Quip, The Fortune Cookie Program From Hell

COLLABORATORS			
	TITLE : Quip, The Fortune Cookie Program From Hell		
ACTION	*NAME*	*DATE*	*SIGNATURE*
WRITTEN BY		October 27, 2024	

REVISION HISTORY			
NUMBER	DATE	DESCRIPTION	NAME

Contents

Chapter 1

Quip, The Fortune Cookie Program From Hell

1.1 Quip, The Fortune Cookie Program From Hell

```
+------------------------------+
|       |
|        Quip   |
|       |
|   The Fortune Cookie Program  |
|      From Hell    |
|       |
+==============================+
|  Joseph Edwards Van Riper III |
+==============================+
|       |
|  A Cheese Olfactory Workshop  |
|     Production    |
|       |
+------------------------------+
```


1.2 Legal Stuff

```
+------------------------------+
|       |
|    LEGAL STUFF    |
|       |
+------------------------------+
```

As is the way of man, sometimes, you just have to spell stuff out to people.

1.3 Terms of Use

You are free to execute this program in any way you want. You may put it on a disk and fire live rounds from your favorite rifle into it, lick the diskette you may have picked this up from.. heck, you can even use your favorite sector editor and write little <00>'s into it.

Of course, you may also use this for your BBS or your home system.

You owe me nothing except perhaps gratitude.. maybe not even that after you read some of the supplied quips.

Most people call this "Freeware", except I add on other special bonus to the whole thing... I have included the Source Code! With THIS in mind, the program should probably be more properly called "Public Domain". I only ask that you do not claim this work as your own, unless you modify it significantly, and even then, I ask that you include my name SOMEWHERE in the credits, by way of respect for the work that went into this frivolous project.

Otherwise, feel free to enjoy this program in any way that you feel you can find enjoyment.

1.4 Distribution

If you dare to charge people anything more than what Fred Fish charges for a disk from his definitive PD/Shareware collection, you will be in violation of my right to have this product distributed freely to all peoples who wanted a product like this. That is to say, You Should Be Ashamed Of Yourself, Where Is Your Mother To Discipline You?

If you are a user, and you happen to come across this work, and you've paid for it, you've been ripped off. Please try to get in contact with some kind of authority in your respective country, to get your money back, and maybe eat the evil bastard for lunch. At the very least, you should feel insulted that he thought so little of you as to allow this remark to remain in the documentation.

If you have written a very nice program.. perhaps a BBS or a database or something, and you wish to include my program with the distribution, regardless of ShareWare or Commercial fees, you are welcome to include this program in your distribution, as long as it is made clear that I (Joseph Van Riper) am the author of Quip. I do not even require a copy of the registered version of the program be sent to me (although that would be exceptionally nice <grin>). It should also be made clear that Quip is a Public Domain program, freely available, and that the user is paying only for your own product.

1.5 Disclaimer

I, Joseph Edwards Van Riper III, and all peoples working for the Cheese Olfactory Workshop (in whatever guise), do hereby accept no responsibility for the results of this program. Since you haven't paid for it, you accept full risks for what this program might do. We will not be held responsible for anything lacking in the documentation, either... you, the user, are to accept the full risks of running this program.

While we've taken great pains to insure that nothing terribly serious could happen as a result of running this program, we cannot accept responsibility for this program's actions, or the documentation's wording which may lead you to whatever actions. Enjoy this program, but don't try to pin anything on us <grin>.

So, if by some weird fluke, you read something in the documentation that suggests you try some parameter, and you find yourself in Alpha-Centari, sipping Methane in a plastic container, wearing naught but a joystick and several rubber skid-stops, we cannot be held responsible for it. Find your own way home.

I strongly recommend, however, that the user read the instructions for the use of this program before running it, in order to minimalize any potential damage. Please keep in mind, Quip modifies certain files under certain conditions that are easily predictable. Quip also creates/deletes certain files according to another file/string's text... eg:

".profile"

Might become:

".profile.tab" (or) ".profile.0"

while

"RAM:garn.dat"

Could become:

"RAM:garn.0" (or) "RAM:garn.tab"

And

"RAM:blooie"

might become

"RAM:blooie.tab" (or) "RAM:blooie.0"

Also, all reported rumours about Quip stealing girlfriends away from men are completely without base. Even more untrue is the unusual rumour that Quip will make you impotent, or will reduce your IQ by 5 points with each invocation. The one about Quip making calls to the serial device is a total fabrication, and anything about Quip having Artificial Intelligence is truly overblown.

```
Read the docs.
```

1.6 History Lessons

```
      +------------------------------+
      |       |
      |   History Lessons |
      |       |
      +------------------------------+
```

```
A Gilgamesh was the King of Uruk.. no, wait.. you can't be interested in THAT
subject right now!

But you might be interested in  Quips History  or perhaps  COWs History .

 Quips History
 Cows History
```

1.7 Quips History

```
      +------------------------------+
      |       |
      |   Quip's History  |
      |       |
      +------------------------------+
```

```
Quip has gone through a lot of changes over its time, and I didn't really
keep a very good handle on all the various changes from version to
version, but I can try to give somewhat of an idea, if you REALLY care
about it.

 Quip One
 Quip Two
 Quip Three
 Quip Four
 Quip Five
 Quip Six and Others
 Quip Seven part One
 Quip Seven part Two
 Quip Seven part Three
 Quip Seven Part Four
 Quip Seven part Five
 Quip Seven part Six
```

1.8 Quip One

```
Quip 0.01 (perhaps July 1991, Lattice C++)

The very first version of Quip was written because I wanted to see if I
could program a fortune-like program for the Amiga (because I hadn't seen
```

any up to that time). I wanted to do it so one wouldn't have to have an extra table file hanging around the system. Of course, this meant that the program would run rather slowly. However, Quip 0.01 was SO incredibly slow because I hadn't figured out that Level 1 I/O meant non-buffered, and thus, I had to wait for disk-movement. Boy was *I* an idiot!

1.9 Quip Two

Quip 0.02

I had discovered that Level 2 I/O would speed things up considerably <grin>.

1.10 Quip Three

Quip 0.03

I implemented the tablefile, as an option, because I was quite tired of how slow quip retrieval was. It was pretty cool, suddenly seeing my little quips pop up almost instantly!

1.11 Quip Four

Quip 0.04 (sometime in 1992)

I think I fixed some bugs in this version. I can't remember what they were, though. I DO remember that they were SO bothersome that I completely re-wrote the code, trying to take better advantage of C++'s object oriented features. This also started the MS-DOS series of Quip programs, distributed almost exclusively in Charlotte, NC, and Minneapolis. Prefix and suffix files support started to appear in this version.

1.12 Quip Five

Quip 0.05

More bug-fixes. There were a couple of 'sub-versions' (like this wasn't already a sub-version) that were meant to fix minor problems (such as reading the filename from the arguments correctly in both Amiga and MS-DOS versions).

