ChatBox 1.129 Alpha

COLLABORATORS

	TITLE : ChatBox 1.129 Alpha	

ACTION	*NAME*	*DATE*	*SIGNATURE*
WRITTEN BY		February 12, 2023	

REVISION HISTORY

NUMBER	DATE	DESCRIPTION	NAME

Contents

Chapter 1

ChatBox 1.129 Alpha

1.1 ChatBox - The new world of IRC

1.2 Distribution

As of ChatBox v1.145, distribution has moved to freely distributable so long as the original archive found on Aminet is intact. No files may be added, or removed. Clear?

Also, I would appreciate registration with the following information:

Full Name, Address, State/County/Province, and Country, E-Mail address, and IRC nickname most often used on #Amiga.

If you do not register with me, I will NOT respond to E-Mail, nor will I take suggestions.

E-Mail registration to: jweb@primenet.com Subject: CB REG

You will be placed on the ChatBox mailing list...

1.3 Overview - Copyright, implementation, goals

ChatBox was started as a personal project to overcome the limitations that I experienced using the IRC clients currently available for the Amiga. And I certainly did not like the UNIX version.

ChatBox is Copyright (C) 1994-1996 by JDW Development. This version is currently an evaluation version, so, as with all software, use it at your own risk. JDW Development will not be held liable for any damages resulting during the use of this software. It is provided 'as-is' with no warranty, neither expressed nor implied.

ChatBox alpha and beta versions may NOT be distributed.

The command set of ChatBox is implemented as the common 'slash' command method.

My goal in designing ChatBox is to allow complete operator control. ChatBox will be designed in such a way that is simple for new users, yet has the power for seasoned users.

I didn't over 'idiot-proof' ChatBox, so there are no cute requesters for every little thing you wish to do.

Simplicity is the design goal. *I* do not like GUI's to do such simple tasks as are required on IRC.

This in mind, don't ask for 'DOCKS', 'REQUESTERS'*, or 'GADGETS' unless they ADD to the functionality of ChatBox's communication facilities. For example, no gadgets for 'KICK', or 'BAN'... You can ADD smart kicks and bans via ARexx, I will therefore leave that up to the user.

Currently under construction:

/DCC SEND	Writing custom routines
Prefs Utility	Not yet completed in evaluation version

Current known bugs/quirks:

Refresh errors	Some artifacting
Crashes with 'on-the-fly' Prefs adjustments	On History & Scroll back only

Send bug reports to: jweb@primenet.com Subject: CB BUG
If you do not set the subject to "CB BUG", it WILL be overlooked. Bug reports

are automatically processed by a program...

1.4 Acknowledgements, and thanks

Special thanks goes to:

Jim Huls	Faithful alpha tester, without whom 100% of the bugs would not have been found
Mike Latinovich	Super "oof!'r" Deluxe! For his suggestions, and alpha testing
Paul Reece	For his special interest in playing with the 'Friends' feature
RobR	For alpha testing and encouragement
Jonathon Potter	For refreshing my memory about refreshing
Osma "Tau" Ahvenlampi	for use of his DCC clients. And for MOST of the DCC.CHAT code that ChatBox's DCC Chat client uses.
Christoper Aldi	for designing and allowing me to use ClassAct for ChatBox's preferences facility.
Josef Faulkner	For the excellent ARexx scripts he wrote that really put CB's ARexx port to the test.
Sami Itkonen	For his many suggestions, and calm outlook.

And many others whose names escape me...

1.5 Requirements, and setup

What is required to use ChatBox?

You'll need AmiTCP 3.0b2+, minimum of 1-meg, 2.04+ of the Amiga OS, and a little patience while ChatBox is still under development. If you wish to use DCC, you'll need to assign a path to your DCC.TYPE.AS225 files to the device: "DCC:", i.e.: Assign DCC: Work:Utilties/DCC

How do I install ChatBox?

There is no REAL installation necessary, other than putting AmiTCP together. The only other task necessary is setting some ENV: variables.

The following Environment Variables can be set (and stored in ENVARC: if you wish to make them permanent, and available upon reboot):

DCCPATH Path where you'd like dcc's to go.

```
REALNAME   Your real full name.
IRCSERVER  Your default server.
USERNAME   Your login name (don't fake this)
IRCNAME    Your default nickname
HOSTNAME   Your host name (not your dialups)
NODENAME   Your domain (ie: primenet.com)
DCCDIR     Path where the AS225 DCC files are located
PUBSCREEN  Public screen name, if it doesn't exist, it will be created.
TEXTFONT   Font for text window, names list, and gadgets
SCREENFONT Default font for custom public screen
SETUPFILE  Alias file to load on startup...
PREFSFILE  Prefsfile to load, useful for project icons
TRIGGER    Trigger character for Friends commands. (default is '%')

NOTE:  Do not enclose tooltype settings in QUOTES!!!

    The above are also options on the command line, or can be specified as
ToolTypes in ChatBox's Icon or in Project Icons.

    ChatBox's template is:

"USERNAME/K,HOSTNAME/K,NODENAME/K,IRCNAME/K,IRCSERVER/K,REALNAME/K,DCCPATH/K,
 DCCDIR/K,PUBSCREEN/K,TEXTFONT/K,SCREENFONT/K,SETUPFILE/K"

    Note that HOSTNAME must be specified, in one of the three ways (on the
commandline, as a tooltype, or in ENV:).  The ENVironment variables are the
fallback in case options are not specified as tooltypes or on the command
line.  Addendum:  HOSTNAME may also be specified in Prefs.

Note that the preferences facility does not override the Above...  ToolTypes
and CLI options have priority.

NOTE:  Most tooltypes and CLI options will be removed in future versions,
       get used to using Prefs...
```

