TICK v2.10

WHAT IS IT?

Tick is a program which does for files what echomail does for messages. It was largely inspired by the program "Flea", by Ron Bemis. Tick picks up on that concept, and adds to it.

Using any ASCII editor, you set up a configuration file to tell TICK about your system and to set up file echo areas. TICK then looks in your inbound area for received TIC attaches, tosses them to the echo-area directory you specify, appends the FILES.BBS in that area, and optionally echos the files to other systems you specify. If your BBS doesn't use a FILES.BBS, you can have TICK create a user-customized file-list in the format you would like (so long as it's ASCII). You can set up different areas for different file-echos, much as you may have many echotags in echomail. In each file-echo area, you may list up to 100 nodes to which you echo the files. The program establishes passwords which are used to verify that the files you receive are from the node you expect them to be from. You may also specify which nodes in a file echo are allowed to send you files, which may only receive files from you but not send them to you, or which nodes may send you files but never receive them from you.

TICK will allow you to configure so that it creates FLO file attaches for mailers such as Binkley and Opus, which use them, or MSG attaches for Seadog-type systems that use that kind of attach. This second method is referred to as "FIDO" mode within TICK. When TICK creates an attach, it sends both the desired file, and a small control file which contains seenby and other information. The preferred information file is the TIC file, which is defined in FSC-0028. TICK will also generate the FLE file format for communication with FLEA. The choice of which file format to use may be made for each echoed system, independently of other systems echoed. You may set TICK to receive both TIC's and FLE's, or have it process only TIC's received. The choice of which file format you send to other nodes is independent of which file format you received with the original file.

Files are entered into an echo using the companion program HATCH. HATCH uses the same configuration file as TICK.

If you are upgrading, please read this entire document before running this version of TICK, as there are many changes you need to be aware of.

WHAT DOES IT DO

When TICK operates, it looks for inbound files with the extension TIC (or FLE). These are "control" files which tell the program what the name of the "real" file is, the

echo area it is to go to, and what systems have already seen the file. The information is checked against the configuration file, and if passwords match and the area exists, the file is tossed to the "destination directory" established when the AREA was set up. (The file is moved to the destination directory by renaming it if it is possible, or by copying and deleting the original if it is not possible). The FILES.BBS in that directory is appended with the description (again, part of the TIC file). If there are other nodes listed for that echo, TICK will then create new TICs or FLEs for them, and will create FLO files or MSG attaches to those nodes in the outbound. The attaches will send the new TIC and the "real" file to the other nodes. This does NOT happen if that node is already listed in the seenby line of the original TIC.

TICK should be called in your batch file after receiving files. It is designed so that you may redirect its output to the Binkley.log, Opus.log, or to a log of its own. In the simplest form, the command would look something like this:

TICK >> Binkley.Log

or

TICK >> Tick.Log

If running from a shell (not suggested), it would be preferable to use a separate log to avoid problems of writing to a file which may already be open.

THE CONFIGURATION FILE

Before TICK or HATCH can run, you will need to create a configuration file to tell the program about your system, select optional parameters, and to establish file-echo AREAS. Each AREA (you may have many) is equivalent to a different echo area in echomail, and lists the nodes that you receive from and send to. The addresses of the nodes are zone aware.

The brackets indicate optional items, and should NOT be entered in the real configuration file.

The format of the configuration file follows:

IN c:\file\inbound (This specifies the inbound area)

ZONE 1 c:\opus\outbound (This specifies the Outbound area)
(The FIRST Zone is the DEFAULT Zone)

ZONE 2 c:\opus\outbound.002

ZONE 3 c:\opus\out3 (Zones need not follow Binkley style)

```
NET 266                 (Your Net)

NODE 12                 (Your Node)

HOLD c:\holddir\        (Where outbound TICs and FLEs are
                        kept)

QDIR c:\qdir            (MUST be defined - This is the
                        directory used as a holding area
                        for pre-release files.)

[ListFmt %3:-13 %1]     (Alters the default format of the
                        FILES.BBS)

[ListName Files.Bbs]    (Alters default name / location of
                        FILES.BBS)


[AKA 1:1/313]           (Adds your AKA addresses to the
[AKA 5:678/90]           Seenby lines)


[STOPDUP [c:\tickdir]]  (Optional parameter, turns on
                        Stopdup feature - Specifies where
                        DUP files are to be are to be
                        kept)

[QUIET]                 (Optional - Stops beeping on fatal
                         error)

AREA c:\file\ticktest TICKTEST
    Local ListName c:\files\RBBS.LST
    1:266/1 Passwrd1 [FLAGS]   (Where flags = [*][&][F]
    2:512/26 Pass2p                    [C|H|P|T][An])

[TEMP c:\ramdisk]       (Optional directory for temporary
                        files)

[FIDO]                  (Send files as MSG attaches instead
                        of FLO attaches)

[MAIL c:\netmail]       (Location of Netmail - Required if
                        FIDO specified)

[FLEA]                  (If present, tells the program to
```

also process inbound FLE files)

[LOGPATH] (If present, log the PATH lines to the logfile)

[LOGSEEN] (If present, log SEENBY lines to the logfile)

[CRC] (Enable CRC testing)

[LogCRC] (Place copy of CRC in the log)

[Crc2Dup] (Place copy of CRC in the DUP file, Necessary for DupByCRC)

[DupByCRC] (Dupe file by CRC. See [LOCAL DupByCRC] below for more information.)

[NoWait] (Prevent HATCH's from hanging a batch file on errors or ommissions)

[Local] (Specifies the next argument is to be processed for only this area)

[MailerKills] (Defines how TICK will generate FLO attach files, with respect to the deletion of the TIC files. See below for more information.)

[RAID] (Allows creation of .RAD file for certain utilities such as RAID to generate announcments of locally hatched files.)

[REPLACE] (Allows the replacing of files, optionally on an area-by-area basis. See below for more infomation.)

[FDLog] (Forces TICK to log all activities in FrontDoors logfile format.)

[LOCAL BigDesc] (Allows large file descriptions See below for more information.)

[ForceINTL] (An INTL Kludge will be inserted for a MSG attach even if it's the same zone)

The brackets indicate optional items, and should NOT be entered in the real configuration file.

Now ... an item by item description:

IN c:\netfile

This entry should point to the inbound files area. Directory entries in the TIC configuration file may be entered with or without trailing backslashes, and must reference an existing directory.