1.13 Quip Six and Others

Quip 0.06 (and incarnations)

Again, more bug-fixes, and some more features (better argument parsing, and all kinds of nifties). Included with the archive for the Amiga version was a program that helped Citadel sysops convert archived messages into Quip datafiles in a nifty little format (handled anonymous messages, networked messages [although there was a stupid bug I put in it that will make it generate messages that look pretty dumb.. maybe I fixed this before I sent it out.. I dunno], time and date stamps, etc). Pretty cool, really. I also had assembled a rather huge datafile filled with over two thousand quips in it, and started distributing this file separately. It was at this time people started complaining about how large the datafiles were, and could I PLEASE make them smaller somehow. That's when I started working on.... (the next version of Quip).

1.14 Quip Seven part One

Quip 0.07 (June-July 1993, Amiga E compilor)

I had had enough of working with the Lattice C++ program. I didn't have problems programming in C++, but I had problems with messing around with a stupid compilor. I sorta copped an attitude against Lattice, 'cause I felt sorta funny about them charging for technical support when they give such a crummy manual. I started looking for PD compilors, and stumbled into g++... which gobbled a full quarter of my humble hard-drive, so I dumped it in favor of Amiga E. WOW! Talk about a NICE compilor! The docs are not the greatest, but functional enough for me to get around, and there was one major error in the docs that almost caused me fits when I first started working in E, but after a while I settled into it, and voila! Quip!

This version of Quip was meant to be the last version I should have to mess with. With that in mind, I gave it more features than the average person should have to mess with. Unfortunately, MS-DOS support had to be dropped <giggle> since I programmed this in E, but the code size dropped dramatically, and the functionality of the program has increased beyond anything one could really have expected in such tiny code. The source code is roughly 1,300 lines long, and could be tightened up if I bothered. I finally added icon support, and created the new QuipScript "language" for use by Quip. I changed a few things in the CLI interface to make it more Amiga-like instead of unix-like. It now uses ReadArgs, allowing me to avoid having to write a function to handle '?' and stuff like that.

1.15 Quip Seven part Two

Quip 0.07a

Oops.. so much for the version to end all versions. I fixed the bug that caused all quips that were generated from a file not having a tablefile to have a '@' character preceding them... no longer has the '@' character. Super-easy fix... stupid oversight on my part (shoulda tested for this

before sending it out.. to one person <grin>).

Fixed bug that caused it not to release its deathhold on various files when the user broke out of it (if invoked by icon).

Also fixed another silly bug that would make Quip exit with ridiculously high error codes, screwing up scripts. Now exits properly (another easy fix).

Enabled randomized quips for datafiles without tablefiles (thanks to Lars Magnus Nordeide, who wrote WiseCrack, and gave away the source code in C.. cute algorythm.. I have no idea why I didn't see something so simple myself). Also, using his idea of taking advantage of Nico's ReqTools library, I've added support for it in the form of the REQ tooltype/cli argument. Oddly enough, this was a real bitch to do, since normally I don't bother to put all the quips in a single variable <sigh>.

1.16 Quip Seven part Three

Quip 0.07b

Created delay able quips. Now you can use the requesters, and not have to press 'Quip' to quit... after a specified amount of time, Quip will quit on its own. Actually... after an obscene amount of time, Quip will quit on its own anyway (something like 90000 seconds). Of course, if you specify the NUM keyword, it'll print NUM quips, one after the other, waiting the specified amount of time before quipping on.

1.17 Quip Seven Part Four

Quip 0.07c

Added UPDATE feature, so tablefiles no longer have to be created from scratch. Also fixed a bug that caused 40 bytes of memory to disappear each time Quip was run from CLI (stupid error.. I forgot to FreeDosObject() a pointer). Quip is now a very CLEAN program <grin>. Quip now actually does the WHICH command, and during roll-over, will not read the '0' quip (the information that comes between the quip-number and the first quip). Within the scripting language, I created a new option called RESET , which allows you to reset all the various options to their default values. This means I changed the way I handled the setting/resetting of various values after MAKE_QUIPS and IN_FILE , too. I also included a new bug that uses the user's serial port to log into various nuclear weapons sites and fire warheads at the state of Georgia. I'll leave it to you to figure out if I mean United State's Georgia, or the country of Georgia...

1.18 Quip Seven part Five

Quip 0.07d

Quip now doesn't default when given bad arguments.. instead it'll give you some kind of idea why the arguments were bad. In fact, all the error texts for Quip have been changed to be MUCH more informative. Quip now tries to give you a reason why it couldn't do what you wanted <grin>. Thank you, Commodore, for Fault()! Also, you may now specify a file to send error messages, and anything else that isn't specifically a quip. That is to say, using the ERROR command, you may redirect anything that isn't specifically a quip (or isn't a prefix/suffix/format text, or isn't a bad keyword in QuipScript) to another file. Could lead to dancing.

Yet another improvement.. Quip now looks for a default script (called 's:quip.script'). If it finds it, it'll QuipScript the script automatically, thus allowing you to handle a bunch of stuff by simply calling 'Quip'. Also, QuipScript files no longer HAVE to put a # IN_FILE keyword in the script... QuipScript now defaults to "S:Quip.dat", just as the command-line/icon options do. NOTE: if Quip finds an S:Quip.dat file to execute, it'll quit when the script is finished (just like any other script file). I've also changed the code so you can specify a datafile to work with a particular script.. so if you invoke Quip with a DATA parameter, as well as a SCRIPT parameter, and the script doesn't set the datafile within it, the datafile you specify will be used instead.

I also changed the FORMAT command so that, if using a REQ or DELAY argument, the FORMAT ted string will appear in the menu bar, rather than in the text itself. Something to keep in mind.

1.19 Quip Seven part Six

Quip 0.07e

I fixed a few annoyances with the error-reports. Quip now appends error messages with a date/time stamp heading the error-report. QuipScript ing now sends #'s with non-commands (normally echoed to stdio while the script is running, always preceding any quips) in the order that it appears in the script. Therefore, if you echo something to stdio, then generate quips, and echo something else to stdio, you'll see the text before, then after your quips, instead of always before. You may also use a switch to toggle whether such messages are sent to stdio or to the error port.

A new MIDDLE option was created, to allow text to appear between quips when using the NUM option.

Prefix/suffix/middle/format options may now be direct text, or files specified in the command-line options/tooltypes (or in the QuipScript).

Prefix/Suffix/Format/Middle files/texts may now be whatever size you want (that the system can handle). The only restriction is that the actual name of an environment variable may be no more than 80 characters long. Environment variables may now hold text of any length.

You may now run a command using the filename env:quip.tmp as an argument from within Quip. This is useful if you want to do things like, uh,

run certain filter programs, or maybe use 'muchmore' to view your quips from an icon. Beware, though.. some uglinesses abound (eg: 'type' doesn't work with Quip for some reason).

New QuipScript command FINISHED lets you end a script at that point. Could be handy for certain goto statements.

