wherever that word occurred later in the file. If the following route command were given:

```
     Route Crash 1:123/300 Hub300
```

all of the mail for nodes 300 through 306 would be routed through 1:123/300.

However, replacements will only take place when the defined <token> is surrounded by whitespace or punctuation.

Dos <cmd>

The Dos command can be used to pass the name of an external command to the operating system. The command will simply be run via the default command processor (normally COMMAND.COM for DOS, or CMD.EXE for OS/2).

Schedules

Squish supports the concept of "schedules", which are routing commands that can be run at different times of the day. Squish also supports a section of global routing commands which are always run.

The "Sched" statement in ROUTE.CFG is used to begin a schedule. All routing commands which precede the first Sched statement are in the "global section", and they are always run when a SQUISH SQUASH is performed. Squish will execute statements inside a schedule up to the next Sched statement, or until the end of the file is reached.

The format of the Sched statement is as follows:

```
     Sched <tag> [day] [start] [end]
```

<tag> is a one-word identifier for this schedule. If you wish to explicitly execute this schedule on demand, then this tag must be specified on the command line through the -s command-line switch. Otherwise, the name given to each schedule is irrelevant.

<day> instructs Squish to execute the schedule on certain days of the week. To always execute a schedule, use the word "All". To execute a schedule on weekends only, use the word "WkEnd". To have it executed only on weekdays, use "WkDay". To execute the schedule on a particular day of the week, use the words "Mon", "Tue", "Wed", "Thu", "Fri", "Sat" or "Sun". To execute the schedule on more than one day of the week, use the "|" (pipe) character to string together two or more of the above words. For example, "Tue|Wed|Fri" would cause the schedule to be run only on Tuesdays, Wednesdays and Fridays.

<start> specifies the starting time for the schedule, specified in 24-hour format (hours:minutes). The schedule will only be executed if the current time is equal to or greater than the time specified.

<end> specifies the ending time for the schedule, specified in 24-hour format. The schedule will only be executed if the current time is equal to or less than the time specified.

WARNING FOR EX-QMAIL USERS!

IF YOU WILL BE RUNNING A SCHEDULE BY THE COMMAND LINE, <tag> IS THE ONLY REQUIRED PARAMETER! When Squish is scanning through ROUTE.CFG, it will automatically execute any schedules which fall within the specified time frame, regardless of those schedules' tags. This means that ALL schedules with a date/time of "All 00:00 23:59" will be executed. Unless you want to run all of your schedules each time a SQUISH SQUASH is performed, only use the date and time for schedules which actually need to be run at different times.

Example

For example, the following ROUTE.CFG segment has a set of global routing commands which are run all the time, a set of commands which are run in the morning, and a set which are run in the afternoon/evening:

```
; GLOBAL COMMANDS
; Always route mail for net 123.

Route Crash  1:123/1  123/All

; Route mail for my NC to the right place.

Route Crash  1:222/0 1

; MORNING COMMANDS

Sched Morning All 00:00 11:59

     ; Convert previously-held mail to crash for nodes in zone 2

     Change Hold Crash 2:All
     Send Crash 2:All

; AFTERNOON/EVENING COMMANDS

Sched Afternoon All 12:00 23:59

     ; Convert crash zone 2 mail back to hold

     Change Crash Hold 2:All

     ; Now send all new zone 2 mail as hold

     Send Hold 2:All

; COMMANDS RUN ON DEMAND
```

```
Sched PollNEC

     Poll Crash 1:222/2
```

When run in the morning, Squish would handle routing for nets 123 and 222, in addition to sending crashmail to all zone 2 systems. When run in the afternoon, Squish would still handle routing for nets 123 and 222, but it would send mail to zone 2 systems as hold. When run with the command line switch "-sPollNEC", Squish would generate a crash poll to node 222/2.

BinkleyTerm Routing Commands

The following commands can only be used on a BinkleyTerm-style system. These commands directly modify the outbound directories, so there are no equivalent commands for ArcmailAttach systems.

Leave <node> [<nodes>...]

The Leave command modifies outbound mail so that it won't be sent by BinkleyTerm. Squish will rename both *.?UT and *.?LO to an extension which is not recognized by BinkleyTerm, thereby preventing the mail from being sent or picked up.

Leaving mail is useful for mail hubs and NECs, since it stops all mail from being given to the specified nodes. If your system has a crucial polling window and you want to stop others from receiving mail when they call in, the Leave command can be used to make that mail inaccessible.

To convert mail back to its original form after using "Leave", see the documentation for the Unleave command.

As many nodes can be specified for the Leave command as necessary. Wildcards are permitted.

In the BinkleyTerm outbound area, using Leave has the following result:

```
xxxxyyyy.?UT   -->  xxxxyyyy.N?T
xxxxyyyy.?LO   -->  xxxxyyyy.N?O
```

Unleave <node> [<nodes>...]

The Unleave command undoes all of the changes which were made by the Leave command. Mail which was left for the specified nodes will be converted back to its original flavour, such that the mail can once again be sent or picked up.

As many nodes can be specified for the Unleave command as necessary. Wildcards are permitted.

In the BinkleyTerm outbound area, using Unleave has the following result:

```
xxxxyyyy.N?T   -->  xxxxyyyy.?UT

xxxxyyyy.?LO   -->  xxxxyyyy.N?O
```

Change <from_flavour> <to_flavour> <node> [<nodes>...]

The Change command is used to change the flavour of existing packets and file attaches.

<from_flavour> specifies the flavour to convert FROM. Only mail of this flavour will be converted. Valid flavours are crash, hold, direct and normal.