ZONE 1 c:\opus\outbound

This specifies the Outbound area for zone 1. TIC allows multiple zones in multiple outbound areas (Binkley style). The ZONE line may be repeated for as many zones as you are communicating with. THE FIRST ZONE STATEMENT MUST REFERENCE YOUR OWN ZONE, and is used as the default zone when a control file (TIC or FLE) does not contain a specified zone. The control file must have at least one ZONE line. Your MAIN (DEFAULT) address consists of this zone, and your NET and NODE statements.

ZONE 2 c:\opus\outbound.002

This tells the system where to place FLO files for zone 2. Note that a directory must be declared for each zone you plan to address, even if you run FIDO mode and don't create FLO files.

ZONE 3 c:\opus\out3

The directory must be specified for each zone. Tick does NOT asssume that zones other than your own are named the way they are done in Binkley.

NET 266

This line must contain your primary net number, and is required.

NODE 12

This line must contain your primary node number, and is also required. (In a future release, I'll change this so you can just specify your address as NET/NODE on the same line).

HOLD c:\holddir\

This specifies the location of the "HOLDing" directory. This is where the outbound TIC and FLE control files are stored until your mailer sends them. It also will be used by future TICK versions to hold other information as well. This directory is maintained by TICK, and should not contain other files. Know what you are doing before changing anything in this directory.

QDIR c:\qdir

This directory should be a separate directory and should contain nothing. It is used to hold files that are pending pre-release.

[ListFmt %3:-13 %1]

This optional entry allows you to alter the default format of the "Files.BBS" file that TICK has always created. The format may be compatible with TBBS, RBBS, Maximus, and nearly any other BBS that uses an ASCII file as a files list. Additional descriptions are below. The numbers shown in this example are the default parameters used if you do NOT declare a ListFmt.

[ListName Files.Bbs]

Just as TICK may give you a different format for the FILES.BBS, it is not restricted to the name "FILES.BBS". It is possible to have it called by any legal dos filename. You may locate the file in any directory, rather than the same directory that the files go to, and you may specify that ALL descriptions go to the SAME file if that is what you require. (See more below).

[AKA 1:1/313]
[AKA 5:678/90]

These entries direct TICK to add these nodes to the Seenby list. It is now possible to establish a link with chosen nodes using the AKA address instead of your main address. (See more below.)

[STOPDUP c:\tickdir]

This optional line, if present, activates a function similar to the STOPDUP program I had written to help limit duplicate files from being passed by Flea. What it does is to keep a list of all filenames processed in each echo area. Whenever a new file is received in an echo area, the filename is compared to all names in the list. If that name has previously been seen, the incoming TIC or FLE file has its extension renamed to .BAD, and the received file is ignored. If you later decide that the file should be passed anyway, you may rename the BAD file back to TIC or FLE, and delete the filename from the appropriate ECHONAME.DUP file. The next time TICK is run, the file will be passed. As implied above, when STOPDUP is specified, TICK keeps a file in the specified (or default, if none is specified) directory for each echo area you have set up. If the echotag is "NODELIST", then the file name is "NODELIST.DUP". This behavior is modified by the "LOCAL STOPDUP xxx" line (below) and when you use "DupByCRC".

[QUIET]

If this is not present, TICK will beep should it need to abort with a fatal error. Non-fatal errors will not cause a beep in any case.

AREA c:\file\ticktest TICKTEST
Local ListName c:\files\RBBS.LST
1:266/1 Password [*][&][F][T]([H][An] or [C]) 2:512/26 SECRET

This structure is used to define the echo areas. It is analogous to the AREAS.BBS used by confmail. For each echo, you use the keyword AREA, followed by the directory, followed by the echotag. The following lines are in the format shown above, consisting of ZONE:NET/NODE PASSWORD FLAGs. There may be up to 100 such addresses lines present in each AREA. An AREA ends at the first line NOT in the ADDRESS PASSWORD format, such as a blank line. The exception is that lines beginning with the keyword "LOCAL" do not end an area if they immediately follow the AREA line. These LOCAL lines are used to establish special treatment specific to that area only. Please note that AREA NAMES ARE LIMITED TO 8 CHARACTERS, and must be valid characters for DOS filenames.

The password is the password used for that particular node, and may be up to 8 characters. It is case insensitive. The other node must specify the same password for your node in his TIC.CFG file. The "*", if present, specifies that you will accept files coming from that node. If not present, you may send files to that node, but will not accept them from him/her.

The "&", if present, tells TICK and HATCH to ignore that node when sending files. Files are never sent to a node which has the "&" flag. If combined with the "*" flag, that node becomes a "receive-only" node for that area. You will accept files coming from the node, but will NOT echo files TO it. If the "&" flag is present but the "*" flag is NOT present, the node may neither send nor receive from you, and is effectively commented out without ending the area.

This is probably the least understood area of TICK. Here is an example:

```
AREA C:\DIST\ANSI\ANSI_ADM ANSI_ADM
      1:116/36       PASSWORD       H
      1:274/13       SECRET        &*
      1:365/12       UNKNOWN         &*
      1:395/3       OTHERONE        &*
      1:2604/101      BAR           &*
      1:3629/269      FOO           &
```

This allows 1:274/13, 1:365/12, 1:395/3 and 1:2604/101 to hatch files into the ANSI_ADM file area. TICK will only create ONE outgoing TIC, to 1:116/36. 1:3629/269 is "commented out", thus not allowing that node to send or receive files in this area.

It is possible to tell TICK to generate CLO (crash) and HLO (hold) files directly, or to generate FLO (normal) files as the default. The way this is done is to append a "C" for crash or a "H" for hold as the last token in the system you are connecting to. The "C" and "H" are mutually exclusive. If you attempt to use them both in the same line, TICK will issue a warning, and will treat the system as a HOLD. If you are running in the FIDO mode, the MSG produced should have the Crash or Hold bits set as appropriate. In the Non-Fido mode, FLOs, CLOs, or HLOs are produced. The Address and password are separated by whitespace, as are the password and flag fields. The FTCH*& flags are in any order, but are contained in a single "word". They must NOT be separated from each other by spaces.

The "F", if present, may be upper or lower case (as may be the C, T or H), and specifies that you will send Flea compatible files (FLE extension) to that node (instead of sending TIC files). The format of the received file is irrelevant.

The "T", if present, will send the destination node the file that is being distributed, but not the .TIC. This should only be used if the receiving node is not intending on passing the file on to another node via TICK, and has no desire to have TICK automatically update their FILES.BBS.

The "An" flag takes the form of the letter "A", followed immediately by a single hex number ranging from "0" to "F". This is used to designate that an AKA address is to be used in the link to the node declared on this line. Further description is below.