1.20 Cows History

```
+------------------------------+
|       |
|     The Cheese Olfactory  |
|     Workshop's    |
|      History    |
|       |
+------------------------------+
```

It all started with the answering machine. I had to figure out a new outgoing message, because the old one was stale (you know how it is). In a sort of desperate attempt to come up with something, I said something like the following:

"Hello, and welcome to the Cheese Olfactory Workshop. We have all kinds of cheese available for your nasal needs. Our personal favorite is the limberger, which we've made especially potent with our new patented Cheese-Odorization formula. Our manager, Mr. Van Riper, is unable to come to the phone right now, but if you leave your name and number, he'll be happy to inundate you with cheese later."

Actually, the original message was far stranger than even that, but I can't remember it. I was just thinking about how funny such a name would sound. Anyway, Piouhgd called, and became entranced with the name as well, so much so that he suggested I use it for all the programs I write.

Pioughd's girlfriend (now wife) has a certain affection for cows. She has a cow-stool (for sitting), billions of cow magnets, cow seatcovers, cow utensils... I'm not entirely sure, but I think she even has a cow dress somewhere. She refuses to eat cow products (milk and steak are OUT), and all the rest. We, of course, got somewhat caught up in this, and started noticing weird things related to cows... eg. my ancestors came to the United States on a ship called The Spotted Cow. Citadel BBS, our favorite BBS program, had a 'feature' where if you typed 'm' at a certain time you'd get a 'Moo...' out of it. Funny stuff like that. We, of course, noticed quite by accident that the Cheese Olfactory Workshop has COW for it's initials, and we went totally nuts over it.

There is a sort of Cheese Olfactory Workshop related Pathos that is always sort of developing, mostly related to my own hyperactive imagination, excuses as to why this-and-that program doesn't work right... perhaps you'll read about it if you are wasting your time reading this <grin>.

Pathos

1.21 Pathos

```
+------------------------------+
|       |
| The Pathos Of COW |
|       |
+------------------------------+
```

Some of you may become angry at reading this useless tripe in the middle of a document file somewhere, but I figure it's just short enough and entertaining enough that perhaps you won't mind.

There are many reasons why COW takes so long to develop programs. Mostly, the problems have to do with our slave programmers. We've been trying to find more concientious slaves, but such things are always difficult to find.

Much earlier, we had a fairly good batch of slaves working for us, but one of them got uppity and started a rebellion. In the process of trying to quell the rebellion, many of the slaves were killed. It took us months to rebuild our stock of slave programmers, and we could only work the remaining ones so hard before their minds started to blow.

Then, a second rebellion occured, except it was much more subtle. One of our slave programmers tried to write something into the code that would open up a dimensional portal to another place, allowing them to free themselves. Unfortunately, they hadn't considered our specialized compilor trapping, and the portal opened up into a void, sucking fully half of our slaves into it. We managed to close it before the rest were taken. But, now we were left with so few programmers it was amazing we were able to get ANYTHING done.

Now, however, there has been a third rebellion, just before the writing of this latest version of Quip. Someone tried to hack into one of the US Government's nuclear weapons' sites, and train a missile at one of our headquarters. Unfortunately for them, they got the co-ordinates wrong, and bombed their own sites, meaning I had to program this puppy all by myself. That's why it's taken over a year for Quip 0.07 to be written (sigh). I'm still trying to get more slave programmers for COW, but good slave programmers are SO hard to find.

1.22 CLI & WorkBench Usage

```
+------------------------------+
|       |
|     CLI & Workbench Usage |
|       |
+------------------------------+
```

Quip has always offered CLI support, however, now Quip uses ReadArgs to handle the argument parsing. This offers some extra flexibility, although users of the older versions of Quip will notice that the FILES parameter now only accepts one filename. I apologize for this limitation, but it was easier to program this way, for the extra improvements added to the

program. If you REALLY want to process more files, try using the script language. Or you could always import from Mars.

Now, Quip supports icon use. If Quip doesn't have a need to print anything, it will not open a window. However, if it must print anything, Quip opens up it's own window to the WorkBench (or your own public screen if Shanghai'd) with a consoled clippable bit of text. Provided there are no errors, the text will be your quip. Regardless, to close the window, you need only press RETURN while the window is active.

The tooltypes you may use include all the various arguments used by the CLI, but you may not use any of the CLI's shortcuts. Trust me.

```
 VERSION
 COL
 DATA
 TABLE
 FORMAT .com line.
 FFORMAT
 PRE
 SUF
 MIDDLE .com line.
 FPRE
 FSUF
 FMIDDLE
 WHICH
 RANDOM .com line.
 DISPLAY .com line.
 NUM
 FILES
 REQ .com line.
 DELAY .com line.
 UPDATE .com line.
 ERROR .com line.
 RUN .com line.
```

1.23 VERSION

```
      +--------------------------------+
      |       |
      |       VERSION     |
      |       |
      +--------------------------------+
```

Just shows what version of Quip you're running (and displays a quip according to defaults). The version is not the release version, but the program version.. something to keep in mind. I'm not following StyleGuide conventions (totally) with regards to my selection of what version of Quip you're running.. my apologies.

1.24 COL

```
+------------------------------+
|       |
|   COL     |
|       |
+------------------------------+
```

This sets the column width of the file to be read as a script. It defaults to 80 characters, however, if you need more for some reason, this option lets you have more.

1.25 DATA

```
+------------------------------+
|       |
|        DATA     |
|       |
+------------------------------+
```

This lets you specify which file you want to read a quip from. It defaults to s:quip.dat.

1.26 TABLE

```
+------------------------------+
|       |
|       TABLE     |
|       |
+------------------------------+
```

This tells quip that you want the file specified by DATA to have a new tablefile generated for it.

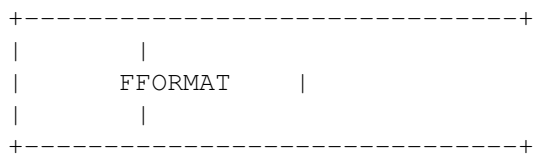
1.27 FORMAT .com line.

```
+------------------------------+
|       |
|      FORMAT     |
|       |
+------------------------------+
```

FORMAT takes a string holding text you want to appear between any PRE and the actual quip itself. The usual formatting is done to the contents of the FORMAT string.

The string specified will appear in the menu bar of the quip if REQ .com line. or DELAY .com line. was specified.

1.28 FFORMAT

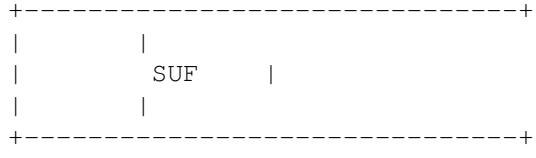


This lets you specify a filename holding the text you want to see appear between the PRE and quip. The text in the filename will be subject to formatting .

1.29 PRE

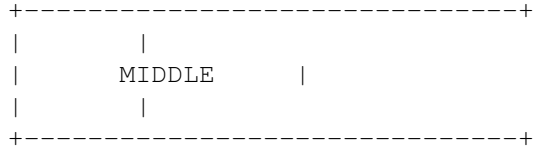
This specifies text that will come before the actual quip is written. Subject to formatting .