<to_flavour> specifies the flavour to convert TO. All types of mail with a <from_flavour> will be converted to the <to_flavour>. Valid flavours are crash, hold, direct and normal.

<node> and <nodes> specify the nodes for which mail should be converted. Squish will process mail which are addressed to these nodes only, and apply the specified conversions.

In the BinkleyTerm outbound area, using Change has the following result:

```
xxxxyyyy.?UT   -->  xxxxyyyy.?UT
xxxxyyyy.?LO   -->  xxxxyyyy.?LO
```

Poll <flavour> <node> [<nodes>...]

The Poll command is used to generate a poll to another system, even if there is no other mail waiting for that node.

<flavour> specifies the flavour to use when creating the poll.   Valid flavours are normal, crash, hold, and direct.

<node> and <nodes> specify the node numbers which should be polled. Wildcards are NOT permitted.

In the BinkleyTerm outbound area, using Poll has the following result:

```
Create: xxxxyyyy.?LO
```

HostRoute <flavour> [File] [NoArc] <node> [<nodes>...]

HostRoute is similar to the Route and Send commands, except that HostRoute will route normal-flavoured mail to each node's net host, as opposed to routing it all through a central hub. (In other words, mail for 123/456 will be routed through 123/0, just as mail for 678/901 will be routed through 678/0.)

<flavour> specifies the flavour to use when creating the routed packets.

The File modifier instructs Squish to route files in addition to messages. In general, the HostRouting of files is discouraged.

The NoArc modifier stops Squish from archiving the resulting packets. Messages will be sent to the net hosts in an uncompressed form.

<node> and <nodes> specify the nodes for which routing will take place. Wildcards are permitted.

In the BinkleyTerm outbound area, using HostRoute has the following result:

```
     xxxxyyyy.OUT   -->  xxxx0000.?UT
```

The File modifier causes the following changes to take place, in addition to the above:

```
     xxxxyyyy.FLO   -->  xxxx0000.?LO
```

Dynamic Routing (FrontDoor)

Squish has only minimal support for dynamic routing, since dynamic routing is always performed by your mailer. The Route command CAN be used to route mail, but using your mailer to perform all routing is recommended.

All schedules for an ArcmailAttach mailer should end with the command "Send Normal World". This command compresses all remaining mail and gives it a flavour of normal. Since ArcmailAttach mailers only recognize compressed mail, this command is required to ensure that all mail is properly sent.

SECURITY

In most environments, EchoMail security is not a problem. However, in some instances, other systems may attempt to inject unwanted mail into normal EchoMail areas. Squish provides two forms of protection against this:

* "Secure mode". When the "Secure" keyword is added to SQUISH.CFG, Squish will check the origination addresses of all incoming packets, and it will only toss messages from systems which are listed in the "<nodes>" section of your area definitions. In other words, unless you have added the address 123/456 to the definition for a particular echo, that node will not be able to send messages in that echo to your system.

  Secure mode provides protection against unlisted systems "crashing" messages into an EchoMail area through your system. If messages cannot be tossed due to a security violation, they will be placed in your BAD_MSGS area.

* Packet passwords. Although secure mode protects against most unwanted packets, other systems may try to send you mail using the address of another node. Squish protects against this by allowing packet passwords to be used; by using Password statements in SQUISH.CFG, you can declare a password for nodes that you regularly connect with.

  Passwords are "bidirectional", meaning that the same password is used for both sending and receiving mail. The packet password must be the same for both sides of the link. In other words, if node 123/456 wants to use the password "qwerty" for node 234/567, the configuration file on 123/456 must look like this:

  ```
  Password 234/567 Qwerty
  ```

  and the configuration file on 234/567 must look like this:

  ```
  Password 123/456 Qwerty
  ```

  In either case, the intent is to list the password to use when sending mail to the other system. When creating packets for that system, Squish will insert the specified password into the packet. In addition, when tossing packets from that system, Squish will check to make sure that the password is the same as the one listed in your configuration file. If the two passwords do not match, the incident will be noted in the log, and the packet will be renamed to *.BAD and will not be tossed.

* Multiple NetFile paths. Squish supports multiple NetFile paths with varying attributes in SQUISH.CFG. When used with a mailer which allows different inbound directories to be declared for passworded and non-passworded systems, the modifiers for the NetFile keyword (such as NoArc and NoPkt) can be used to stop Squish from tossing certain types of mail from certain systems. This helps prevent so-called "ARCmail bombs" from being decompressed and filling up all of your disk space.

* Receive-only nodes. The -x<node> flag can be used with an EchoArea definition to stop Squish from tossing messages from a certain node in a certain echo. This allows an echo to be sent to a system, but it stops that system from sending any messages back into the echo.

If all of these security features are enabled, Squish becomes a very secure and rigid mail processor. Most of these features won't be necessary for day-to-day operation, but they will become extremely useful when trying to stop unwanted mail from entering your system.

MULTIZONE OPERATION

Squish was designed especially for use on a multizone system. In addition to full zone and point support, Squish can use an alternate primary addresses for each individual message area on your system. Squish can control the SEEN-BYs on an area-by-area basis, generate packets with proper 4D addressing information, and more.

The first key to multizone operation is the -p flag in SQUISH.CFG. This flag allows an alternate primary address to be selected for each individual message area. Squish will use the alternate address when adding to the ^aPATH line, SEEN-BYs and the message origin. In addition, the origination address in outbound packets will also use your alternate address for that area.