1.6 ChatBox's Preferences File

```
    As of v1.148, the prefs file is no longer a binary file.  Rather, it has
gone to a text based tagfile.

    Included in this archive is a program called "CBCvtPrefs", as you may have
guessed, this program converts the old binary prefs file to the new ASCII file
format.

    The default prefs file name is 'cb.prefs', so, to convert 'cb.prefs' use:

CBCvtPrefs cb.prefs cbnew.prefs

    After you have done this, you can delete the old prefs file, and replace it
with the new one (after renaming it to 'cb.prefs').

    Here are all of the supported tags, and their expected arguments:

They are grouped respective to the layout of the ChatBox prefs editor.
```

```
    NOTE: <string> can be any string of characters.
          <decimal#> is an ascii representation of a decimal #

User Prefs:

    REALNAME  = <string>
    LOGINNAME = <login>
    PASSWORD  = <password>
    DEFNICK   = <nickname>
    USERINFO  = <string>
    HOSTNAME  = <hostaddress>
    DOMAIN    = <localaddress>


Server Prefs:

    DEFSERVER = <servername/IP>
    AUTOJOIN  = <channellist>


Paths Prefs:

    DCCPATH   = <dcc_clients>
    DCCDIR    = <dcc_receiveto>
    ALIASFILE = <alias_file>
    STARTUP   = <startup_file>
    DEFPATH   = <default_path>
    SOUNDFILE = <8SVX_sound>


Display Prefs:

    SCREENMODEID = <32-Bit_ModeID>  (in hexadecimal, without leading '0x')
    SCREENTYPE   = <screen_type>    (below)
        Screen Types:
            DEFAULT
            PRIVATE
            PUBLIC

    PUBSCREEN    = <pubscreen_name>
    SCREENWIDTH  = <decimal#>
    SCREENHEIGHT = <decimal#>
    SCREENDEPTH  = <decimal#>
    OVERSCAN     = <overscan_type>
        Overscan Types:
            TEXT
            STANDARD
            MAX
            VIDEO

    AUTOSCROLL                      (no arguments required, this is a flag)
    TEXTFONT     = <font/size>
    SCREENFONT   = <font/size>
    SPACING      = <decimal#>       (# of pixel spaces extra between text lines)
    SCROLLBACK   = <decimal#>       (# of kilobytes for scrollback buffer)
    HISTORYLINES = <decimal#>       (# of history lines)
    WINDOWTOP    = <decimal#>
```

```
    WINDOWLEFT   = <decimal#>
    WINDOWWIDTH  = <decimal#>
    WINDOWHEIGHT = <decimal#>


Misc Prefs:

    KICKMSGTYPE   = <kick_type>
        KickMsg Types:
            DEFAULT
            RANDOM
            AREXX

    DEFKICKMSG    = <string>
    KICKFILE      = <kick_file>
    DEFQUITMSG    = <string>
    USEWHOISREQ                     (no arguments required)
    SIGNOFFISQUIT                   (no arguments required)

    TRIGGER       = <character>     (character for triggering friends commands)

Prefs items in the Prefs menu:

    SHOWISON                        (no arguments for any of these)
    SHOWCTCP
    SKIPMOTD
    BEEPONMSG
    BEEPONBEEP
    BEEPONAWAY
    INTERNALBEEP
    EXTERNALBEEP
    BEEPSCRFLASH
```

1.7 ChatBox's preferences facility

```
                In this section preferences will be described by it's individual ↩
                   components.


Prefs Menu

    Edit...           Brings up the preferences editor window.
    Skip MOTD         Skip MOTD server message (message of the day)
    Show CTCP         Show CTCP's when they are sent.
    Show ISON         Show ISON server replies. (see:
                Silent use of /NOTIFY
                )
    Beep              Beep options
        Private Msgs  Beep on incoming messages
        On BEEP       Beep on incoming BEEP control codes
        When Away     Beep only when away, based on former two options
    Internal Beep     Not implemented
    External 8SVX     Not implemented
    Flash Screen      Flash screen on BEEP
```

Preferences Editor

The preference editor is separated into four pages.

User page

From this page you can modify information that ChatBox may share with other users via various CTCP's. Password is not used, and is completely ignored in this version of ChatBox.

Real Name	Your real name, or any info that you'd like to appear in WHOIS
Login Name	The login name you use, significant when using IdentD.
Password	* unused *
Def. Nick	The nickname you'd like to begin each session with
UserInfo	Information you'd like to appear on a USERINFO CTCP.
Host Name	Your host's named address
Domain	Your named domain, if not the same as Host.

Defaults

Real Name:	ChatBox IRC (C) 1994-1996 JDW Developments
Login Name:	Unknown
Def. Nick:	same as Login Name
UserInfo:	ChatBox version string
Host Name:	taken from ENV:HOSTNAME
Domain:	taken from ENV:HOSTNAME

Server Page

You may specify the default server to use from here, as well as specifying all the channels you wish to automatically join on connection.

There are no defaults.

The listviewbrowser is for the server list you have specified to select from. You must have the file "cb.serv" in the home directory for this to be active.

Notes on 'Auto Join': If more than one channel is specified, separate each channel name with a comma and NO spaces. You may specify up to 10 channels to join, this is the current IRC server limit.