The AREA statement, as mentioned, may have up to 100 nodes listed for each echo. You may repeat the AREA statement for as many echos as you carry. The statement is considered to end at the first line which is not in ZONE:NET/NODE PASSWORD form. (The LOCAL lines are a partial exception to this rule). DO NOT TRY TO "COMMENT OUT" (with a ";") A LINE IN AN AREA! It will cause the area to END at that line, which is probably not what you had in mind.

[FLEA]

This statement tells the program that it is also to process any inbound FLE files as well as TIC files. The default is to process only TIC format. (Again, you may SEND either format regardless).

[TEMP] c:\ramdisk

This allows you to specify where TICK will write its temporary files. Choosing a ramdisk here will speed up the processing. If this is not included in your CFG file, TICK will use the default directory for your temporary files.

[FIDO]

What this will do is to have TICK create MSG attaches rather than FLO files. (See WARNING below!) This will force TICK to handle attaches to messages rather than creating FLO files. The TIC's and FLE's are still placed in the outbound. The code will try to put both file attaches in the same message if there is room. If the combined length of the paths and filenames exceed the 72 byte field allowed, two messages will be created instead. The created messages will have the killsent flag set, so that your mailer may kill the message once it is no longer needed. I am assuming that whatever software you are running will respect the "killsent" flag and delete the MSG file which "points to" the TIC of FLE in the outbound. What TICK does is to find all MSG files which are file attaches, local, private, have the killsent flag set, and are from "TICK". It takes the subject lines from MSG's meeting these criteria, and writes them to a temp file. It then looks at all TIC's and FLE's in the outbound(s). Any which do not have an active MSG attach are considered "orphans" and are deleted.

WARNING: DO NOT RUN TICK IN THE FIDO MODE IF YOU HAVE ANY TIC'S OR FLE'S IN THE OUTBOUND WHICH ARE "POINTED TO" BY FLO FILES.

TICK will find no MSG attach, assume that the TIC/FLE's are orphans, and delete them!

[MAIL c:\netmail]

This entry is REQUIRED if you are running in the FIDO mode. It allows TICK to find your netmail directory for MSG's. If you do NOT use the FIDO mode, this entry is not needed.

[LOGPATH]

If present, path information (present in the TIC format files) will be sent to your log file for future reference. This is useful in determining topology. Also, since the time and date that each system processed the file is included in the PATH, this allows you to see how much delay was encountered on each leg of the trip.

If present, all seenby lines in the received TICs or FLEs are also logged to the LOG file. This results in very large logs, and is only intended for debugging and finding problems.

[CRC]

Tick generates a CRC-32 on all files that it processes. If you include CRC in your CFG file, the CRC in the inbound TIC file will be compared to the one calculated. If they don't match, TICK will not process the file. The most common reason for the CRC-32 to not match is the system that sent the file added an archive comment to the file before it was sent. If you add archive comments, ensure it has been sent to all the systems it was destined for BEFORE adding the comment. The other common reason is a grunged file was received during the file transfer.

[LogCRC]

If this is in your CFG, the CRC-32 will be logged to the logfile.

[Crc2Dup]

This will cause the CRC-32 to be stored in the DUP file. An example of why you would want to use this keyword is if you had an area that received a file on an updated basis, and it was always the same filename. You would want to only abort the distribution if the file has not been seen before.

[NoWait]

This prevents HATCH from looping back for new input when you enter an incorrect FILEname or AREAname.

[MailerKills]

Tick users may now choose between the "#" flag and the "^" flag in FLO files, to select the method of killing the TIC files after they are sent. The default is to set the "#" flag, which instructs the mailer to truncate the file after sending. TICK will delete it on the following run. If you place the keyword "MailerKills" in the CFG file, TICK will use the "^" flag, and assumes that the mailer will kill the TIC file after it is sent. NOTE: When you first make this conversion, any existing attaches of TICs will still be truncated rather than deleted. You will need to either run a program that will kill the remaining 0-length files in the HOLD directory (such as Kill0.LZH), delete them manually, or do a run of TICK *WITHOUT* the MailerKills flag to get rid of those old ones.

[RAID]

If you add the keyword "RAID" to your TIC.CFG, HATCH will generate a new file in the HOLD directory. The file will have the extension "RAD", and should enable a future version of RAID (and any other file announcement program that cares to use it) to announce files that are locally hatched.

[REPLACE]

TICK now supports a REPLACE function. If an inbound TIC has a "Replaces abc.xyz" line in it, and if you configure TICK to allow it, the old file (if it exists in the destination directory) will be either deleted or moved to a different directory before the new file is moved to the destination. To activate this feature, add the keyword "REPLACE" to your TIC.CFG. That will cause

the replace function to be active in all areas, and will result in old versions being DELETED if the inbound TIC has that name as the parameter of the REPLACES line. If you follow the "REPLACE" keyword with a valid directory name, the old file will be moved there instead. If a file by the same name already exists in the "old file" directory, TICK will leave the file alone and note that in the log. The Replace function does not accept wildcards for replacement . . . I felt that the security issues were too great. There are also 2 additional LOCAL keywords . . . If you have put a REPLACE line in your TIC.CFG, but do NOT want to allow replacements in certain areas, then add a "LOCAL NoReplace" line in those AREA blocks (after the AREA line, but before the node listing). If you do NOT want REPLACE to be active in most areas, but DO want it in certain areas, then do not put a REPLACE line in your TIC.CFG, and add a "LOCAL REPLACE" line to those AREA blocks. Note that if you want a replaced file to be MOVED to an "old files" directory, you must put a REPLACE line in the CFG.

(e.g. - Local "Replace c:\oldfiles" is NOT valid).

I should mention that TICK will not remove the entry from the FILES.BBS at this time.

[FDLog]

If you add the keyword "FDlog" to your TIC.CFG, TICK and HATCH will log in a format that should be more compatible with Frontdoor. There is one case where the log format will be invalid, If TICK or HATCH detects an error while reading the CFG, but before it has read the line telling it to use Frontdoor mode, the log message will be in the default (Opus/Binkley) format. You can reduce the risk of this happening by placeing FDLog near the beginning of the CFG file so it's read early.

All lines other than the ZONE and AREA structure lines may be in any order, and need not be left justified. Unknown lines are ignored.

DEFINING A FILES.BBS-TYPE FILE

Thanks to a lot of help from Bob Hartman, TICK supports the file-list format needed by many different BBS packages. If you don't do anything special, the defaults establish a file-list file called "FILES.BBS" in each "toss-to" directory. (The "toss-to" directory is the directory that is associated with an AREA, and to which the received file is tossed).