To use the -p flag, simply place that flag and your alternate primary address before all the addresses of the systems that you feed. For example, to declare an alternate primary address of 89:487/106, in an area called "IMEX.R82" (which you receive from 89:487/0), the following line should be used in SQUISH.CFG:

```
     EchoArea  IMEX.R82  C:\Msg\ImexR82 -p89:487/106 89:487/0
```

The alternate primary address also changes your default address for the area definition, so the following would have the same effect:

```
     EchoArea  IMEX.R82  C:\Msg\ImexR82 -p89:487/106 0
```

Squish can support an unlimited number of alternate primary addresses, as long as you use the proper -p flag for each area. The -p flag is not required for areas which use your primary address (the first address declared in SQUISH.CFG), since Squish will use your first address by default.

Squish is also capable of adding node numbers to the SEEN-BYs for one area only. In a manner similar to using alternate primary addresses, simply add a -+<node> flag for each area and for each node that you wish to add. (Your alternate primary address will be added automatically, so you don't need to worry about adding it as well.)

For example, if you wanted to add the nodes 487/2 and 487/3 to messages passing through your system in the ABC echo, using an alternate address of 89:487/106, the following declaration in SQUISH.CFG would do the job:

```
     EchoArea ABC C:\Msg\ABC -p89:487/106 -+487/2 -+3 487/0
```

Finally, you should make sure that you have declared all of your alternate primary addresses using the Address keyword in SQUISH.CFG. Otherwise, Squish won't realize that messages for your alternate address are really for your system, and it will try to forward them as in-transit netmail.

That's all there is to it! No secondary configuration files, and more importantly, no kludges are required. Squish provides transparent support for multizone systems, with a minimum amount of hassle and with no speed drawbacks.

POINTS (4D, FAKENETS AND BOSSNODES)

Squish incorporates full support for 4D and fakenet points, both as a point itself and as the managing bossnode. Bossnodes can also transparently support both types of points, still using the same configuration file.

Fakenet Points

In the early days of FidoNet, support for true points was minimal or nonexistent. As a workaround, instead of giving true 4D addresses to a system's points, a dummy net number was created. In turn, the node numbers in that net would actually represent points of the bossnode. Since this dummy net/node combination was a simple net/node address (with no point numbers), adapting existing software was no problem. This net number would be used for internal communication between the bossnode and the points. For example, if 123/456 were using a dummy net number of 14122, the fakenet version of the address 123/456.1 would be 14122/1.

When using the fakenet scheme, it is vitally important to ensure that fakenet numbers do not "escape" out into the rest of the net. Since many systems could theoretically be using the same dummy network number, the fakenet addresses are usually stripped or converted by the boss system before messages are sent to the rest of the network.

Squish supports fakenet points through the use of the "PointNet" keyword in SQUISH.CFG. Simply specify which dummy net you are using, and Squish will automatically handle the details of stripping and adding SEEN-BYs.

NOTE! As the boss of a fakenet system, you should make sure to use the actual fakenet number whenever you refer to one of your points. Since Squish supports both 4D and fakenet points in the same configuration, you must be careful to use the correct address. In other words, if you have a fakenet point at 123/456.1, with a fakenet address of 14122/1, you must ensure that your control files and AREAS.BBS always refer to this point as 14122/1. 4D addresses can only be used when dealing with true 4D points.

4D Points

Squish supports true 4D points in addition to fakenet points. However, to use 4D points, both the bossnode and the point must be running 4D-capable tossers, scanners and mailers. As of this writing, the only 4D mailers in common use are FrontDoor, InterMail, D'Bridge, and BinkleyTerm 2.50+. 4D tossers and scanners include Squish, TosScan, Imail and others. Unless all of your software supports 4D addressing and the "2+" packet header proposal, you won't be able to use 4D points.

Squish supports 4D points in an extremely simple manner; just use the 4D point address like a normal address. Squish will automatically handle the tossing and scanning of messages from 4D points, and there is no need to remap addresses. SEEN-BYs are ignored when sending messages to 4D points, since SEEN-BY lines are only two-dimensional.

As with fakenet points, make sure that you always use the 4D address when referring to 4D points. If you attempt to mix and match 4D and fakenet addresses for the same node, Squish may not work the way you intended it to.

IF YOU USE *.MSG FORMAT MESSAGE AREAS AND WISH TO SUPPORT 4D POINTS, SINGLE-PASS MODE MUST ALWAYS BE USED. *.MSG areas cannot hold the origination and destination point numbers that Squish requires for 4D operation. However, Squish can use 4D points with *.MSG areas, as long as "SQUISH IN OUT" is always used.

Squish message areas contain proper zone and point operation, so 4D points can be used with both the single-pass and multipass modes.

Remapping

Squish includes a built-in node remapper. This remapper readdresses inbound netmail based on the "To:" field, and it also remaps outbound netmail by changing the fakenet back to a 4D address, if applicable.

When specifying point numbers in the remapper, make sure that you specify either a 4D or a fakenet address, as appropriate. Again, since Squish supports both 4D and fakenet points in the same configuration, you must make sure to use ONLY fakenet addresses for fakenet points, and ONLY 4D addresses for 4D points.

For more information on the remapper, please see the "Remap" keyword in the SQUISH.CFG reference.

USING SQUISH-FORMAT MESSAGE AREAS

In addition to the FidoNet standard *.MSG format, Squish also supports the proprietary, flat-file *.SQ? message base. This message base was designed from the ground up to be fast, reliable and small. Some of the key features of the Squish message format are:

* Flat-file message areas. A separate message database per message area. Using a separate message file for each area still provides for quick access, but if disaster should happen (such as a message base crash), only one area will be damaged.