Paths Page

Clients	The full path to your DCC clients *
Receive To	The full path to the directory where DCC's receives should go *
Alias File	The full path and filename of your default alias file *
Startup	Not implemented *
Def. Path	Not implemented *
Sound File	Not implemented *

Display Page

Screen/Window Prefs

Screen Mode	Allows you to select a screen mode for 'owned' screens *
Window Font	Font to use for gadgets, and text in the window *

```
        Screen Font  Screen font, for title bar *

    Public Screen  The name of the public screen to open, or to open on.
    Screen Types
        Public Screen  Use an 'owned' public screen, if named screen (Above)
                       does no exist, else open on existing named screen
        Custom Screen  Use a privately owned screen
              Default  Use the default public screen

       Spacing  Additional spacing between text lines in channel window
    Buffer (K)  Scrollback buffer size in Kilobytes
       History  Number of lines in History

Misc Page

    Kick Prefs
        Use Default Message  Use the default kick message on all kicks if one
                             is not specified.
        Random From File     Use a kick file, randomly selects a line.
                             See also:
                Kick File
                 for format
        ARexx Script         Execute an ARexx Script for the kick.
                             (valid only from WHOIS Requester) The format used
                             for calling the script is:
                             script.rexx <nick> <address> <banmask>

        Kick File            Random Kick file, or ARexx script file to use,
                             depending on selection above. *

        Default Kick         Default kick message if one is not specified

        NOTE:  If you'd like CB to use a random kick message from the /KICK
               command, and you don't have 'Random from file' set in prefs,
               simply use '$' as the 'kick reason', and CB will use the
               random file. (An alias perhaps?  "/alias rkick kick $0 $")

        Use Whois Requester  Open WHOIS Requester on double clicks in names
                             list.

        /SignOff = /Quit     With this option set (checked) /SIGNOFF will not
                             only send a 'QUIT' to the server, but it will
                             also close down the ChatBox Session.  If not
                             checked, then it will merely QUIT the server
                             and close the connection, leaving the Session
                             open.

        Default Quit         Default message to use for quitting, if one is
                             not specified.

* NOTE:  All prefs items above with a '*' next to them allow the use of an
         ASL requester of the appropriate type for option selection.  Please
         note that the ASL Library LOCKS input to the window, so that it
         ignores incoming messages (i.e. from the timer.device, dcc clients,
         and the connection).  If you keep the ASL requester open for too long
         and you are connected to a server, you WILL ping-timeout.  Keep
         this in mind!  It is NOT a bug.
```

1.8 ChatBox's Random kick message file

```
If you wish to use random kick messages, you'll have to create a kick file.
The format is quite simple.  Just make some kick messages up, any amount,
and put them in a standard text file.  Then, at the beginning of the file
put a '#numlines', so that ChatBox will know how many lines there are.

The '#xx' MUST be the first line of the file...

An example:

|------------> Cut Here <-----------|
#4
Get outta here!
Go away, you bother me!
Bruised?
Did I say I'd OP you?
|-----------------------------------|

Easy enough?
```

1.9 ChatBox's Menu Items

```
                    Not all of ChatBox's menus are active, some perform minimal ↩
                       actions, some
are completely active.  Please read this section carefully so that you
understand which, and why.

Project Menu

    New Window  Open a new ChatBox window, same as issuing /NEWWIN command
    Save        Save the current project (prefs file, server list)
    Save As...  Save the current project under a new name
    Open        Not active...  Subsequent releases will have this active
    Quit        Gee, what's this for?


Channel Menu

    Join...           Simply puts the command /JOIN in the text entry gadget
    Leave             Leave the current channel (BUG, actually acts like QUIT)


*Channel Mode operations: (If item is checked, then that mode is set)
    Secret            Channel does not show up on LIST or WHOIS
    Private           Channel is Private
    Moderated         Only VOICED or OPPED users may speak here
    Invite Only       Can only join if invited
    No Messaging      Cannot send message to channel from outside the channel
    Topic Protection  Topic can only be changed by those with OPs.
```

```
    Op...             Not active...
    Ban...            ...
    Voice...          ...
    Keyword...        This and the 3 above will be removed, most likely
    Banlist           Get channel banlist


Scripts   (this menu will probably be removed entirely.)

    Execute   Start an ARexx script (why not just read the section on AREXX?)
    End       not active
    Stop All  not active


Prefs Menu

    See section:
                ChatBox's Preferences Facility
                Nameslist

    All of the items under this menu perform the action on the selected name in
    the nameslist.


Scrollback

    Save...             Save scrollback buffer
        All             Save ALL text
        Current to End  Save from top of current page to the end of buffer
        Top to Current  Save from top of buffer to bottom of current page
        Visible         Save current page (all visible lines)
    Save As...
        All             Same as above...
        Current to End  ...
        Top to Current  ...
        Visible         ...
    Clear               Clear Scrollback buffer...
```

1.10 ChatBox's ARexx Port

```
                    The Quick & Dirty:  ChatBox's port is named 'ChatBox', and ' ↩
                       ChatBox.x' for
subsequent invocations of ChatBox (where 'x' is a number).

    The rest:

Currently supported Commands:


                WAIT
                 - wait for text from ChatBox's ARexx Port

                CMD
                 - Send a ChatBox command as if it were entered manually
```

```
                RAW
                 - Send a RAW IRC command

                MATCH
                 - Match a "friend's" address

                NICKONCH
                 - Check if a nickname is currently joined in a channel

                FINDNICK
                 - Find a nick and return the channels where it is joined

                ISOP
                 - Find out if a given <nick> has VOICE or OPs

                PATTERN
                 - Check if a given pattern matches a given string.

                NAMES
                 - Get ChatBox internal list of names for a given channel

                LOCAL
                 - Similar to
                /ECHO
                 but allows a custom header

    Currently, FINDNICK is not implemented.

See the section on
                Formats for command IRC commands
                Also, see
                Special notes on on ChatBox's command parser.
```