1.30 SUF



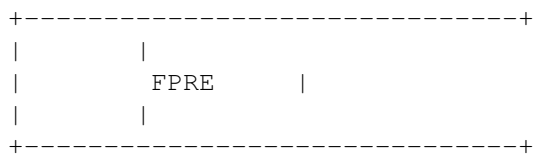
This specifies text that will be appended to whatever quip is generated. Subject to formatting .

1.31 MIDDLE .com line.



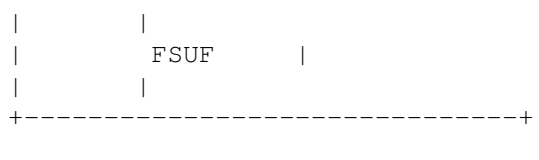
This specifies text that will appear between quips on stdio. Of course, this text won't appear if NUM isn't set to a value other than 1. This is potentially useful for filtering programs that someone might design, or perhaps some kind of arcane purpose I haven't thought of. And, as usual, the text undergoes formatting .

1.32 FPRE



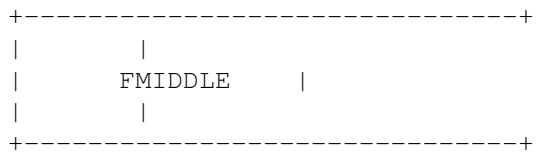
Specifies a file holding text to prefix before a quip. The text will undergo formatting .

1.33 FSUF



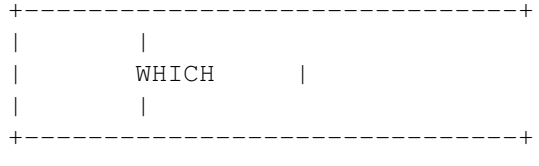
Specifies a file holding text to append to a quip. The text will undergo formatting .

1.34 FMIDDLE



This specifies a file holding text that will be treated as the MIDDLE (CLI/ICON option. Text undergoes formatting .

1.35 WHICH



This specifies a number which determines which quip will be generated (or from which quip a series of quips will be generated if the NUM argument is used).

1.36 RANDOM .com line.

```
+------------------------------+
|       |
|       RANDOM    |
|       |
+------------------------------+
```

This is a switch telling Quip that you want your quips generated randomly. If your DATA file doesn't have an associated TABLE file, Quip can now generate random quips, but cannot tell you which quip it has picked up. Therefore, if you use FORMAT .com line. , and you use the {quip} variable, you will get a '0' instead of the expected result <grin>.

1.37 DISPLAY .com line.

```
+------------------------------+
|       |
|       DISPLAY     |
|       |
+------------------------------+
```

This is a switch telling Quip that you want quips being generated by the FILES parameter to also be sent to stdio (or a window, for the Icon users) so you can see them. Each quip will appear exactly as they appear in the file.

1.38 NUM

```
+------------------------------+
|       |
|        NUM    |
|       |
+------------------------------+
```

This specifies how many quips you want to make with this invocation of Quip. It defaults to 1 quip. It's especially useful for Citadel sysops who want to create 60 quips for their BANNER, LONOTICE, NOTICE, or NOCHAT messages (when using the FILES parameter).

1.39 FILES

```
+------------------------------+
|       |
|       FILES     |
|       |
+------------------------------+
```

This specifies a set of filesnames to create with quips in them (mostly

useful for Citadel sysops). The filename will have its name, plus a number appended to the end of it. So, if you generate two quips (via NUM), and the filename you specify is 'BANNER', you get:

```
BANNER.0
BANNER.1
```

Very useful for Citadel sysops.. perhaps less so for others. Still, you can redirect stdio output if you really wanted to create only one quip with a specific filename (eg. Quip >myfilename.txt).

1.40 REQ .com line.

```
+------------------------------+
|       |
|        REQ    |
|       |
+------------------------------+
```

This tells Quip that you want it to use Nico François' reqtools.library to produce a very nice looking quip on the screen.

1.41 DELAY .com line.

```
+------------------------------+
|       |
|       DELAY     |
|       |
+------------------------------+
```

The number coming after this option tells Quip that you want to use Nico François' reqtools.library to produce a nice looking quip (as with the REQ .com line. option), but you want it to get rid of the quip after that number of tens of seconds have passed.

Therefore, if you want the quip to be displayed for no more than 10 seconds, you would use 'DELAY=100'. Of course, the user can type a RETURN to get rid of the quip before the ten seconds are up.

1.42 UPDATE .com line.

```
+------------------------------+
|       |
|       UPDATE    |
|       |
+------------------------------+
```

The number after this option tells Quip to update the tablefile from that number quip in the datafile you specify. If you specify '0', it'll assume you mean the last known quip in the tablefile, and will add the new ones to

the tablefile. This command could save you a LOT of time in updating your tablefiles, or fixing skewed tablefiles. I had wanted to put this in Quip Four , but found it hard to program. I guess I had programmer's block at the time. No.. wait, it was that one slave programmer...

1.43 ERROR .com line.

```
+------------------------------+
|       |
|        ERROR    |
|       |
+------------------------------+
```

With this option, you specify what file you want error messages to be sent to. Sick users can use CON:, if they really want to (but the message will disappear as quickly as it was made, and it'll always create that CON: window.. really not a nice option). Some might want to use NIL:.

1.44 RUN .com line.

```
+------------------------------+
|       |
|        RUN    |
|       |
+------------------------------+
```

This option will run a command, passing 'env:quip.tmp' (a file Quip will generate holding a quip in it) as an argument. This can be useful for such things as 'quip run more' or the like. Quip has no way of finding out the error code of the returned command, but it may tell you something via stdout. If used from an icon, a window is created before running the command, and can be removed by tapping <RETURN> in the window.

EG: 'quip run more' will generate a quip, put the quip in 'env:quip.tmp', then execute a 'more env:quip.tmp' command.

This option is not too reliable, so always test it first.

1.45 QuipScript

```
+------------------------------+
|       |
|   QuipScripting   |
|       |
+------------------------------+
```