* Maintainability. The Squish format utilizes a customized circular file structure. For the most part, Squish message areas are completely self-maintaining. If a message is deleted in a Squish message base, the space left by that message can be reused when more messages are written to the area.

  Squish attempts to fill the message base in an optimal manner, so "packing" is not as important as it is with other message base types. Depending on the volume of mail you process, Squish areas may only need to be packed once a week! Squish areas are also renumbered on the fly, so an external renumbering utility is not required.

  Finally, the Squish file format also allows for a "maximum message limit" to be set for any given area. This limit will be used to automatically purge old messages as new ones are written to the message base, which eliminates the need for an external message deletion utility. A separate set of message numbers is maintained for each message area, which eliminates the "messages in this area are numbered 89,216 through 90,784" eyesore of other message formats.

* Reliability. The Squish format was designed to make message recovery an easy task, even in the event of a total message base crash. The SQFIX utility can be used to fix a damaged Squish area with no user intervention. SQFIX has a high success rate; in most cases, not a single message will be lost!

* Speed. Since Squish uses a flat-file message base, it is much faster than the FTS-0001 *.MSG format. Based on preliminary testing, Squish is also faster at tossing messages than QECHO 2.66 and several other utilities which use the Hudson message format.

* Size. Messages are stored in the equivalent of a random-access file with a record length of 1 byte, so no "blocks" are necessary. Unlike the Hudson format, Squish stores messages directly after one another with no padding. When the SEEN-BY information is turned off, Squish message bases are consistently smaller than the equivalent Hudson-format message base.

* Multitasking and LAN awareness. Squish was designed from the ground up to be compatible with multitasking and network systems. Multiple programs, such as Squish and Maximus, can access the same message base at the same time with no danger of corruption.

* Flexible structure. Squish areas have built-in support for full 4D origination and destination addresses, date-of-creation and date-of-arrival binary timestamps, and more. In addition, ^A kludge lines are stored in a separate logical record before the message body, providing a unified way for developers to access message control information.

Even with all of these advanced features, the Squish format is easy to use. Squish format areas can be declared in both SQUISH.CFG and AREAS.BBS, although the maximum message limit can only be set in SQUISH.CFG for compatibility reasons.

On disk, Squish message bases are stored using two files per message area. To name a Squish area, you must specify a path and a "root filename". A root filename is the base part of a filename, eight characters or fewer, with no extension. Squish will then add the appropriate extensions to access the message data and index files.

The Squish message format itself only requires two files per message area:

areaname.SQD	This file is used to store the message data. Information about the area, the number of messages, message headers, the message body, and the linked lists are all stored in this file.
areaname.SQI	This file is used to store the message index. This contains a copy of the "To:" field in the message header, and it allows the Squish format to be accessed quickly in a non-linear fashion.

The above are the only required files for a Squish-format area.

However, other programs create files with similar extensions, such as:

areaname.SQB	This file is used by the Squish tosser to store duplicate message information.
areaname.SQL	This file is used by Maximus to store lastread pointer information. This file is an array of UMSGID numbers, using the user's lastread_ptr field as the index.
areaname.SQO	This file is used by the Squish tosser to provide a custom origin line for the given area. If an .SQO file is present for a given message area, the first line in the file will be used as the origin line for messages in that area, overriding the "Origin" setting in SQUISH.CFG. Squish will automatically add the "* Origin" prefix and the trailing "(zone:net/node)" text.

For more information on the Squish file format, see the Squish Developer's Kit, to be released shortly after Squish 1.1 is made available.

To declare a Squish-format area in AREAS.BBS, simply add a "$" character to the beginning of the path. In other words, to convert the following to a Squish-format area:

```
     E:\MSG\ASDF    ASDF 249/99
```

simply add a "$" at the beginning to make it look like this:

```
     $E:\MSG\ASDF   ASDF 249/99
```

For an area declared in SQUISH.CFG, simply add a "-$" flag to the area definition. In other words, given the following definition for a *.MSG area:

```
     EchoArea  ASDF E:\MSG\ASDF    249/99
```

a "-$" should be inserted to convert the area to the Squish format, like this:

```
     EchoArea  ASDF E:\MSG\ASDF -$ 249/99
```

To add a maximum message limit to a Squish area, the "-$m" flag can be used in SQUISH.CFG. For example, to limit the ASDF area to 100 messages, the following definition could be used:

```
     EchoArea  ASDF E:\MSG\ASDF -$m100  249/99
```

When using the above definition, Squish will automatically purge messages such that no more than 100 messages are in the area at once time.

A limit on the maximum age of messages can also be set using the -$d flag. However, purging messages by date is only performed when SQPACK is run, so Squish won't perform date deletion when tossing messages. For more information on the -$, -$d and -$m flags, please see the SQUISH.CFG reference.

In terms of usage, Squish will automatically create nonexistent areas, so you don't have to worry about creating them manually. Squish treats *.SQ? areas just like *.MSG areas in all respects, including direct tossing and scanning. However, Squish areas can be accessed much faster than their *.MSG counterparts, and Squish areas will also use less disk space.

Once you have tossed messages to a Squish-format message base, those messages can be accessed with any program which supports the Squish format.

If you are using the -$m flag, Squish areas are mostly self-maintaining. However, small "holes" can creep into the message base over time, even with the circular file format. For this reason, the SQPACK program can be run at predefined intervals to pack and compress message bases.

Also, if you are using the -$d option, SQPACK must be run to delete old, expired message. For information on installing SQPACK, please continue reading in the next section, entitled "SQUISH-FORMAT MESSAGE UTILITIES".