Bob Hartman has provided a great deal of the code to allow multiple formats of files.bbs files. Following is the description provided by Bob. You can set the output format of the files.bbs file by adding a line to your config file with the keyword "LISTFMT" followed by the format you desire.

Any character not proceeded by a % is just put into the string. If the character is a %, then the stuff following it is interpreted as follows:

> y stands for the length of the field to use. If it is a negative number, then it is meant to be left justified. A positive number is right justified. If it does not appear, or is zero, then the field is considered to be unlimited.
>
> z stands for special processing. The first one defined was '1', and it is used only in the file description for TBBS systems. It will append a return and an exclamation point and then continue the description. This is for TBBS' "linked comment" fields. 2 and 3 are also now defined, and are described below.
>
> %x[:y[.z]] (y = LENgth of field, z = "specification")

where x is 1-8 (or 100) and stands for:

1 - file description

2 - path to file with trailing backslash

3 - actual file name

4 - file size

5 - date as mm-dd-yy - There are changes here.
If you simply specify this as "%5", you will get the "file-date". If you specify it like this: "%5:8.3", you will get the SYSTEM date. This should be helpful for RBBS boards, and others that need to list the date received, rather than the file-date.

6 - Day-of-the-week - This will give you the day of the week (spelled out fully) if you specify "%6". This is the day the FILE was created. If you want only the first 3 characters (as in "Mon", "Tue", etc), then specify it as "%6:3". If you want the SYSTEM day-of-week, then specify it as "%6:3.3" for the 3 character string, or as "%6:0.3" for the full string. (A length of "0" is interpreted as an unlimited field).

7 - Hex CRC value for the file.

8 - Prints the name of the PRIMARY area (that the file was received in) to the Files.bbs

100 - See below.

So, for TBBS systems, the file should look like: full_path_name file_name description(40 chars+linked comments)
and the line format would be:

%2%3 %3 %1:-40.1

For RBBS it would be:

%3:-12%4:9 %5 %1:-43 001

where the 001 would be whatever file area they wanted to specify.

For Opus systems it would be:

%3:-13 %1

(which is the default if you don't set the ListFmt descriptor in the CFG file.)

Now for the really fun part! There is another value x can take. The format %100:y.z is used to set up special wrapping for long comment lines. Here is a portion of the description Bob sent me.

%100 is very new. It is used to set the indent and the first indent character for overflow lines. The length parameter (after the colon) gives the indent length, while the specification parameter (after the period) gives the ASCII value for the first character of the line. It is probably wise for people to make this the first thing in their format string if they wish to use it (otherwise they may have wrapped before it gets evaluated). For example, Opus systems might want:

%100:1 This makes it a single space as indent (or) %100:20 which would give a 20 character indent.

They can also do wild things like:

> %100:20.45 which (I think) will output 20 spaces on the new line, but after a '-' is put there first.

%100 comes into play whenever the specification type of a parameter (the field after the period) is a 2. TBBS uses a 1, while this new stuff uses a 2. I imagine that descriptions are the most likely place for it to be used. TBBS needs two special characters for the continuation line, so it cannot be done using the %100. Of course, this makes things VERY interesting, and very tricky. It is easy for people to screw I up, so recommend that they experiment. It is all automatic for TBBS systems, but for others, this hopefully gives them the ability to do what they need.

One interesting sidenote. When using a length with either the TBBS .1 or other style .2 specification, the length given is used on ALL lines. For example, using %1:-40.1 as in a TBBS style line will break the description into a 40 character block, add \n!> (tbbs specific), then add the next 40 characters of the description (and repeating if necessary), padded out with spaces if necessary! This way things always remain in the correct columns if necessary. It shouldn't matter. If people complain, then it might be able to change it later on.

For Opus, try this: ListFmt %100:31%3:-12 %1:-48.2

That tells TICK to set indent at 31 spaces for wrap-around. The first entry in the line is the filename (which starts flush against the left, hence there is no space after the "31"). The filename field is left justified, and occupies 12 characters. It is followed by a space and then the DESCription, which is left justified, occupies 48 spaces, and will wrap to the next line inset by 31 spaces.

The wrap feature is not polished. It will wrap at the specified length even if it comes in the middle of a word.

You now have the option of having the file created called something other than "FILES.BBS", by using the keyword "ListName".

The FileFmt and ListName may be different for each AREA if you want. Here's how it all works:

If you do nothing, TICK will create FILES.BBS in the "toss-to" directory as always.

If you add the keyword LISTNAME, followed by a "simple" (ie.-without drive:\path\) filename, TICK and HATCH will use that filename in each "toss-to" directory.

If you follow LISTNAME by a complete filespec, then the FILES.BBS type file will be named what you want, and will also be created in the specified directory, instead of the default of the "toss-to" directory. This would allow you to send *all* areas descriptions to the *same* description file, as is done for RBBS.

Last . . . If you follow LISTNAME with a drive:\path\ ONLY, you will get the default "FILES.BBS" filename in the specified directory.

Here are some examples of LISTNAME:

LISTNAME RBBS.FIL

This gives you a file called "RBBS.FIL" in each "toss-to" directory.

LISTNAME C:\files\TBBS.TXT

This will give you a single file named TBBS.TXT in the C:\files\ directory for all AREAs (unless overridden by a LOCAL keyword - see below).

LISTNAME c:\foo\

This will give you a single FILES.BBS in directory c:\foo\ for all areas (unless overridden with a LOCAL).

You may, as before, choose an alternate line format with thev "ListFmt" keyword. The format you select serves as the default format for any areas not overriding it with a LOCAL command.

LOCAL AREA KEYWORDS

You may also have differing files.bbs-type filenames and directory locations for specific areas. You accomplish this by the "LOCAL" keyword in the area definition.

AREAs were previously defined as the keyword "AREA", followed by the "toss-to" directory, the AREA's tag, and the (optional) secondary tag. This line was followed by one or more lines of "addresses - passwords - flags". The area ended at the first line not in the standard "Address-PW-Flags" format. This format will still work, but you may now have additional lines between the "AREA" line and the address lines, transforming the AREA line into an area definition BLOCK, instead of LINE.

The additional lines MUST begin with the keyword LOCAL.

NOTE: IF YOU HAVE ANY LINES (INCLUDING BLANK LINES) BETWEEN THE "AREA" LINE AND THE ADDRESS LINES, AND THE LINES DO NOT BEGIN WITH THE KEYWORD "LOCAL", THE AREA WILL END RIGHT THERE, AND NO NODES WILL BE ASSOCIATED WITH THAT AREA!