1.11 ARexx: LOCAL command

```
COMMAND:

NICKONCH NICK/K CHANNEL/K STEM|VAR <variable>

PARAMETERS:

    HDR/K     Required!  Header to use.
    TXT/F/K   Required!  Text to place on line.

EXAMPLE:

    LOCAL HDR '[LocalTest]' TXT 'This is a test of the LOCAL command.'

    Yields:
    [LocalTest] | This is a test of the LOCAL command.
```

1.12 ARexx: NICKONCH command

```
COMMAND:

NICKONCH NICK/K CHANNEL/K STEM|VAR <variable>

PARAMETERS:

    NICK/K    Required!  The nick to look for on CHANNEL
    CHANNEL/K Required!  The channel to search for NICK

RESULTS:

    stem.ISON or <variable> is set to 1 if NICK is on CHANNEL, else 0

EXAMPLE:

NICKONCH NICK 'SASCMan' CHANNEL '#amiga' VAR ison

if ison==1 THEN
    say 'SASCMan is on channel #amiga'
else
    say 'SASCMan is not on channel #amiga'
```

1.13 ARexx: FINDNICK command

```
Not yet implemented.
```

1.14 ARexx: ISOP command

```
COMMAND:

ISOP NICK/K CHANNEL/K STEM|VAR <variable>

PARAMETERS:

    NICK/K    Required.  Nick to check for OPs, VOICE, or even existence...
    CHANNEL/K Required.  Channel name to look on (must be joined)

RESULTS:

    stem.ISOPPED, or <variable> is set to:

        1 if NICK is opped
        2 if NICK is voiced
        3 if NICK is not on the channel
        0 if NICK is not opped or voiced, but IS on channel

EXAMPLE:

ISOP NICK 'SkyGuy' CHANNEL '#amiga' VAR isopped
```

```
if isopped == 1 THEN
    say 'SkyGuy is opped'
else if isopped == 2 THEN
    say 'SkyGuy has voice'
else if isopped == 3 THEN
    say 'SkyGuy is not on #amiga'
else
    say 'SkyGuy is on #amiga, but is not opped or voiced'
```

1.15 ARexx: PATTERN command

```
COMMAND:

PATTERN ADDRESS/K PATT/K STEM|VAR <variable>

PARAMETERS:

    ADDRESS  The 'string' to match/check
    PATT     The pattern to match against

RESULTS:

    For STEM: variable.ISMATCH is set to the BOOLEAN values 1 or 0.
     For VAR: variable is set to BOOLEAN 1 or 0.

EXAMPLE:

    PATTERN ADDRESS SASCMan!jweb@priment.com PATT *!*jweb@*primenet.com VAR  ↩
       ismatch
    if ismatch THEN
        say 'We have a match!'
```

1.16 ARexx: NAMES command

```
COMMAND:

NAMES CHANNEL/K STEM|VAR <variable>

PARAMETERS:

    CHANNEL  The channel to get the names list from (you must be joined on
             this channel for a succesful return.

RESULTS:

    For STEM: variable.USERS.0-n, an array of 'names'
     For VAR: variable is set to a string of names, separated by spaces

EXAMPLE:

    NAMES #amiga STEM names.
    say names.USERS.1
```

1.17 Formats for common IRC commands

```
Only the most common RAW IRC command formats will be covered here.  For a more ↩
   indepth
review of other IRC Commands see the IRC-RFC.

When someone sends a /msg or just sends text to a channel it is sent with the
"PRIVMSG" IRC command:

Format for PRIVMSG:

    Nick!user@host PRIVMSG recipient :message text

    Recipient can be a channel or a nickname.  Message text can also contain a ↩
       CTCP
    (marked by a leading and trailing HEX $01).


When someone joins a channel the "JOIN" command is used:

Format for JOIN:

    Nick!user@host JOIN :channel


When someone leaves a channel, the "PART" command is used:

Format for PART:

    Nick!user@host PART :channel


When someone signs off (quits), the "QUIT" command is used:

Format for QUIT:

    Nick!user@host QUIT :signoff message


When someone issues a '/mode' change, the "MODE" command is used:

Format for MODE:

    Nick!user@host MODE <flags> :parameters

    Flags: i.e. +o, +b, -b, etc.   Parameters can be a list of nick names
    (for +o, +b, i.e.), or a list of addresses (for +/-b), a keyword (for +k),
    or a limit number (for +l).


"/notice" takes the same format as PRIVMSG, only substituting NOTICE where
PRIVMSG is.

Nick is the Nick name of the user that invoked the command, user@host is his/her
address.
```

```
HINT:  To distinguish between a private "PRIVMSG" and a public "PRIVMSG" you need
only check if the first character of stem.CHANNEL is "#" for global channel, or  ↩
    "&"
for server specific channel.  If it is one of those characters, it is a public
message.

Now, you should have read about the commands first...
```