BROADCAST MODIFY/DELETE

Squish 1.10 introduces support for broadcast message deletion and modification. This feature allows users on remote systems to write an EchoMail message, and even after the message has been sent to a remote system, it allows the user to amend or delete the message on all of the systems which carry that conference.

First, broadcast messages are currently only supported in Squish message areas. While the concept of broadcast messages could be easily applied to other area types, Squish needs to maintain a database of MSGID values for broadcast message processing. In the interests of speed, Squish currently does not maintain this information for *.MSG areas.

Note that the broadcast feature also relies on the Squish .SQB file to store information on messages being tossed. The dupe file must also be large enough (as defined by the "Duplicates" keyword) to store information for all of the messages in the area.

THE BROADCAST FEATURE ONLY WORKS ON MESSAGES THAT WERE ORIGINALLY TOSSED TO A SQUISH AREA BY SQUISH 1.10 OR ABOVE.

The broadcast feature must be enabled on an area-by-area basis, using the "-u<node>" flag for each EchoArea definition.

<node> specifies the address of a trusted uplink which is allowed to submit update/delete requests. (The actual origin of the update/delete message is not important. However, Squish will only process update/delete messages if it receives them from the node specified by -u.)

For example, the following definition sends PVT_ECHO to both 249/106 and 107, but it only accepts update/delete messages that it receives from 249/106. Note that 249/106 is specified twice; it is listed once in the scan list, and it is listed once for the -u flag.

```
     EchoArea  PVT_ECHO  \path\pvt_echo -$   1:249/106 107 -u106
```

An automatic update/delete message has the same format as a normal message in an EchoMail area, except that it contains an "Area Control Update" kludge in the following format:

```
     ^aACUPDATE: MODIFY <addr> <serial>
          or
     ^aACUPDATE: DELETE <addr> <serial>
```

For either format of the command, Squish will scan the area for a message containing a MSGID kludge of the format "MSGID: <addr> <serial>".

When processing an ACUPDATE MODIFY, if the message containing that MSGID is found, it will be replaced in its entirety by the update message. The old message will retain its original message number. Note that Squish will strip the "ACUPDATE" line from the modified message before writing it to disk.

When processing an ACUPDATE DELETE, if the message containing that MSGID is found, that message will be deleted. The contents of the ACUPDATE message are ignored.

Note that all ACUPDATE processing is performed during "SQUISH OUT" phase. If an ACUPDATE message passes the "-u" security check, the ACUPDATE message (in its original form) will be scanned out to all of the nodes before the ACUPDATE operation takes place.

In other words, ACUPDATE acts as a broadcast message to all nodes listed for an area. The modification/deletion is performed by each node that receives the message; the messages which are consequently modified are NOT transmitted to other nodes.

All remote modify/delete transactions are noted in the Squish log file.

SQUISH-FORMAT MESSAGE UTILITIES

In addition to the main tosser/scanner/packer, the Squish package includes several utility programs designed to help users of Squish-format message bases.

SQPACK: Weekly maintenance

The SQPACK program is used to "pack" Squish-format message areas. Over the course of time, small holes can develop in Squish message areas, so SQPACK can be used to recover this wasted space. If you are using the -$m flags on your areas, you probably won't need to run SQPACK more than once a week. However, if you are killing messages by age (through the -$d flag), you'll need to run SQPACK whenever you wish to delete messages.

The command-line format for SQPACK is as follows:

```
SQPACK <filespec>
```

<filespec> should specify the name and path of a Squish-format message data file. Wildcards are allowed. For example, to pack the message in the file D:\MSG\MUFFIN.SQD, the following command should be issued:

```
SQPACK D:\MSG\MUFFIN.SQD
```

In addition, if there are a number of message areas in the D:\MSG directory, the following command could be given to pack all of those areas:

```
SQPACK D:\MSG\*.SQD
```

When SQPACK finishes processing all of the selected areas, it will print out a short statistics report detailing how much space was used before, how much space is used now, and a percentage savings. This percentage can guide you when figuring out how often to run SQPACK.

Note to Maximus users

In addition to the name of a Squish data file, SQPACK can also accept the name of a Max 2.00 AREA.DAT file. If you enter "SQPACK D:\MAX\AREA.DAT", SQPACK will automatically pack all of the Squish-format areas defined in AREA.DAT.

SQCONV: Conversion between *.MSG and *.SQ?

The SQCONV utility is used to convert message areas between the *.MSG and Squish message formats. SQCONV is normally run from the OS prompt, and it can only be run on one area at a time.

SQCONV uses the following command-line format:

```
     SQCONV <from_path> <from_type> <to_path> <to_type> <zone>
```

<from_path> specifies the directory or root filename of the area that you wish to convert.

<from_type> specifies the type of the <from_path> area. Valid types are either "*.MSG" or "Squish".

<to_path> specifies the directory or root filename for the area to be created.

<to_type> specifies the type of the <to_path> area. Valid types are either "*.MSG" or "Squish".

<zone> specifies your default zone number. Since *.MSG format messages don't always have zone numbers available, you must tell SQCONV which default zone number is used in that area.

For example, to convert a *.MSG area in C:\MAX\MSG\LOCAL to a Squish area called C:\MAX\MSG\PRIVATE.SQ?, the following command line would be used:

```
     SQCONV C:\Max\Msg\Local *.MSG C:\Max\Msg\Private Squish 1
```

The above assumes that you are in zone 1; for other zones, simply substitute your default zone number for the "1".