If you do not define a LOCAL for a certain area, then that area defaults to the name or format defined globally, or to FILES.BBS in the toss-to directory, and "%3:-13 %1" for the format. Examples follow:

```
AREA c:\file\softdist SOFTDIST
LOCAL LISTNAME f:\farm\animal.bbs
LOCAL LISTFMT %3:-13 Hello there %1
1:266/21 password C*
1:261/662 hitom C

AREA c:\file\beans BEANS
LOCAL LISTNAME Foo.txt
7:26/67 whomi H
1:266/13 wham C*
```

The first example creates an area with 2 listed nodes. The "files.bbs" for that area will be called "animal.bbs", and will be located in the directory "f:\farm". The Line format will be the filename in a left justified 13 character field, followed by the constant text "Hello there", followed by the description text. The area ends after the second node, because there is a blank line there.

The second area is tagged "BEANS". The "files.bbs" will be called "Foo.txt", and will be located in the "toss-to" directory called "c:\file\beans". The format will default to the format defined in the global LISTFMT statement, or to "%3:-13 %1 if none has been defined.

[LOCAL STOPDUP]

Tick now allows a LOCAL STOPDUP feature. You still need to have the main [STOPDUP c:\dupdir] line in your TIC.CFG to activate STOPDUP processing in the first place. The presence of a LOCAL STOPDUP in an area all by itself will NOT turn on STOPDUP in that area.

If you do not put a [LOCAL STOPDUP dupfile] statement in your area, then that area's dup file will use the area's tag with a DUP extension as the filename for its dup file, as before. If you *do* put in the new local line, then the "dupfile" name that follows the "Local Stopdup" will be the root name for that dup file. It will be appended with a DUP extension. For example:

```
AREA d:\file\long BRAVO
```

Local Stopdup Tango
1:266/12 password *C
1:13/13 wordpass C

The DUP file for the above area would be called Tango.DUP, and would reside in the Dup directory specified in the regular STOPDUP cfg line. Without the local line, the file would have been called BRAVO.DUP.

What this does is gives you the option of merging certain echos to use the same DUP file. This possibility should be used with care. Just because *YOU* happen to carry both of the merged echos, that doesn't always insure that all your downstream nodes carry both echos. If a DUP entry generated from echo A stops the same file from passing in echo B, you may be preventing nodes who carry only B from getting it at all.

[LOCAL BigDesc]

Tick now has provisions for sending large descriptions to a message in netmail or an echo. Here's how it works:

In each are you want to activate large-description capability, add the local line:

LOCAL BigDesc mask tag [tag ...]

Where "MASK" is a file mask indicating what files will trigger a description message. If a file fitting that mask is being echoed in that area, the file will be echoed as normal, but also a PKT in the inbound will be created, containing the text of the message. The TAG entries indicate where the message should end up. If the tag is an echomail area tag, then the description will enter that echo. If the tag is "*", the description message will end up in your netmail area. For example:

Local BigDesc *.SDA newfiles *

This will result in any *.SDA files (which arrive with valid TICs) to be tossed to the file echo area and echoed to other nodes in that area, (all as before). In addition, a PKT will be created in the inbound, with a message in the echomail area "newfiles", and in your netmail area. You can send a description to as many echos as you please, simply by adding additional tags to the local Bigdesc line.

Right now, I'm filtering any characters below 20H execpt for C/R and Tab, and I'm also filtering any characters above 7FH. Message size is limited to approx 10K. A description larger than that will be truncated.

Presumably, the SDS will not want to use SDA as the agreed upon extention to trigger a description message. Since the mask is local to each area, that presents no problem. (I suggest the mask *.DES, but that's not up to me).

With this method, a user calling your board might have the choice of reading a description echo online, or of simply dpowloading the DES files in the file area.

You will probably want to define a few more lines in your TIC.CFG. The message fields TO, FROM, SUBJECT default to "All", "Tick x.xx", and the filename of the file being shown respectively. To change them, define the following lines:

```
TO Binkley Zealots
SYSOP Tom Hendricks
SUBJECT Latest Binkley Utility!
```

The ORIGIN line defaults to "Lazy Sysop (Z:NET/NODE)", and can be changed with the line:

```
ORIGIN  Your source for BBS Utilities!
```

Please note that these 4 lines are GLOBAL in scope. There is presently no provision for LOCAL definitions here.

Note that you should *NOT* add the address at the end of the line yourself, as TICK adds it for you. The Origin line is limited to 79 characters, including the " * Origin: " which TICK also adds.

Some tossers will not toss PKT's if there is no SEEN-BY line present in an echo message (Squish is one such program). To please such programs, add the keyword PKTSEENBY to the TIC.CFG.

The PKT will contain your main address as the TO and From address in the headers, and in your Origin line. If you run a secure tosser, you will need to add your own address to your areas.bbs (or whatever it's called on your system) for that echo.

[LOCAL DupByCRC]
[LOCAL NoDupByCRC]

If you add the keyword DupByCRC, and have the CRC2DUP option active, then STOPDUP will cause TICK to test not only the NAME of the file, but also the CRC32 of the file. A "hit" will only occur if the the CRC matches as well as the name. This will allow same-named files to pass Stopdup if the files are different than the one previously seen. If the older / same-named file is still in the "toss-to" directory, it will (presently) be overwritten.

If you choose to not set the global DupByCRC keyword, which affects ALL areas, you can selectively still employ the new dup-check routine in selected areas of your choice by adding a "LOCAL DupByCRC" to the list of LOCAL lines following the AREA definition line.

Should you decide you want to use the CRC dup checking routine in most areas, but wish to exclude a few, you may place the DupByCRC keyword in the TIC.CFG, and add a "LOCAL NoDupByCRC" line to the list of LOCAL lines following the AREA definition line.

If you decide to use the newer CRC-based DUP detection, any DUP file lines that contain only a filename and no CRC number (such as may exist from previously running the programs without CRC2DUP in the TIC.CFG) will be tested by the older stopdup logic.

[Local Passthru]

To make an area a passthru area (Much like passthru areas in EchoMail), Use the PassThru keyword in its AREA block. TICK will make its attaches as usual, but will *not* update the FILES.BBS. At each run thereafter, TICK will test all attaches (either MSG or FLOs, for FIDO or native mode respectively), and will kill the file when there are no active attaches. If the passthrough file is also a pre-release file, the passthrough process will be postponed until the file is released. Be aware that if you define a secondary area, and that secondary area is passthru, the file will be tossed there and treated as a passthru file (ie deleted). Likewise, if a nonpassthru area is secondary to an area that *is* passthru, the inbounds will be tossed to the non-passthru area and treated as non-passthru. (These actions will be influenced by stopdup, as previously).