1.18 ARexx: WAIT command

```
                COMMAND:

WAIT STEM|VAR <variable>

PARAMETERS:

    NONE

RESULTS:

    I recommend using a stem variable, which is invoked via:   WAIT stem variable.
    (don't forget the trailing period!!!)

    In the above example, all of ChatBox's text information would be stored in
    stem variable:  'variable.x'

    The defined stems are:  TOKENS, NICK, ADDRESS, CMD, CHANNEL, LINE

    TOKENS  is an array that holds each token in the RAW IRC command line sent
            by the server.
    NICK    is the nickname of the person that sent the command (or server address ↩
       )
    ADDRESS is the address of "NICK" (or server)
    CMD     is the command that was issued, or an error/reply number
    CHANNEL is the recipient (target) of the 'CMD', CHANNEL is not necessarily
            a channel name, it could be a nick name, if the action was private
    LINE    is the message text or parameters of the CMD line...
    CTCP    a BOOLEAN value (1 = TRUE, 0 = FALSE), tells if PRIVMSG was a CTCP

EXTRACTING TOKENS:

    If the following command were received:

:SASCMan!~jweb@ip000.phx.primenet.com PRIVMSG #amiga :ChatBox rules! ;)
0                                     1       2      3        4      5  <- TOKENS

    THEN:

stem.TOKENS.0 = SASCMan!~jweb@ip000.phx.primenet.com
stem.TOKENS.1 = PRIVMSG
stem.TOKENS.2 = #amiga
stem.TOKENS.3 = :ChatBox
stem.TOKENS.4 = rules!
stem.TOKENS.5 = ;)
stem.NICK     = SASCMan
```

```
stem.ADDRESS  = jweb@ip000.phx.primenet.com
stem.CMD      = PRIVMSG
stem.CHANNEL  = #amiga
stem.LINE     = ChatBox rules! ;)

In most circumstances you'll not need the TOKENS.  But they are invaluable when
you wish to pick the MESSAGE TEXT apart.

NOTES:

    ChatBox will queue up to 256 lines of text...  if your script is so slow
    that this limit is too low, then you're overdoing it.  256 lines of text
    can easily reach memory usage of 100k of RAM (or more).
    If the queue gets out of hand (i.e. scripts seem 'lagged'), simply issue a
    "/wait ON" command to flush the queue.
    You should always issue a "CMD '/wait on'" at the start of the script, and
    a "CMD '/wait off'" at the end of the script.
    See the
                /WAIT
                 command for more information.
```

1.19 ARexx: CMD command

```
COMMAND:

CMD CMDLINE/F [VAR <variable>]

PARAMETERS:

    CMDLINE  A ChatBox '/' (slash) command line.  See example below.

RESULTS:

    CMDLINE is executed by ChatBox's parser as if it were typed in the command
            entry gadget.
            Currently, CMD returns nothing.

EXAMPLE:

address 'ChatBox'
options results

CMD '/msg '||stem.NICK||' Hello, buddy!'
CMD '/join #amiga'
CMD '/msg '||stem.CHANNEL||' re Everyone!'
CMD '/op '||stem.NICK
CMD '/server irc.some.other.net'
```

1.20 ARexx: RAW command

```
COMMAND:

RAW RAWLINE/F

PARAMETERS:

    RAWLINE is a valid client to server IRC protocol command line.  See the IRC ↩
       RFC,
            or the section on the format of IRC commands.

RESULTS:

    Sends RAWLINE to the server to be executed.  RAW currently sets no return  ↩
       values.

EXAMPLE:

    RAW 'PRIVMSG #amiga :Hello Folks!!!'
    RAW 'MODE +oo-o :OpMe AndME DeopMe'
    RAW 'JOIN :#amiga'
```

1.21 ARexx: MATCH command

```
COMMAND:

MATCH NICK/K ADDRESS/K STEM|VAR <variable>

PARAMETERS:

    NICK    Nick to look up in ChatBox's internal 'FRIENDS' list.
    ADDRESS address associated with <nick>
    if 'STEM' is specified, followed by a stem variable, MATCH will place the
    results of it's match attempt in that stem variable.  If 'VAR' is used then
    ISMATCH and STATUS (see below) are concatenated into <variable>.

RESULTS:

    stem.ISMATCH - 0 if no match, 1 if there was a match
    stem.STATUS  - string indicating status i.e.:  +ckbo  (chop, kick, ban, op  ↩
       capable)

EXAMPLE:

address 'ChatBox.1'
options results

WAIT stem irc.

MATCH irc.NICK irc.ADDRESS stem info.

if info.ISMATCH == 1 then
    say irc.NICK||' is on the friends list with status of: '||info.STATUS
else
    say irc.NICK||' is not on the friends list.'
```

```
exit 0
```

1.22 Command set for ChatBox

```
                Special notes on on ChatBox's command parser.
                Channel operations:


                /JOIN, /J
                 <channel>

                /LEAVE, /L
                 [<channel>]

                /PART
                 [<channel>]

                /NAMES
                 [ON/OFF]|[<channel>,{<channel}]

                /TOPIC
                 [<channel>] [<topic>]


Channel operator operations:


                /MODE
                 [<nick>|<channel>] {[+/-]<modes>} [<limit>] [<user>] [<ban mask>]

                /KICK
                 [<channel>] <user> [<comment>]|[$]

                /OP
                 [<channel>] <user> [<user> [<user>]]

                /DEOP
                 [<channel>] <user> [<user> [<user>]]

                /BAN
                 [<channel>] <user> [<user> [<user>]]

                /UNBAN
                 [<channel>] <user> [<user> [<user>]]

                /VOICE
                 [<channel>] <user> [<user> [<user>]]

                /UNVOICE
                 [<channel>] <user> [<user> [<user>]]


Messaging:
```

```
/ME
 <action>

/MSG, /M
 <nick>|<channel>[,<nick>|<channel>[,...]] <message>

/NOTICE
 <nick>|<channel> <message>

/PING
 <nick>|<channel>

/CTCP
 <nick>|<channel> <type> [<parameters>]

/QUERY
 [<nick>|<channel>]
```