To convert an area back from Squish format to *.MSG, the following could be used:

```
     SQCONV C:\Msg\Sqarea Squish C:\Msg\Msg_Area *.MSG 1
```

Also, please note that both Squish and *.MSG areas can have the same name. Since *.MSG messages are placed inside a separate directory, and Squish areas are placed in *.SQ? files in the directory above, the following command is perfectly acceptable:

```
SQCONV C:\Msg\Local *.MSG C:\Msg\Local Squish 1
```

The above command would convert all of the messages in C:\Msg\Local\*.MSG and place those messages in the files called C:\Msg\Local.SQ?.

WARNING! You must exercise caution when converting a Squish area back to the *.MSG format. Since SEEN-BYs are not updated in Squish EchoMail messages, converting a Squish area to *.MSG may cause messages in that area to be rescanned.

You should make sure that the high water marker is properly set when converting a Squish area back to *.MSG. (Converting *.MSG areas to Squish is not a problem, since Squish will never scan a converted Squish-format message.)

SQSET: Control for message deletion

The SQSET utility is used to manually update the "maximum message limit" and "protected messages" for a given message area. These limits can be set automatically in SQUISH.CFG using the -$m and -$s flags in SQUISH.CFG, but some users may prefer to set these limits from the OS prompt.

The command-line format of SQSET is as follows:

```
     SQSET <area> [<max_msgs> [<skip_msgs>] [<days_to_keep>]]
```

<area> specifies the path and root filename of a Squish-format message area.

<max_msgs> specifies the maximum number of messages to keep in this area at one time. The next time a message is written to this area, Squish will automatically delete and renumber messages such that there are no more than <max_msgs> in the area. If this parameter is omitted, Squish will simply display the current settings for <max_msgs> and <skip_msgs>.

<skip_msgs> specifies the number of messages to skip at the beginning of the area, before starting to automatically delete messages. When <skip_msgs> is used, <max_msgs> must also be specified. When Squish is deleting old messages, this command causes it to only start deleting after the first <skip_msgs> messages. This is useful for keeping "posting rules" as the first message in each area, or for keeping a certain number of messages for an indefinite length of time.

<days_to_keep> specifies the maximum age of messages in this area. Any messages which are more than <days_to_keep> days old will be deleted during the next SQPACK run.

SQINFO: Diagnostics

SQINFO is a diagnostics and information utility for Squish-format message areas. SQINFO will walk through both the chain of message frames and also the chain of free frames, displaying information about each message and reporting any errors it comes across. SQINFO can be used to quickly diagnose problems in a Squish-format message area.

This program is not required for normal use, so it can be deleted if disk space is at a premium. SQINFO is primarily of interest for third-party utility authors who are writing *.SQ?-compatible programs.

The command line format of SQINFO is as follows:

```
     SQINFO <area> [-q] [-b] [-e]
```

<area> is the path and root filename of a Squish-format message area.

-q is the optional "quick" switch. Instead of displaying a verbose report on each message, SQINFO will simply list the location of each message. SQINFO will still check for and notify the user of errors, since this option simply disables most screen output.

-b is the optional "bug-find" switch. When in this mode, SQINFO will display nothing except the area name. SQINFO will still check for errors, but it won't tell you where the error occurred. This mode is useful when checking a large number of areas for problems; simply run "SQINFO <area> -b" over all message areas, and then rerun SQINFO (without the -b) on areas which have problems.

-e is the optional "errorlevel" switch. When in this mode, SQINFO will operate as in "bugfind" mode, but it is also geared for batch operation. SQINFO will not prompt the user to press a key, and it will return an errorlevel based on the condition of the base. If the message area is okay, SQINFO returns an errorlevel of 0. If the message area is damaged, SQINFO will return an errorlevel of 1.

The -q, -b and -e options are mutually exclusive. Only one of the above switches can be specified on the command line.

SQREIDX: Repair (minor)

SQREIDX is an indexing utility for Squish-format message areas. This utility can be used to correct simple indexing problems, but it is not required for normal use. If more than just the index is damaged (as reported by SQINFO), SQFIX should be used instead.

The command-line format of SQREIDX is:

```
     SQREIDX <area>
```

<area> should be the path and root filename of the Squish-format message area with the grunged index.

SQREIDX will recreate the index for <area> by walking through the message chains in the data file, and then by rewriting the original index. If the index is not the only problem with the area, the full-fledged SQFIX should be used instead.

SQFIX: Repair (major)

SQFIX is a full-fledged restoration utility for Squish-format message areas. Even when run on a heavily-damaged area, SQFIX can recover all of the messages which were not individually damaged. SQFIX is completely automated and requires no operator assistance. The command-line format for SQFIX is:

```
     SQFIX <area>
```

<area> is the path and root filename of the message area to be fixed.

Squish will automatically restore a damaged fixed area. If the area can be fixed, the old data files will be renamed to <areaname>.XXD and <areaname>.XXI.

SSTAT: Statistics generation

If the Statistics option is enabled in the Squish configuration file, Squish will create a binary statistics log. This log contains a great deal of information, but not in a human-readable format. The SSTAT utility included with Squish is very minimal, but it does provide useful information.

SSTAT is more of a billing report generator than an analysis utility; unless you are scanning mail to someone else, SSTAT won't produce any output. However, if your system does scan mail to more than one system, SSTAT is capable of producing a 100% accurate billing report for each system, based on the volume of mail sent to each node that you feed.

SSTAT works from a configuration file called SSTAT.CFG. This file must be in the current directory when SSTAT is run, as must SQUISH.STT (the statistics log produced by Squish).

SSTAT.CFG is a free-format configuration file, similar in nature to SQUISH.CFG and ROUTE.CFG. At the current time, SSTAT only handles the following two keywords:

Track <node> [<nodes>...]