MORE ON CRC-32

CRC-32 checking is present in TICK/HATCH. All files processed by either program will have the CRC computed and present in the outbound TIC's. If you add the keyword "CRC" to your CFG file, then inbound TIC's that have CRC's in them will have that CRC compared to the computed CRC. If the numbers don't match, the file will fail. TIC's received from older versions, having no CRC line, will be spared this check, but outbound TIC's will still have the CRC line in either case. One negative side effect of this is that significant time is required to calculate the CRC. I feel that the added safety is worth it, however.

If you would like the CRC value to appear in your log file, add the keyword "LOGCRC" to your TIC.CFG. If you choose to log your CRC's, the value in the log will be followed by a star (*) if the inbound file also had a CRC in the TIC.

If you want to also save the CRC in the *.DUP file after each filename, add the keyword "CRC2DUP".

Having the CRC compared to the CRC present in the inbound TIC can cause problems in at least one situation. If you run a program like STRIPZIP to eliminate potentially dangerous ASCII sequences from received ZIP files, the CRC of the file you received will no longer match the CRC in the TIC file. To get around this problem, there is a command line switch for this situation. What you do is to NOT define "CRC" in your TIC.CFG file. When you receive new files, first run TICK with the command line switch "/OC". That will cause TICK to test the unmodified file's CRC against what is present in the TIC. (It also tests for proper password, proper FROM address, etc.) If the CRC's do not match, the TIC file is renamed to "*.BAD". If all is well, the file is not further processed, and outbound TICs/FLEs are *not* created. This mode is a "check-only" mode. After running TICK with the /OC switch, you can then run STRIPZIP. After Stripzip, have the batch file call TICK, this time *without* the /OC switch. Tick will process normally, and will ignore the CRC in the inbound TIC's, since you don't have CRC in the TIC.CFG. The new CRC generated in the outbound TIC's will be proper for the file as you are sending it.

AKA's

Several of you have asked for this feature. Here's how it works:

For each alternate address you are known by, add a line to the CFG file beginning with the keyword "AKA", and followed by your address in [zone:]net/node format. If the zone is not present, it defaults to the default zone (the first ZONE statement in the CFG file). For example:

AKA 1:1/313

When I place this in my CFG file, I will add both my main address of 1:266/12, and 1:1/313 to the seenby list in my outbound TIC's and Fle's.

You may specify for each node you send to, what address you are sending from. All addresses still appear in the seenbys.

Here's how it works: There is an additional flag for the node's flag field (the field where you specify if that node is <C>rash, <H>old, <*>, etc). If you want that node to receive from you as your primary address, you need make no change. If you want to send to that node as an AKA, the new flag is "A", followed by the number of the AKA to use (from 0 to F in Hex). The number corresponds to the order that you have defined AKA's in your CFG file . . . The first AKA is "0", the second is "1", etc.

NOTE THAT THE NUMBERING STARTS AT '0', NOT AT '1'.

For example, to send to node 2:512/26 using my first AKA of 1:1/313, I would set the node up like this:

2:512/26 Password HA0*

This will instruct TICK to communicate with that node as 1/313, send it HOLD, and accept files from him as well. Since I will be now sending to 2:512/26 as 1:1/313, he must set up my node as 1/313 in his TIC.CFG, and need NOT set up my node as 266/12.

The number after the "A" must be a single character, and must not be separated from the "A" by a space

USE CAUTION WITH SENDING TO OTHERS AS YOUR AKA! If not carefully used, this will increase the likelyhood of a DUP ring. It's recommended that AKA's only be used when crossing between zones at a designated "gate", which does not loop back into the first zone somewhere else. Oh yes . . . The limit on AKA's is 16 of 'em.

FINDING THE CFG FILE

The TICK program, and accompanying HATCH program should be placed in any convenient directory. When run, both programs need to find the configuration file. Tick / Hatch first looks to see if you have entered the configuration file as a command line parameter. If you choose this method, the following form applies:

```
TICK /Cc:\other.cfg
```

The "/C" may be upper or lower case.

If there is no command line parameter, TICK next looks to see if you have set the environmental variable TIC as in:

```
TIC=C:\OTHER.CFG
```

Where C:\OTHER.CFG is the configuration file to be used this run. If neither option is chosen, then the default is to use the file TIC.CFG in the current directory. If no such file is available, TICK aborts with a fatal error.

In addition to the CFG file, you must set the TZ environmental variable to the difference between local time and GMT. This is the same variable used by Opus and many other bbs-related programs. For the Eastern time zone, it would be set to TZ=EST5EDT. In your batch file, have the line:

```
SET TZ=EST5EDT
```

WARNING: Bob Germer pointed out to me that this form of TZ variable can cause problems for Opus, depending on what version of DOS you run. If you experience problems with other programs using this TZ string, you could use the form:

```
SET TZ=EST5
```

This will work for both TICK and Opus. Its disadvantage is that you will need to change the "5" to "4" during daylight savings time. The first format will give the correct results within TICK in standard and in daylight time zones without the need to manually alter the string. An alternative would be to set the EST5EDT format in your batch file that calls TICK, and reset it to the EST5 format after TICK has been run.

TICK AND OTHER FILE-ECHO SOFTWARE

The TIC file format should be compatible with the files produced by Randall

Greylock's UPCL program. TICK also offers support of the FLE file created by Ron Bemis' FLEA.

HATCH

Files are started into a file-echo with the program HATCH. HATCH uses the same configuration file (see below) as TICK, and the CFG file must be present for either program to run. File-echos are set up in AREAs, each AREA having an echo tag, similar to the echotag in echomail. An AREA name may be up to 8 characters, and must consist of characters legal in a dos filename. You may have multiple AREAs (and probably will). When you run HATCH, you need to tell it what AREA you are hatching the file in, the file's name and location, and the description to put in the FILES.BBS of the directory associated with that AREA. This may be done interactively, or from the command line.

If you choose to use hatch interactively, as most do, just enter HATCH (optionally followed by the CFG file to use, as in :

HATCH /cOTHER.CFG

Hatch will first ask you how many days to wait before releasing this file. If you want it to be released immediatly, simply press enter. If you want to use the "pre-release" function of TICK / HATCH, then enter the number of days to wait until it can be released. Please see the notes later in the documentation about pre-release files.

Hatch will then prompt you for the AREA (file-echo tag) that you want to send the file in. If the AREA is not valid, HATCH will let you know.

Hatch will then prompt you for the filename. HATCH prefers that the file you HATCH to be in the "toss-to" directory associated with the AREA at the time of hatching. If you MUST hatch from a directory normally NOt associated with that AREA, You will need to include the full pathspec as in previous versions of HATCH. If you place the file in the "toss-to" directory first, you need only give the filename when you hatch. If the file is not found, hatch will say so and abort.