Server queries:

```
/ADMIN
 [<server>]

/CONNECT
 <target server> [<port> [<remote server>]]

/INFO
 [<server>]

/KILL
 <nick> <comment>

/LINKS
 [[<remote server>] <server mask>]

/STATS
 [<query> [<server>]]

/TIME
 [<server>]

/TRACE
 [<server>]

/VERSION
 [<server>]
```

Information queries:

```
/LIST
 [<pattern>]|[<channel_list>] [<server>]] [MIN <val>] [MAX <val>]
```

```
                /WHO
                 [<name> [<o>]]

                /WHOIS
                 [<server>] <nickmask>[,<nickmask>[,...]]

                /WHOWAS
                 <nick> [<count> [<server>]]


Server operations:

/QUIT      [<comment>]

                /SERVER
                 [<server>]

                /SIGN
                 [<comment>]

                /SIGNOFF
                 [<comment>]


Personal operations:


                /AWAY
                 [<comment>]

                /INVITE
                 <nick> [<channel>]

                /NICK
                 <nick>


ChatBox internals:


                /ADD
                 <flags> <address> [<nick>]

                /ALIAS
                 [<alias> [<command> [<parameters>]]]

                /DCC
                 <type> <nick> [<parameters>]

                /DESCRIBE
                 <nick> <action>

                /ECHO
                 <text>

                /ERRWIN
```

```
/EXEC
 <script>

/FRIENDS
 <ON/OFF>

/FREE
 <ALIAS>|<FRIENDS>

/HELP
 [<command>] [FULL]

/IGNORE
 <user | #> <type | option> [<type> [...]]

/MSGWIN

/NOTIFY
 [-]<nick> [[-]<nick> [...]]

/REM
 <*|address> [<nick>]

/NEWWIN

/STAT
 <address> [<nick>] [<flags>]

/LOG
 [<file>] [<channel>] [OPEN|CLOSE]

/LOAD
 <cmdfile>

/ON
 <type> <mask1> [<mask2>] </action> [<args>]

/RAW
 <raw_irc_cmd>

/RAWWIN

/READ
 [<textfile>]

/SEARCH
 <string> [BEGIN|MIDDLE|CONTINUE]

/WAIT
 <ON|OFF>
```

1.23 Special notes on ChatBox's command parser.

ChatBox's command set is pretty complete, but we always find a command we'd like to see added/supported that isn't really practical to implement inside of ChatBox's internal code.

This is where ARexx, and the special features of ChatBox's parser come into play.

With ARexx, ChatBox's command set is infinitely expandable. There are two ways to access ARexx scripts. Via the /EXEC command, or via ChatBox's 'special case' invocation. The /EXEC command is very straightforward, simply specify the ARexx macro to run as an argument to /EXEC (as well as any arguments that the script/macro itself may expect).

```
/EXEC kick.rexx SomeGuy
```

The above would execute the ARexx macro/script named "kick.rexx", which accepts the argument "SomeGuy" as input.

You can also use /ALIASes to create 'super commands', although this is cumbersome in light of ChatBox's 'special case' invocations.

'Special Case' invocations are cases where the command entered is not found in ChatBox's internal command set. In this case ChatBox will attempt to execute the ARexx command of the same name, with the extension ".cb" appended to it. For example, the /FIND command is not a ChatBox command, but can be easily created via "Special Case" and ARexx. By typing /FIND, you essentially tell ChatBox to try to execute "find.cb".

So, typing "/FIND ChatBox" would execute "find.cb ChatBox", or, in other words, it executes the script "find.cb", with the argument "ChatBox".

1.24 Command: /READ

COMMAND:

/READ [<textfile>]

PARAMETERS:

<textfile> Textfile to /READ, must be ASCII, and less than 20k in size.
If no file is specified, an ASL File Requester will be opened.

FUNCTION:

For online reading of textfiles.

1.25 Command: /ON

COMMAND:

/ON <type> <mask1> [<mask2>] </action> [<args>]

```
PARAMETERS:

    <type>    Type of /ON hook to install
    <mask1>   Pattern to match, varies with each <type>
    <mask2>   Secondary pattern to match, also varies with <type>
    </action> ChatBox slash command, or ARexx script to execute upon activation
              of this hook
    <args>    Any arguments that apply to </action>

FUNCTION:

    Installs, or REMOVES an /ON hook.  /ON hooks allow for trapping of certain
    stream events (like public messages, private message, channel operations,
    etc.)  Upon receiving an event that matches the criteria of an /ON hook,
    ChatBox will execute the </action> command with ChatBox's command parser.

OTHER PARAMETERS:

    If you type /ON with no parameters, you will get a list of all /ON hooks
    that are currently installed.  Indicated in the list are the /ON # (for
    referencing), the ACTIVE or INACTIVE state, and the /ON hook type and
    format.

    REMOVE <on #>

        <on #>  is the # you'd like to delete from the /ON hook list

    ACTIVATE [<on #>] [<life>]

        <on #>  is the # you'd like to activate
      [<life>]  specifies the number of times to execute this hook, until it
                deactivates itself.  If [<life>] is omitted, the previous life
                length is used (which may very well be 0).  If life is set to
                0, then it will never deactivate itself.

    DEACTIVATE [<on #>]

        <on #>  is the # you'd like to deactivate


    ***NOTE:  If the <on #> is omitted for ACTIVATE and DEACTIVATE, then the
              last /ON entered is assumed.  This is mainly useful in the
              'Alias File' for deactivating /ON hooks immediately after
              creating them.

    DEACTIVATEd /ON hooks are bypassed.  If you install a hook with a <life>
    parameter, it will only be executed that number of times before it
    deactivates itself.  Good for when you'd like an event to happen only a
    limited number of times.

    ***NOTE:  To set the <life> at /ON hook creation time, merely prepend the
              life limit to the TYPE.  i.e.:
              /ON 2/PUBLIC * /msg $C this is a limited /ON, it will run 2 times

              or...

              /ON 2PUBLIC * /msg $C '/' is an optional separator (see above)
```

NOTES:

Types of /ON hooks:

```
PUBLIC    - Hook that responds to PUBLIC channel messages.
MSG       - Hook that responds to PRIVATE messages
CTCP      - Responds to CTCP events/commands
JOIN      - Responds to JOIN events
LEAVE     - Responds to channel PARTS
QUIT      - Responds to signoffs/QUITs
NICKNAME  - Responds to NICKNAME changes
KICK      - Responds to KICK events
MODE      - Responds to MODE changes
TOPIC     - Responds to TOPIC changes
```

A response from the server must satisfy some criteria in an /ON hook before it is activated. First, the TYPE must be matched. If it is, then <mask1> is checked, and then <mask2> (if applicable). If all criteria are met, then the </action> is executed. <mask1> applies to all TYPES of hooks, whereas <mask2> may not need to be specified at all. It all depends on what you wish to respond to. The various types, and what the masks mean to them is specified below:

```
PUBLIC    - <mask1> is to match the address of the user that sent the
MSG         PUBLIC, MSG, or CTCP message.  <mask2> is applied to the
CTCP        actual text sent.

JOIN      - <mask1> is applied to the address of the person joining the
LEAVE       channel, <mask2> matches the channel name on which the join
            occurred.

KICK      - <mask1> is applied to the nick of the person being kicked,
            <mask2> is applied to the channel name on which the kick
            occurred.

TOPIC     - <mask1> is applied to the address of the user that changed the
            topic, and <mask2> is applied to the channel on which the topic
            was changed.

QUIT      - <mask1> is applied to the address of the user that quit,
            <mask2> is ignored.

NICKNAME  - <mask1> is applied to the address of the person that changed
            their nick (their original nick can be extracted from this),
            <mask2> is applied to the new nickname.
```

The </action> takes a form similar to ChatBox's alias, supports the $x variables. It also adds a new set of variables that apply only to PUBLIC, MSG, CTCP, and TOPIC.

Variables that apply to the server command lines:

```
$0 - $9  are the respective tokens of the server command issued, but only
         up to the 'text to send' portion (see notes below).
         $0 is typically the nickname of the sender
         $1 is the command name (raw irc server command)
```

```
            $2 is often the channel or recipient

    $C      is used to extract the channel, or recipient of this command (if
            applicable)

    $N      is used to extract the Nickname of the user from which the server
            command originated.

    $T      is used to extract the type of /ON hook that was activated (only
            to the ARexx interface)

    $U      is used to extract the full address of the user from which the
            server command originated

    []      is used to extract the 'rest of line', starting from the highest
            variable ($0 - $9) addressed (see notes below)

    Variables that apply to the "text sent":

    ^0 - ^9 are the respective words of the "text sent", useful for tokenizing
            a line word by word (separated by spaces, see notes below)
            ^0 is the CTCP type for a CTCP /ON

    {}      used to extract the 'rest of line', starting from the highest
            variable (^0 - ^9) addressed (see notes below)

    {x}     used to extract from 'x' to end. ( i.e.: {4} = ^4 {} )
    {-}     same as {}
    {x-y}   used to extract from 'x' to 'y'.
    {-y}    used to extract from highest variable address previously (^0 - ^9)
            up to 'y'
    {x-}    used to extract from 'x' to end. (same as {x})


    Format of a Server Command:

    :nick!user@host COMMAND argument [argument1 ...] :text to send

    All normal server commands and replies are issued in this format,
    varying based on the type of command/reply.

    The tokens ($0 - $9 and []) apply to the portion BEFORE the second ':' in
    the server reply.  Here is a sample PUBLIC command:

    :sascman!jweb@primenet.com PRIVMSG #Amiga :This text is seen on #Amiga
    ^-           0          -^ ^- 1 -^ ^- 2-^ ^-      text to send      -^

If a PUBLIC /ON hook was activated, and the </action> was specified as:

    (i.e.:  "/ON PUBLIC * /msg $C $U {}")

    /msg $C $U {}

Then the parsed command would look like:

    /msg #Amiga sascman!jweb@primenet.com This text is seen on #Amiga
```

```
Which would send (to #Amiga):

    sascman!jweb@primenet.com This text is seen on #Amiga

NOTE:  You will need the IRC-RFC to use the /ON command fully.

Standard UNIX pattern matching is used, case is ignored:

    "*" matches an arbitrary number of characters.
    "?" matches exactly ONE character.

Variables for "text to send":

    In the above server command example, the variables would return the
    following:

    ^0 This
    ^1 text
    ^2 is
    ^3 seen
    ^4 on
    ^5 #Amiga

    But, if you specifed something like:

    /action ^1 {}

    The {} would be replaced with tokens ^2 through to the end of the line,
    make the above '/action' parse out to:

    /action text is seen on #Amiga

    {} is all text after the highest token address (in this case ^1)


EXAMPLES:

/ON PUBLIC * @find * /find ^1 {}

    Whenever a user sends '@find' to the channel, the /ON hook will be
    activated, send a /find command to the parser.  The ^1 refers to the
    second word (^0 is the first) of the 'text to send'.
```