> This keyword causes SSTAT to track mail for the specified nodes. <node> can be a full-fledged address, including zone, net, node and point number. SSTAT will track all mail which is addressed to that node. Wildcards may NOT be used.

Area <tag> [<tag>...]

> This keyword causes SSTAT to track mail for the specified area tags only. SSTAT will only report mail for the specified areas, and these areas will be the only ones included as part of the billing reports.

A sample SSTAT.CFG is included in the distribution archive. After you have customized SSTAT.CFG, let Squish run for a while, and then run SSTAT. If all goes well, and if you are scanning mail to at least one other node, SSTAT should produce a highly-detailed report describing all of the message activity on your system.

SSTAT performs two sets of calculations: one set is based on the number of messages sent to each node, while the other is based on the quantity of bytes sent to each node. In most cases, the number representing the quantity of bytes sent is usually more reflective of long-distance telephone charges.

At the end of the report, SSTAT will include an overall summary with percentage totals, suitable for use in a cost-sharing arrangement. SSTAT uses a sophisticated algorithm for determining the TRUE cost of messages, but in short, the billing scheme works like this:

1) For each area listed in SSTAT.CFG, SSTAT will determine the total quantity of mail (whether that be bytes or messages) that you received in the area. Squish will divide this number by the total quantity of mail received in ALL areas listed in SSTAT.CFG. The result of the division is a percentage for the area; this percentage represents the area as a portion of your total inbound mail. In other words, if you multiply your long-distance charges by this percentage, the result will represent how much it cost to bring in the specified area.

2) For each area listed in SSTAT.CFG, and for each node listed within each area, SSTAT will calculate the total quantity of mail sent to that node for that area. SSTAT will then divide this by the total quantity of mail sent to all nodes within that area. The result of this division represents the percentage of mail (within that area) which is being consumed by the node in question.

3) For each node in each area, the percentage determined in step 2) is then multiplied by the percentage determined in step 1) for that area. This figure now represents the actual percentage that this node must pay for receiving the mail in that area.

4) The percentages for each node are then added up for each area, which represents the percentage total as shown at the end of the billing report. This percentage represents the actual amount that this node should pay for any long-distance charges in a cost-sharing system.

APPENDIX A - Errorlevels

Squish will set one of several errorlevels after termination. The errorlevels currently supported by Squish are:

Erl	Action
0	No tossing or scanning took place.
1	Error. Squish encountered some sort of fatal error and had to abort.
2	Sent EchoMail. This means that Squish exported one or more EchoMail messages. (This errorlevel is only used when performing "SQUISH OUT" as part of a multipass operation.)
3	Tossed NetMail only. This means that Squish tossed one or more packets, but only NetMail was received.
4	Tossed EchoMail and/or NetMail. This means that Squish tossed one or more packets containing EchoMail (and possibly NetMail).
5	MaxMsgs was reached. This means that Squish reached the MaxMsgs limit when processing mail in a multipass environment. This means that SQUISH SQUASH should be invoked, followed by another round of exporting.

With the exception of errorlevel 1, higher errorlevel numbers will take precedence. In other words, if EchoMail and/or NetMail was tossed, but MaxMsgs was also reached, Squish will exit with an errorlevel of 5.

APPENDIX B - Problem Reporting

If you discover a problem in the Squish software package, you are encouraged to report the problem to the author. Problem reports can be placed in the TUB or MUFFIN echomail areas (for Squish and Maximus, respectively). If the problem is urgent, you can also send a NetMail message to the author at 1:249/106.

If you encounter a "trap" error with the 32-bit DOS version (or any of the OS/2 versions), information from the register dump should be included when submitting a problem report.

Under OS/2 1.x, the screen will be presented right away and will start with a "TRAP 000D".

Under OS/2 2.0, you will get a "A program in this session encountered an error" pop-up screen. Cursor down to the "register dump" option and write down the information in the register dump window.

Under SQ386, the extender will display "Trap 000d" and the registers will be simply dumped to the console.

In all cases, the most important registers to write down are CS and IP (or CS and EIP). If possible, the register values for AX, BX, CX, DX, SI and DI are also helpful. (The 32-bit equivalents are EAX, EBX, ECX, EDX, ESI and EDI.)

In addition, you should include any other files needed to recreate the problem, such as SQUISH.CFG, AREAS.BBS, ROUTE.CFG, and any relevant packets or message bases.

GLOSSARY

*.MSG

The message format originally used by Fido, also used as the FidoNet standard for local message storage The *.MSG system requires a separate directory for each message area, and a separate file for each message. This makes the *.MSG format inefficient, in terms of both disk space and time. For compatibility reasons, Squish uses the *.MSG format by default.

*.SQ?

The message format originally used by Maximus. *.SQ? (or "Squish format") uses two files per area; a .SQI file contains a message index, and the other contains the message headers and text.

*.PKT

The "transport layer" for FidoNet-compatible messages. Packets are used when transferring messages between two different FidoNet systems. Since all systems use the same type of packet, *.PKT can be used to transfer messages between systems which use unlike message bases (such as *.MSG and the QuickBBS/Hudson format). See also "2+" and "StoneAge".

2+

A new, backwards-compatible form of *.PKT files. The original packet design had no allowances for zone and point information; the 2+ packet format corrects this shortcoming. Squish creates 2+ packets by default, but it can also handle StoneAge packets. See also "*.PKT" and "StoneAge".