That's right, folks, Quip can read a script file in its very own language (a language much easier to master than BASIC, C, ARexx, or darn near anything else, because it doesn't have the greatest vocabulary in the world). The first line in a script is ignored, so always start your scripts with some

```
comment at the beginning.  Don't worry, you can put comments darn-near
anywhere.. just be careful not to put them within lists, or on the same
line as an option.  It would also be safer not to put comments on the same
line as a keyword (although you can usually get away with it.. it's only
dangerous if your comments happen to collide with any of the other
keywords that exists.. eg: #IN_FILE  "I think a good PREFIX file..."

NOTE: Quip defaults to running a script called "S:Quip.script".  If it
can't find this script, it'll run as normal.  If it CAN find it, it'll run
that script.  If you specify a script with the SCRIPT option, it'll
override the default script.  If you specify a script file called "NIL:",
it will nullify the default script.

Therefore, if you have a file "S:Quip.script", but you want to run Quip
without running any scripts (including the default script), try:

Quip script NIL:

(all caps on the 'NIL:').

Also, QuipScript files default to using "s:quip.dat" as the datafile, but
if you specify a different datafile while invoking it, THAT datafile will
be used instead... therefore "quip script boo.script data cookie.dat" will
use the datafile 'cookie.dat' instead of 's:quip.dat', provided
'boo.script' doesn't have a #IN_FILE keyword used.

All keywords in QuipScript are in all caps, and start with a '#' character
on the beginning of the line, as in:

#KEYWORD

Options for the keyword appear on the next line, are usually
case-sensitive, and are preceded with a space:

#KEYWORD
 option

All lines in QuipScript may only be 80 characters long, unless the  COL
command-line option is used to change it.  I cannot imagine why anyone
would want to use more, but some people are strange <grin>.  If it's really
a problem, try ASSIGNing to paths, or using environment variables.

Here is a listing of all the various keywords that exist in QuipScript:

 QuipScript Lists
 IN_FILE
 OUT_FILE
 FORMAT
 PREFIX
 SUFFIX
 MIDDLE
 HOWMANY
 RANDOM
 SERIAL
 WHICHQUIP
 MAKETABLE
 DISPLAY
```

```
 MAKE_QUIPS
 SCRIPT
 GOTO
 LABEL
 CHANCE
 REQ
 DELAY
 UPDATE
 RESET
 ERROR
 OTHERECHO
 RUN
 FINISHED
```

1.46 QuipScript Lists

```
Lists are used when you want one selection chosen from a range of values.
They are terminated by a '#' character appearing at the beginning of a
line (I generally put a '#END' at the end, but it's not looking for the
work 'END', just the '#' character):

#KEYWORD_LIST
 option1
 option2
 option3
#END

There are only two keywords that indicate lists...  RND  and  SEQ .

 RND
 SEQ
```

1.47 RND

```
If 'RND' appears with the keyword, and a list follows, the script will
randomly select one of the options in the list:

#RND KEYWORD
 option1
 option2
 option3
#END

In the above example, option1, option2, or option3 will be selected
randomly and used as a the option for 'KEYWORD'.  The only keywords that
allow the use of the RND keyword with them are:

#RND  IN_FILE
#RND  OUT_FILE
#RND  FORMAT   (also FILE_FORMAT)
#RND  PREFIX   (also FILE_PREFIX)
#RND  SUFFIX   (also FILE_SUFFIX)
```

```
#RND  MIDDLE   (also FILE_MIDDLE)
#RND  SCRIPT
#RND  GOTO
#RND  CHANCE
#RND  ERROR
#RND  RUN
```

1.48 SEQ

If 'SEQ' appears with the keyword, and a list follows, the script will sequentially select one of the options in the list:

```
#SEQ KEYWORD
*option1
 option2
 option3
#END
```

In the above example, option1 will be chosen, and the '*' will move so it's in front of option2. The '*' character tells QuipScript which option to select. If the '*' character is on the bottom option, it'll be set to the top option (from option3 to option1 in the above example). If you fail to put the '*' in, QuipScript will put it on the top option, and will also select the top option for the keyword.

The following keywords can be used with the SEQ modifier:

```
#SEQ  IN_FILE
#SEQ  OUT_FILE
#SEQ  FORMAT   (also FILE_FORMAT)
#SEQ  PREFIX   (also FILE_PREFIX)
#SEQ  SUFFIX   (also FILE_SUFFIX)
#SEQ  MIDDLE   (also FILE_MIDDLE)
#SEQ  SCRIPT
#SEQ  GOTO
#SEQ  CHANCE
#SEQ  ERROR
#SEQ  RUN
```

1.49 IN_FILE

IN_FILE tells QuipScript what Quip datafile you wish to grab quips from. If you forget to specify this keyword, QuipScript will assume you want S:Quip.dat.

```
#IN_FILE
 s:quip.dat
```

This tells QuipScript that 's:quip.dat' will be the file from which you'll grab quips. A little redundant, but you can do this.

1.50 OUT_FILE

```
OUT_FILE tells QuipScript what file you want to write quips out to.  It's
important to note, however, that this option is intended to work with
Sysops running a Citadel BBS system (although others might find it
useful).  The filename actually used is modified with an ending that has a
number (the first is .0, then .1, .2, .3, and so on up to HOWMANY
ANY"} number
of quips you want to generate [minus one]):

#OUT_FILE
 ram:banner.pre

This tells QuipScript to write quips to ram:banner.0, ram:banner.1, etc,
until HOWMANY quips have been created.
```

1.51 FORMAT

```
FORMAT lets you sandwhich text between a  PREFIX  and the quip.  Note that the
text entered here will undergo some  formatting  to certain conventions.

To specify a file from which to grab this text, use 'FILE_FORMAT'.

Note also that this string will appear in the menu bar of a quip if
 REQ  or  DELAY  is specified.

EXAMPLES:

#FORMAT
 Quip #{quip}:\n

Prints "Quip #2342:", and a new line, where the actual quip starts.  Well,
the number depends on which quip was actually grabbed.

#FILE_FORMAT
 s:quip.form

Looks in the file 's:quip.form' for the text to put between the PREFIX and quip


#FORMAT
 {user}'s personal file #{quip}:\n

Prints "Joseph's personal file #234:" and a new line.  Assuming, of
course, that the environment variable 'USER' was set to 'Joseph', and that
the 234th quip was selected.

#FORMAT
 {date}:\n

Prints '2-Aug-93:' and a new line, assuming that you ran the command on
the 2nd of August, 1993.
```

1.52 PREFIX

PREFIX specifies a filename that holds text that you want printed before each generated quip. This text will be printed before the FORMAT string, too, which may be useful. Note: the prefix will pass through the same formatting as FORMAT , but specifying {quip} will print '0'.

To specify a filename, use 'FILE_PREFIX'.

```
#PREFIX
 -------------------\n
```

The above text would appear just before the Quip, except '\n' would be changed to a new-line character.

```
#FILE_PREFIX
 s:prefix.txt
```

specifies 's:prefix.txt' as a file holding text you want to prefix as above.

1.53 SUFFIX

SUFFIX specifies a filename that holds text that you want printed after each generated quip. The same things regarding PREFIX apply to SUFFIX.

Don't forget about 'FILE_SUFFIX'.

```
#SUFFIX
 --------------------\n
```

The above text is appended to the quip, with an additional newline taking the place of '\n'.

```
#FILE_SUFFIX
 s:suffix.txt
```

Whatever's in 's:suffix.txt' will be sent after the actual quip is sent.

1.54 MIDDLE

MIDDLE lets you specify text to print to stdio between quips when HOWMANY is specified. It undergoes formatting , and you can specify a filename by using 'FILE_MIDDLE'.

```
#MIDDLE
 And now.. ANOTHER QUIP!\n
```

would print the above text between quips, replacing '\n' with a newline.

```
#FILE_MIDDLE
 s:middle
```

would look for the text to put between quips in the file 's:middle'.