Then, Hatch will ask for the DESCRIPTION that will be placed in the FILES.BBS for that file. It is recommended that the file to be hatched be located in the "destination directory" normally associated with that AREA. If it is NOT, it will still send the file just fine and the FILES.BBS will be updated. However, your BBS will likely report that the file is MISSING.

If you choose to enter all or some of the information from the command line, there are additional command line switches. These are the most commonly used (Their usages are described below):

/A , /F, /R0, /ON, and /D.

The /A switch specifies the file-echo AREA to hatch into. The /F switch specifies the file to hatch. The /D switch specifies the description line to use for the FILES.BBS. Like the "/C", all the new parameters are NOT separated from the switch-letter. e.g.: The area SOFTDIST would be written as /aSOFTDIST instead of /A SOFTDIST.

An example:

To hatch the file FOO.ext in area SOFTDIST with the description line "This is the Description", use the following line:

HATCH /aSOFTDIST /fFOO.ext /dThis is the description >> logfile.log

Any parameters NOT specified on the command line are prompted for as before. If the /D description switch is used, it must be the last switch on the command line, as it is of variable length and number of tokens (words). The description may be up to 80 characters long, but it is strongly recommended that the description be kept below 50 characters.

In conjunction with DAYNBR.ARC, this should make hatching of nodediffs and other automatically generated files effortless and automatic with the proper batch file.

The above example shows another design feature of TICK and HATCH. Both are designed to allow redirection of output to a log file for review of what has happened. The sign-on credits will go to the screen, not the log. The log is formatted to match the form of the Binkley and Opus logs. In fact, I use the same log for Binkley, Opus, and Tick here. The ">>" redirects the log output as an append.

Hatch allows you to re-enter an incorrectly entered area or filename. The AREA comes first, then the filename, then the description when running in the interactive mode. (The AREA *must* precede the filename in interactive mode, since TICK needs to know where to look for the file when you don't enter the path).

Having HATCH loop back for a file not present or a misspelled area name could present problems for those of you who use batch processing and don't want the hatch to "hang" looking for keyboard input that is never coming. For this reason there is a "NOWAIT" option available. Either put the keyword "NOWAIT" in the CFG file, or have the batch call HATCH with the switch "/ON" (without the quotes. The "O" stands for "Option", by the way.)

PRE-RELEASE AND SECONDARY AREAS

Tick 2.10 added several enhancements, two of the most significant of which are PRE-RELEASE and SECONDARY AREAS. These are more difficult to describe than to use. What follows is an attempt to explain them.

A little historic background may be helpful. Back when the SDS as we now know it got started, one of the ideas was that we could make provision for software to be distributed within the SDS-RC structure into each region before the official release date, so that when a major release, such as a new Binkley came around, it would be readily available everywhere at once on release day. This concept is PRERELEASE. To date, the only implementation had been in FLEA. That provided the ability to distribute a file throughout the network, but would delay the toss to the downloadable directory until the official release time. That was good so far as it went, but it made the assumption that the SDS would consist of a small number of distribution nodes, and that the net-level sysops would get the files by FREQing from their regional SDS nodes. The SDS and other distribution networks developed in a different direction, however. In most cases the net level nodes are directly linked into the networks, elimination the need for FREQ entirely. This expansion rendered the simpler prerelease method unworkable.

What was really needed was a method of limiting the distribution of pre-released files to the core structure of the distribution network, and to pass the files "down the pipe" on the release date. TICK should now makes this possible.

When a file is hatched into an echo, it will now be possible to specify a release delay of a certain number of days. The file will immediately be sent only to those nodes which are designated as having permission to receive pre-released files. Those nodes will likewise send the file only to similarly privileged nodes. Until the time of release, the files will reside in a special "quarantine" directory (QDIR). When the release time arrives, the file will be tossed to its destination directory, and sent to those nodes which have not yet received it. This is the basic PRE-RELEASE concept.

Within an AREA block, those nodes that may receive a pre-release file before the release date should have a "P" flag in the flag field. When you HATCH a file, the prompt for "Release in how many days?" is now active. If you just press <enter>, the file is a normal file and is not delayed. If you have a file that you want to send as a pre-release, place the file into the QDIR directory. (QDIR SHOULD NEVER BE A DOWNLOADABLE DIRECTORY). When the prompt for release days comes up, answer with the number of days to delay. (You may bypass the prompt by using the /Rx command line option, where "x" is the number of days to delay. /R2 delays 2 days, /R0 is a normal release.

If you have specified a pre-release file, then only those nodes that have a "P" flag will get the file immediately. Please note that only nodes that are also running

TICK 2.10 or higher should be given "P" status. Those nodes will have TIC's and attaches created. The TIC's will have the seenbys for all nodes that will eventually receive the file. In addition, there will be a "*.TOK" file also created in the HOLD directory. That file resembles a TIC closely, but has the seenby's for the pre-release nodes only.

Each time you run TICK 2.10+, the program will first scan the HOLD directory for TOK files. If it finds any, it examines them to see if the associated pre-release file is "ripe" yet. When it is, the file is moved to the destination (downloadable) directory, the FILES.BBS is updated, and new attaches are created to those nodes that do not have the "P" flag.

Pre-release can be set up in two ways. Currently, in the SDS the public file-echos go not only to the SDS nodes, but to the "end-user" nodes as well. If files are to be directly pre-released into the public echos such as SOFTDIST, then any node that receives pre-release files in that fashion must be running TICK 2.10 or higher, as lower numbered version will forward all files without reservation, as will FLEA (with minor reservation). This could still be used in this fashion if all SDS RC's decide to run the new TICK when it's ready. The second way, and probably the more secure way, would be to set up a separate pre-release file-echo associated with each public file-echo. For example, PRESOFT could be associated with SOFTDIST, PREBINK could be associated SDSBINK, etc. When these would be set up, the public area would be listed as a secondary area. Then all nodes listed in the PRExxx file-echo could be "P" nodes, and non-P nodes would be in the secondary area. This way, there would be less chance of accidentally linking to a non-prerelease node in the public echo. This setup would not effect the normal operation of the public file-echos at all.