4D

A term used to refer to a full FidoNet address. An address in the form "zone:net/node.point" is called 4D because it allows for four-dimensional addressing.

archiver

An archiver is a program used to compress files. Archivers are very useful in a FidoNet environment, since compressing mail can reduce its size by up to 80%.

ARCmail

ARCmail refers to both a program from System Enhancement Associates and to a mail compression format. Mail which is compressed using the ARC archiver is referred to as "ARCmail". Similarly, mail compressed with ZIP is called "ZIPmail", mail compressed with LHarc is referred to as "LZHmail", and so on.

ArcmailAttach

An ArcmailAttach system is a mailer which requires "file attaches" to send compress mail bundles. ArcmailAttach is not specific to the ARCmail compression method; it simply means that a different method is used for creating compressed mail bundles. Mailers such as FrontDoor, InterMail, D'Bridge and Dutchie require the "ArcmailAttach" keyword in SQUISH.CFG.

AREAS.BBS

A ConfMail-compatible file containing a list of message directories, addresses, and area tags. Squish can use AREAS.BBS, but areas must be declared in SQUISH.CFG to use some of Squish's advanced features.

busy flag

A semaphore file used in the BinkleyTerm outbound area. Busy flags are used to ensure that two programs don't access the same file at the same time, when running in a multitasking or a network environment.

crash

A message flavour. Crash means that a message should be sent directly to its destination, with no routing implied. Crash usually implies "send it NOW".

direct

A message flavour. Direct is identical to crash in all respects, except that the message will be governed by your mailer's event schedule.

downlink

A downlink is a system that receives mail from your system. For example, if 1:1/1 sends mail to 1:1/2 and 1:1/3:

1:1/2 and 1:1/3 are downlinks of 1:1/1.
1:1/1 is the feed for 1:1/2 and 1:1/3.

duplicate messages (dupes)

A second copy of an EchoMail message. When problems crop up in EchoMail topology, copies of old messages occasionally get dumped into the system. Squish usually detects and stops most duplicate messages.

EchoMail

A message conferencing system originally devised by Jeff Rush. Squish fully supports the EchoMail format.

errorlevel

An errorlevel is a number set by a DOS or OS/2 program when that program terminates. This number can be later checked for in a batch or command file, and various actions can be taken based on that number.

FD

An acronym for the FrontDoor front-end mailer.

feed

A "feed" is the system which sends you mail for a particular EchoMail area.

flavour

A 'flavour' (or the American "flavor") is also known as a priority. Flavours can be used to override other routing commands and to explicitly send mail directly to a given node. For more information, see the glossary entries for "hold", "normal", "crash" and "direct". Also see the

"Leave", "Unleave", "Send" and "Route" commands in the ROUTE.CFG reference.

front end

A synonym for "mailer".

FTS-0001

A document describing the base level of FidoNet compatibility. "FTS" is an acronym for "FidoNet Technical Standard". Squish is compliant with FTS-0001.

hold

A message flavour indicating that the message in question should be placed on hold for pick-up.

host-routed

Host-routed means that the messages in question will be sent to the network host (net/0), as opposed to being sent directly to the destination. Squish can optionally perform host routing.

leaf node

A system which only communicates with one other system when sending and receiving EchoMail. Leaf nodes do not forward EchoMail to other systems.

mailer

A FidoNet-compatible program which answers the phone and interacts with other systems, which includes transferring files, messages, and system information. Common mailers include BinkleyTerm, FrontDoor and D'Bridge.

maximum message limit

In Squish-format message areas, a limit can be set on the maximum number of messages to allow in a given area. Once this limit is exceeded, messages will be purged from the beginning of the message area until the message count falls below the maximum.

net
network

As part of a 4D network address, a "net" is a small geographical area, usually encompassing a large city and the surrounding area.

NetMail

NetMail is a direct, point-to-point transfer of private messages. NetMail is analogous to "Email".

node

As part of a 4D network address, a "node" is a single system or computer within a net.

normal

A message priority. Normal-flavoured messages can be routed, but if no routing is applied, a normal message will be sent directly to its destination.

origin line

A control line near the bottom of an EchoMail message. The origin line identifies the origination point of a message. Most origin lines have the following form:

```
* Origin: name (address)
```

where "name" is a brief description of the system, and "address" is a full 4D network address.

outbound areas (outbound directories)

A set of directories used for storing outbound mail. BinkleyTerm and Opus are the only common mailers which use outbound areas.

point

As part of a 4D network address, a "point" normally represents a single-user system that connects with a full-fledged network node to receive mail.

remap

Remapping is the process of readdressing inbound messages based on the name in the "To:" field. For example, messages are commonly remapped for points, since the point number may be occasionally omitted when specifying a system address.

SEEN-BY

A control line at the bottom of an EchoMail message. SEEN-BYs are used to determine which systems have already been sent a particular message.

SHARE.EXE

A DOS program used to enable file locking. SHARE must be loaded if you wish to use Squish-format message areas in a multitasking environment. SHARE is only required for DOS systems.

StoneAge

A term applied to the original *.PKT design. StoneAge packets do not support zone or point information. See also "2+" and "*.PKT".

tear line

A control line at the bottom of an EchoMail message. A tear line is used to end the message body, and it usually contains a short, product-specific banner. A tear line begins with three dashes, such as "--- Squish v1.10".

wildcards

Squish uses wildcards to specify multiple nodes for routing commands. For more information, please see the section entitled "Wildcards".

zone

In a 4D network address, a "zone" is a wide geographical area, usually covering one continent or more.

zonegate

A zonegate is a system which sends EchoMail to more than one zone. Squish is capable of acting as a zonegate.

INDEX