1.55 HOWMANY

HOWMANY lets you decide how many quips to generate at a time. If sent to stdio (the console), each quip will be separated by a newline. If being sent to files (the OUT_FILE keyword), HOWMANY files will be created, starting from 0 to HOWMANY-1.

```
#HOWMANY
 60
```

This will create 60 quips. If OUT_FILE is set to 'ram:banner.pre', 60 files starting at ram:banner.0 and ending with ram:banner.59 will be created. Great for Citadel sysops.

1.56 RANDOM

RANDOM tells QuipScript that you want your quips to be chosen randomly. If you do not have a tablefile for the datafile you specified, RANDOM cannot display which quip it has picked up, therefore the FORMAT command should will display a '0' for the {quip} variable. It's strongly recommended you have previously used MAKETABLE to create a tablefile for RANDOM, or the randomness will be severely effected (longer quips will be less likely to be chosen, and the first quip is less likely to be chosen).

```
#RANDOM
```

No need for options.. this is a switch. NOTE: use of this keyword will cause QuipScript to look for a file called "env:rnd" for a random number seed. If it doesn't find the file, it'll create it, seeding the random number generator with the clock. If it DOES find it, it'll overwrite the first four bytes with a random value that it'll use next time. I've found that this approach dramatically improves the generation of random numbers. Future programs by myself will use this same approach to handling random numbers (same filename, everything) in order to avoid dotting the hard-drive (for floppy <yikes>) with billions of silly four-byte files. I ask that other programmers do the same thing.

1.57 SERIAL

SERIAL cancels a RANDOM keyword. It causes quips to be selected sequentially from the datafiles (reading which quip it's supposed to grab from the datafile itself, and writing the next quip value to search for after all the quips have been taken). It is the default method by which quips are selected.

```
#SERIAL
```

No need for options.. this is a switch.

1.58 WHICHQUIP

WHICHQUIP lets you decide which quip to start grabbing quips from. If used with RANDOM , the first Quip will be WHICHQUIP, but subsequent quips will be random. This trick may be useful to avoid changing the next quip value in the datafile (if all you want to do is read a particular quip, but not change the sequential order in which you want to read that quip). If SERIAL is in use, quips will start from WHICHQUIP, and continue until HOWMANY (or 1) quips are created, then the datafile will point to HOWMANY+WHICHQUIP quip. Confused yet?

```
#WHICHQUIP
 300
```

Find the 300th quip in the datafile and print it.

1.59 MAKETABLE

MAKETABLE tells QuipScript to make a tablefile for the datafile. Tablefiles let you have better randomized quips, and greatly speeds up the time it takes to find quips. If you edit a datafile, you'll probably have to UPDATE a new tablefile, as you'll change where the quips are actually located in the file <chuckle>. Depending on how huge your datafiles are, this keyword could take some time to be carried out... I wouldn't recommend making a new tablefile each time you want a quip <grin>.

```
#MAKETABLE
```

No options.. this is a switch.

1.60 DISPLAY

DISPLAY tells QuipScript to send the quips that are being generated to OUT_FILE to stdio as well. Normally, quips being sent to files are not displayed in order to speed the process up. However, if you want to see them as they are being generated, this option is available.

```
#DISPLAY
```

No options.. this is a switch.

1.61 MAKE_QUIPS

MAKE_QUIPS actually starts writing all the quips. It takes all the information previously entered, and generates a quip from it. This keyword MUST be used, if you want any quips to be generated. After using this keyword, you may use other keywords to modify the settings, and use this keyword again with the new modifications. Only RESET actually changes all the values to their defaults.

```
#MAKE_QUIPS

No options.. this is a switch.
```

1.62 SCRIPT

```
SCRIPT will temporarily quit reading the current script, and start reading
the file you specify, resuming the old script when the SCRIPTed file is
finished.  You may nest as many of these as you want, although memory
becomes a consideration <grin>.

#SCRIPT
 option.1

The above will quit scripting there, reading 'option.1' for more quip
stuff, then resume after finishing 'option.1'.  Be careful about using
this keyword, as putting it in the wrong place in a script could have
peculiar effects.  Also, beware of creating a loop (a SCRIPT that calls
itself, or calls a script that will call the calling script).  SCRIPT does
NOT guard against this, to allow for more flexibility in script handling.
Therefore, funky weirdnesses like:

# RND  SCRIPT
 option.1
 samefilename
 samefilename
 samefilename
#END

where 'samefilename' is the name of the file you're currently scripting,
could create interesting havoc.  And:

# SEQ  SCRIPT
*option.1
 samefilename
 samefilename
 samefilename
#END

would be ugly, indeed.  However, perhaps you might find a need for one of
these kinds of constructions.
```

1.63 GOTO

```
GOTO will cause the script to continue at the LABEL specified.

#GOTO
 option

The script will continue execution from LABEL 'option'.  Read  LABEL  for
much more information.
```

1.64 LABEL

LABEL doesn't really do much of anything, alone, but serves as a marking place for a script. The GOTO keyword will continue execution of the script from the LABEL keyword, provided they have matching options.

```
#LABEL
 option
```

If a "#GOTO <linefeed, space> option" were issued somewhere, program execution would continue from the above keyword. EG:

```
#LABEL
 option
#KEYWORD1
#KEYWORD2
#KEYWORD3
#GOTO
 option
```

would allow KEYWORD1, KEYWORD2, & KEYWORD3 to be executed forever, since the program would read them, get to the GOTO, leap up to the LABEL, and continue doing the KEYWORD commands until something harder stops them (like a truck, maybe).

It is advised to use the GOTO keyword with one of the List keywords, and even then, with GREAT care. Infinite loops are easy to create with the GOTO keyword.

1.65 CHANCE

CHANCE lets you randomly decide whether a certain command will be executed, depending on the number given to it. It will generate a random number between 0 and 999, and if it's number is less than your number, the following command will be executed. I chose '1000' instead of '100' so you could have more precision for your randomness. Think of it as a percentage with the decimal moved over one place (that is, 50 percent chance of having something executed is 500, 75% is 750, 25.7% is 257, etc).

```
#CHANCE
 230
#KEYWORD
(keyword options)
```

#KEYWORD has a 23% chance of being executed each time this script is run.

NOTE: you may use the RND or SEQ list modifiers with your keyword with this command.. CHANCE is smart enough to handle lists. That is:

```
#CHANCE
 500
#RND KEYWORD
 option1
```

```
 option2
 option3
#END
```

would give a 50% chance of having any one of those options randomly chosen for KEYWORD. Pretty darned flexible, eh?

1.66 REQ

REQ is a switch letting you print your quips to an information requester from Nico François' 'reqtools.library'. As long as the quip fits the requester, it's a very beautiful way of looking at your quips, but you cannot clip text from the requester <grin>.

```
#REQ
```

Tells QuipScript to show quips in a Requester. No options.. this is a switch.

1.67 DELAY

DELAY lets you use a reqtools.library requester, as REQ , but will only leave the requester up for a certain amount of time, before taking it down and moving on.

```
#DELAY
 50
```

Tells QuipScript to show the quips in a requester, but to take the requester down after 5 seconds.