EXAMPLE:

```
AREA d:\prebink PREBINK SDSBINK
        1:261/662 password *HP
        1:135/204 passw2    HP

AREA d:\sdsbink SDSBINK
        1:261/662 passw3    *H
        1:266/13  passw4     C
        1:132/101 passww     C
```

In this example, you can receive normal file from 261/662 in either PREBINK or SDSBINK. You can receive Pre-Release file from 261/662 in PREBINK. If you receive a normal file in PREBINK, all nodes listed in BOTH areas will get it immediately. If you receive a normal file in SDSBINK, then only those nodes in SDSBINK will receive the file, and if you receive a pre-release file in PREBINK, then it will be echoed to 135/204 immediately, and will be released to the rest of the nodes when it's ripe.

NOTE: With the addition of pre-release, it's VERY important that the TZ

environmental variable be set properly. If it isn't, release times will be off.

SECONDARY AREAS

SECONDARY AREAS is another added feature, which may be used independently or in conjunction with pre-release. Presently, file echos are defined with an echo tag, just as in echomail. It is now possible to associate a secondary echo (carrying its own echo tag) with a primary echo. In such a setup, any files hatched or received in the secondary area would echo only in that area. Files hatched or received in the primary area would echo in both primary and secondary areas. For example: Someone recently mentioned in SOFTWARE that he was receiving the new releases of "Remote Access" BBS directly from Oz in an echo with the tag of RA_DIST (That's not the actual name, but it will serve for now). He now has to manually alter the tag to SOFTDIST to distribute it via SDS. If RA_DIST is set up as a primary area, with SDSRA as its secondary area, he could receive the file in RA_DIST, and distribute it in SDSRA automatically. Note that the use of SDSRA as a secondary area of RA_DIST does not interfere with its usual use, where it is defined as a primary area elsewhere in the TIC.CFG file. Also note that echoing in SDSRA does not "flow backwards" into RA_DIST. A segment of a TIC.CFG with such a setup follows:

```
Area c:\sds\softdist SDSRA
    1:266/12 Password *C
    2:512/26 Pass2     H
    1:116/29 Lhz       H

Area c:\sds\ra_dist RA_DIST SDSRA
    3:333/29  Pass3   *C
    1:261/662 Pop     *C
    1:116/29  Lhz      C
```

In the above segment, files passed in SDSRA will echo as they always have. They will NOT be echoed to 1:266/12 or to 2:512/26. Files received in RA_DIST from 3:333/29 will be echoed to 1:261/662 and 1:116/29 as RA_DIST, and will be echoed to 1:266/12 and 2:512/26 as SDSRA files. (1:116/29 will NOT receive files received in RA_DIST as a SDSRA file, because when the primary area is processed, his seenby is added before the secondary area [SDSRA] is processed ... He WILL receive files that were received in SDSRA as SDSRA files, however.)

USING PRE-RELEASE AND SECONDARY AREAS TOGETHER

Although pre-release is independent of secondary areas, a good use of the combination would provide additional security against the premature distribution of a pre-released file. What I thought of doing was this: Set up a primary area for each echo, and use those areas for pre-release files only. Each primary area would have as a secondary area, the file echoes we all know and love. Files that were not pre-release would be distributed in the usual echoes. Pre-release files would travel in the pre-release echoes, and be released into the normal echoes at release time. For example: SOFTDIST is defined as a primary echo in one part of the TIC.CFG. PRESOFT is defined as a primary echo in another part of the TIC.CFG, and has SOFTDIST defined as a secondary area to it. Only authorized nodes would be linked to PRESOFT, while the whole world is linked to SOFTDIST. A pre-release file would be distributed by the SDS-RC structure via PRESOFT, and on release date would be tossed into SOFTDIST in each region simultaneously. By using the separate areas, the chance of an unauthorized node inadvertently being sent a pre-release file is reduced.

COMMAND LINE SWITCHES

There are a number of command line switches that are useful. At the present time these consist of /C /A /F /D /ON /OC /Rx /X. They are defined as follows:

"/Cc:\subdir\other.CFG"

This will define the file c:\subdir\other.cfg as the CFG file to use for this run of TICK or HATCH.

"/Asoftdist"

This tells HATCH that the AREA you are HATCHing into is the area called SOFTDIST. It is ignored when used with TICK.

"/Fc:\utility.ext"

This tells HATCH that the file you are HATCHing is c:\utility.ext

"/DThis is a useful utility"

This will set the DESCription to "This is a useful utility". It is used only with HATCH, and if used must be the LAST switch on the command line.

"/ON"

This switch tells HATCH to not loop back for corrected input when it can't find the FILE or AREA you are HATCHing.

"/OC"

Runs TICK in the "check mode". Errors will set the inbound *.TIC to *.BAD, but no tossing or forwarding of files occurs.

"/R2"

This is used by Hatch only, and secifies the number of days to delay the hatching of this file. "X" is the number of days to delay. /R2 delays 2 days, /R0 is a normal release.

"/X"

This is used by Hatch only, to allow you to designate a filename to replace with the file you are now hatching. The switch is /X, and must be immediately (no spaces) followed by the filename you are replacing. The filename should NOT contain a path, Drive letter, or wildcard characters (* or ?). If you enter "HATCH /X" by itself (no filename to replace), HATCH will not ask you about the replace function in the interactive mode.

ERRORLEVELS

Tick and Hatch exit with the following errorlevels if there is a fatal error:

1 = Error reading a file
2 = Error Writing a file
3 = Out of Memory
4 = Invalid CFG file
5 = Invaild PATH or filespec
6 = Processing Error

PARTING COMMENTS

I hope you find this utility useful. Tick may be used in any non-commercial environment free of licensing charges. It may be freely distributed so long as there is

no charge for it and it is distributed only in the original archive.

There is NO warranty with this product, and I assume no responsibility for any damages it might do to your system, nor any consequential damages. If it breaks, you own both pieces.

In other words, you assume all risks should you decide to use it. It is working well on my system, and on the systems of those brave sysops who have helped me beta test it.

I want to thank George Peace (1:13/13), Jeff Myers (1:266/13), Bob Germer (1:266/21), Don Dawson (1:141/730), Randall Greylock (1:321/202), Mike Fuchs (1:266/71), Tom Hendricks (1:261/662), and many others for their assistance in testing.

Special thanks go to Bob Hartman (1:132/101), for sending code to allow us all those wonderful variations on the FILES.BBS-type file. I'm certain that those of you who run a BBS that can't use a FILES.BBS are especially grateful as well.

I also would like to thank Jeff Nonken (1:103/322) for the code which allows the paths set in the environment to end either with or without a backslash, Scott Dudley (1:249/106) for help with the command line parser, and Jim Nutt for the MSG structure I used in implementing the "Seadog-type" file attaches.

Documentation produced by Erik VanRiper (1:107/230).

Barry Geller
Maple Shade Opus
1:266/12
609-482-8609