1.68 UPDATE

UPDATE lets you update the tablefile for the datafile specified by IN_FILE . If you use a '0' for an option, the update will occur from the last known quip in the tablefile, adding the new quips in the datafile to the tablefile, allowing you to use all the new quips you may have accumulated.

```
#UPDATE
 643
```

This will look for the 643rd quip, and start updating the tablefile from that point in the tablefile, rather than starting at 0, as MAKETABLE does. If there is no 643rd quip (maybe only 20 quips are in the file), it'll default to the last known quip.

1.69 RESET

RESET sets all the various settings to their defaults. Normally, even after a MAKE_QUIPS command, prefix files, suffix files, outputfile name, random status.. everything remains intact for the next MAKE_QUIPS command issued. This allows you to reset everything to their original default values:

#RESET

This is switch.. no parameters are taken.

1.70 ERROR

ERROR causes all error messages (and other, perhaps undesirable stuff) to be sent to the filename specified. It defaults to a standard error port (if called from CLI/Shell) or stdout (if called from icon).

```
#ERROR
 NIL:
```

This will send all error messages to nowhere.

```
#ERROR
 ram:boo.txt
```

This will put all the error messages in a file called "ram:boo.txt", if any errors exist at all.

1.71 OTHERECHO

OTHER_ECHO is a toggle switch between sending '#' texts that have no keywords to Standard I/O, or whatever is specified by ERROR .

Default is standard i/o.

#OTHER_ECHO

No parameters.. this is a toggle switch.

1.72 RUN

RUN will run a command, passing 'env:quip.tmp' (a file Quip will generate holding a quip in it) as an argument. This can be useful for such things as 'quip run more' or the like. Quip has no way of finding out the error code of the returned command, but it may tell you something via stdout. If used from an icon, a window is created before running the command, and can be removed by tapping <RETURN> in the window.

```
#RUN
 more
```

Will generate the quip, put the quip in 'env:quip.tmp', then execute a 'more env:quip.tmp' command.

1.73 FINISHED

FINISHED causes the QuipScript to end at that exact point. I figured it might be a useful command when used with GOTO and LABEL commands.

```
#FINISHED
```

No options.. quits the script at that point.

1.74 TECHNOTES

```
      +------------------------------+
      |       |
      |  Technical Notes  |
      |       |
      +------------------------------+
```

For those of you out there who are curious, or want to consider some advanced stuff to make Quip sing well for you.

```
 DATAFILES
 SCRIPTFILES
 Random Number Generator
 Requester Library
```

1.75 DATAFILES

```
      +------------------------------+
      |       |
      |     DATAFILES   |
      |       |
      +------------------------------+
```

If you wish to create your own datafiles, please keep a few things in mind.

Datafiles start with about 15 or 12 characters designated for holding a number. This number is used by Quip to figure out which quip it's supposed to pick next. I chose to do this as an ASCII number rather than a binary number so you can change it manually if you want. But, if you don't give these 15 or so characters to Quip, it'll just gobble them up anyway, trampling on top of whatever information you might have had there in the first place.

After these characters, however, you can put anything you want in the datafile, but remember that all quips are delimited by '@' characters (something I'm thinking about changing in the future.. maybe to formfeed characters). Therefore, the actual quips themselves start the moment you put the first @ character in the file.

You may wish to consider writing scripts in this space: QuipScript quits when it sees the '@' character. But I wouldn't recommend writing anything big, because when Quip 'rolls over' (during sequential quips, if the next quip is larger than the number of quips available, Quip will go back to the first quip again), it'll have to read through all that information to find the first '@' character.

As you may have figured out already, each new quip is supposed to start with a '@' character. Otherwise, you may have whatever set of characters, in whatever combination you want, as you desire, with the exception of the 'NULL' character (ASCII 0). The NULL character will truncate your quip! I would have used it to delimit quips, except most people have a difficult time putting them in with a simple text editor <grin>.

Also, keep in mind that datafiles with tablefiles need to have a new tablefile create every time the datafile is edited. If you don't, your quips may be askewed, or ignored altogether. So, if you add new quips to a datafile, be sure to create a new tablefile for it.

Beware the write-protection bit... make sure that it is not set on ANYTHING having to do with Quip. I'm not sure if it's due to a bug in the OS, or if it's because my hard-drive uses OFS instead of FFS (I've been having trouble converting it over to the other format for some stupid reason), but Quip hasn't been able to handle write-protection errors properly... it thinks everything is fine!

Therefore, tablefiles can ge utterly destroyed by Quip if they are write-protected, and datafiles will not be updated if write-protected (during any options that call for Quip to write to the file, as in UPDATE and TABLE commands).

1.76 SCRIPTFILES

When making scriptfiles, your comments may appear pretty well anywhere you want, except on the same lines as a parameter. Be careful, however, about writing comments in ALL CAPS. You stand a chance of writing a command by accident.

You may have noticed that lines that are preceded with a '#', but don't have a proper command after them, are printed to stdio (or to a window, if invoked from an Icon). This can be used to help trace where your scripts are, or perhaps as a kind of extra 'prefix/suffix' "feature". The '#'

itself is not displayed.

Be careful when creating your scriptfiles. QuipScript really isn't intended as a full-blown language, so it doesn't do any kind of checking to speak of. If you fail to end a RND or SEQ list with a '#' line, weird things can happen in your scripts. THIS IS EASY TO FORGET! While testing this, I've forgotten the thing myself many times. There is really no way I can test for it AND keep the flexibility, so I'll simply assume that good programmers will read the directions <grinning stupidly, in full knowlege that I don't read the directions either>.

Also, there are times when the '@' character is not looked for! While processing lists (!), and during #CHANCE command handling, and while searching for a matching # LABEL from # GOTO (!), the '@' is not looked for, so you can use that character within your scripts as part of the options. BE CAREFUL... if you decide to write a script within a datafile , make sure your script is working PERFECTLY before actually committing to it, or you could wind up with a script that hunts through all 10 megs of data before coming to the realization that something is quite wrong <grin>.

You can do some nifty tricks with QuipScript, if you're clever (and you have the inclination). You can set up a label at the end of the file (say, #LABEL<newline> END), then set up a # CHANCE statement, followed by a #GOTO<newline> END statement, to create a percentage chance of the script being quit! You can also set up crude 'for/next' loops using SEQ statements looped with GOTO statements. Play around with QuipScripting.. you can always press Control-C to get out of Quip at any time.

1.77 Random Number Generator

```
+-------------------------------+
|       |
|   Random Number Generation  |
|       |
+-------------------------------+
```

Quip runs a nifty little routine prior to doing anything. This routine looks for a file called 'rnd' in the ENV: directory. If it doesn't find the file, it'll go into the system clock (through proper DOS channels). The value it's looking for will be a seed for the program to use for generating random numbers. When it gets the seed, it then creates a new one, putting it in ENV:rnd.

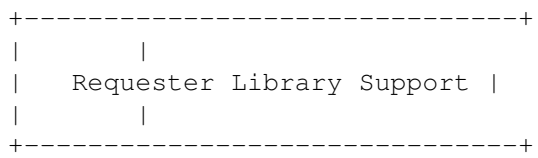
The idea is to avoid having billions of different little files running around with four-byte values used to seed the random number generator, while still giving you really good random numbers. The reason why using the clock is out (at least, to me), is because even over a period of an hour, the numbers are not very random. Since Quip is a short-running program, calling it each time and borrowing the seed from the clock doesn't really yeild a very random number.. especially if you're running a BBS and you're calling it each time a user logs in. The reason I use the ENV: directory instead of tracking quip's own directory is because I figure one centralized place for the random number file, useable by other programs (maybe even by libraries in the future) is A Good Thing. Especially since I intend to use this routine in other programs that

require random numbers.

There are, of course, drawbacks. Firstly, it's just that little bit more time having to go through the filesystem to find a seed. Although I only have to do it once, it might be considered annoying. Secondly, to quote Ulrich Kaufman (author of the XEM standard): "It is against the philosophy of ENV: to use binary data." Text would be nicer, but it's easier to work with four bytes than twelve. Also, unless you run another little program that creates a value in an ENVARC:rnd file every time you boot up, you stand a chance of either having the same random numbers popping up (in the case of an ENVARC:rnd file existing and being written to ENV: each time it boots, without being change itself), or the system will always have to rely on the clock for the first generated random number. Not TOO bad, but I could hope for better <grin>.

All-in-all, though, I think this method works best. Just be aware, you'll be four-bytes poorer in your ENV: directory after running Quip <grin>.

1.78 Requester Library



Quip now takes advantage of Nico François' reqtools.library. Please keep in mind that the reqtools.library's Copyright (c) is owned by Nico François, who has graciously allowed the rest of us lazy people to write nice programs using this bit of software.

Inasmuch as it's wonderful, you should probably try to keep quips that go into the requester somewhat within the requester's size limitations. Quips that are larger than 78 columns (I think) tend to be a little troublesome, and those with billions of lines probably won't look very good, either. Otherwise, it's a very nice, pretty way of looking at your quips.

Another minor note: as much as I would like to avoid busy-waiting, to some extent this program has to busy-wait through a DELAYed requester, in order to tell when the time is up, or when the requester is done. Fortunately, however, I am using a Delay(5) in there, so it'll only check every 10th of a second for a value. So.. while it IS busy-waiting, it's a kind of low-impendence busy-wait, so it shouldn't be as bad. I could have made the value larger (Delay(50)), but 10th/seconds are a good round measure, fast enough to keep someone from waiting for the program to finish, but slow enough to keep resources from going nuts in the multi-tasking.

Also, keep in mind that a REQ requester is really a DELAY ed requester that takes 90000 seconds to finish. Therefore, it suffers from the same busy-wait problem. I did this to save a little extra room in the programming. My apologies if this is ugly.

1.79 CREDITS

```
      +-------------------------------+
      |       |
      |       Credits   |
      |       |
      +-------------------------------+
```

Although I, JEVR3, did all the programming, I couldn't have done everything
without the help of some very special guin.. er, people, who beta-tested
this version of Quip for me (perhaps uncovering stuff I missed). Some of
these guys were also a great source of moral-strength <grin>. I consider
them a part of the Cheese Olfactory Workshop, for their efforts...

Kevin Thomas - Death Colas, MS-DOS help (in earlier
versions), The Static Quip, and various ideas.

Gay Crumley - Helped out with LD calls in the US, letting me
set up the COW room for discussing Quip and other
assorted things (It's a Citadel Thing).

Lars Magnus Nordeide - Author of WiseCrack, a program that sorta does
what Quip does (although, not with the same
amount of steroids). I took a couple of his
ideas and put them in Quip, although my code
itself is certainly mine (nothing beyond an
idea was copyed from Mr. Nordeide).

Stephan Sürken - Author of Text2Guide, which was used to help
make these documents (only AFTER I had already
made the AmigaGuide-style by hand <sigh>).
Any funkyness you see in this document is there
in order to accomodate some of Text2Guide's fancies,
and to help accomodate AmigaGuide's funkyness.
One of these days, I really should send Mr. Sürken
some suggestions for the program (if he cares).

rlang@firebird.newcastle.edu.au
- One of the beta-testers for Quip 0.07.

bhogsett@bix.com - Found a bug I would never have found in my
Quip.guide documentation. Also one of the
beta-testers for Quip 0.07.

ralle@oberon.dinoco.de - One of the beta-testers for Quip 0.07.

1.80 FORMATTING

Prefix, Suffix, Middle, and Format texts/files are run through a
formatting field that may change certain things around, allowing a measure
of flexibility that might be useful.

The following characters in the format strings insert the following
things:

```
{day}  = Day of week, as "Monday" or "Tuesday"
{date} = Date, in dd-mmm-yy format (AmigaDos format)
{idate}= International date, yy-mm-dd
{adate}= American date, mm-dd-yy
{cdate}= Canadian date, dd-mm-yy
{time} = Time, in hh:mm:ss format (sorry, only 24 hour format)
{quip} = The number of the quip that was generated (only works with
         FORMAT/FILE_FORMAT commands, otherwise yeilds '0')
{[any]}= Text from whatever environment variable you specify.
         Great for {user}, {system}, and so on.  Looks at local
   environment variables first, then global ones.  NOTE: make sure you
   set the variable with Commodore's SET or SETENV rather than, say,
   csh's 'set' command, as QuipScript is using AmigaDos calls to find
   this value.
{{}    = { character

%%     = % character.. a quirk of using RawDoFormat()

\n     = New line
\r     = Return (same line)
\t     = Tab character
\f     = Formfeed (clear screen)
\g     = Bell characer (ASCII 7)
\e     = ESC (ASCII 27)
\     = backlash
```

Other '\' values will be weeded out. Environment variables not set will be ignored. Be careful not to use RawDoFormat characters while making prompts (eg. %s, %ld, %c, etc). RawDoFormat is used to generate the number of the quip that is being generated (parameter {quip}). Therefore, if you enter a RawDoFormat character sequence, you may create bizarre, unpredictable formats for your quips. Use the double % to produce a %.

I apologize for not providing for binary values, however if you have a really nice editor, binary values inserted in the line will not be filtered <grin>.

EXAMPLES:

```
 Quip #{quip}:\n
```

Prints "Quip #2342:", and a new line, where the actual quip starts. Well, the number depends on which quip was actually grabbed. Of course, if this string is being used with anything other than FORMAT, you get "Quip #0:" and a new line.

```
 {user}'s personal file #{quip}:\n
```

Prints "Joseph's personal file #234:" and a new line. Assuming, of course, that the environment variable 'USER' was set to 'Joseph', and that the 234th quip was selected, and it's being used in the FORMAT command (otherwise, it'll say "Joseph's personal file #0:" and a new line).

```
 {date}:\n
```

Prints '2-Aug-93:' and a new line, assuming that you ran the command on

```
the 2nd of August, 1993.
```