1.26 Command: /RAWWIN

```
COMMAND:

/RAWWIN

PARAMETERS:

    None.

FUNCTION:
```

```
    Sets a window to RAW mode, showing RAW server messages.

NOTES:

    This function of ChatBox is only useful for those that may be developing
    ARexx scripts.  It shows what the server is sending, and may aid in
    debugging scripts that rely on server communication command formats.
```

1.27 Command: /ECHO

```
COMMAND:

/ECHO <text>

PARAMETERS:

    <text>  Text to display locally in text window of active window.

FUNCTION:

    Displays text locally.

EXAMPLES:

    /ECHO This is a test of the /ECHO command.

    Yields:
    [Echo] | This is a test of the /ECHO command.
```

1.28 Command: /NOTIFY

```
COMMAND:

/NOTIFY [-]<nick> [[-]<nick> [...]]

PARAMETERS:

    <nick>  User you'd like a notify on (for signon/signoff)  If '-' precedes
            the <nick> then <nick> is removed if it is on the list of
            notifies.

FUNCTION:

    Adds/Removes a notify.  Notify uses ISON to detect when someone has joined
    or left IRC.

EXAMPLES:

/notify SkyGuy Tau Fastlane      Adds nicks to the notify list
/notify -SkyGuy -Fastlane        Removes nicks from the notify list

NOTES:
```

```
    If ChatBox should fail to open the timer.device it will fall back to using
    Intuition's 'Ticking' system.  The drawback to using Intuition is that the
    window MUST be active for these 'Ticks' to be sent.  So, if the window is
    not active (at least ONE window) then /NOTIFY will not be polled...

Silent Use of /NOTIFY:

The /NOTIFY command uses the ISON server command to obtain it's information.
This will result in seeing ISON replies from the server, if you wish to ignore
these messages, set the menu item in the Prefs menu "Show ISON" so that it does
not have a check mark next to it.
```

1.29 Command: /SEARCH

```
COMMAND:

/SEARCH <string> [BEGIN|MIDDLE|CONTINUE]

PARAMETERS:

    <string>  A string of characters to be searched for in the channel
              scrollback buffer.  If multi-word strings are being searched
              for, they should be enclosed in quotes.
    BEGIN     Search from BEGINning of buffer, if BEGIN is specified
    MIDDLE    Search from MIDDLE of buffer
    CONTINUE  Search from last position  [default]

FUNCTION:

    Searches channel scrollback buffers for a specified string

EXAMPLES:

    /SEARCH "Hello World!" BEGIN          - Searches for "Hello World!" starting
                                            from beginning of buffer.

    /SEARCH A4000                         - Searches from last position
                                            (CONTINUE) for the string "A4000"
```

1.30 Command: /IGNORE

```
COMMAND:

/IGNORE <user | #> <type | option> [<type> [...]]

PARAMETERS:

    <user>     If <user> is specified then the <types> must follow.  This
               is the usermask (pattern) to 'ignore'.
    <#>        If a <#> is specifed (as #1, #2, etc) then an <option> must
               follow.  This allows operations on a particular ignore.
```

```
    <option>   An <option> is only valid when a <#> is given.  Otherwise
               option names are just ignored.
    <type>     Type of incoming msg/cmd to ignore from <user>

    See Below for OPTIONS and TYPES


FUNCTION:

    Adds/Removes/Alters an ignore in the ignore list.

OPTIONS and TYPES:

    Current ignore types supported are:

    PRV   Ignore private messages from <user>
    PUB   Ignore public (to channel) messages from <user>
    NOTP  Ignore private notices from <user>
    NOT   Ignore public notices from <user>
    CTCP  Ignore CTCP's from <user> (will not reply to PINGs, etc.)


    Current options supported are:

    SET   Set new types for ignore #x
    UNSET Unset types from ignore #x
    REM   Remove ignore #x

EXAMPLES:

/ignore SASC* PRV PUB      (ignores Private and Public messages from any
                            user who's NICK starts with SASC)
/ignore *.primenet.com PRV (ignores all private messages from any user whose
                            address ends in ".primenet.com")

<user> is the usual IRC type of address mask of the form: nick!user@host

/ignore                    (displays HELP for ignore, and lists all ignores
                            currently in memory)
/ignore #2 SET PUB         (set ignore #2 to also ignore PUB messages)
/ignore #3 UNSET PRV       (unsets PRV from ignore #3)
/ignore #0 REM             (removes ignore #0)
```

1.31 Command: /DESCRIBE

```
COMMAND:

/DESCRIBE <nick> <action>

PARAMETERS:

    <nick>     This is the <nick> for the <action> to be sent to.
    <action>   The action to 'perform'

FUNCTION:
```

Sends an action message (similar to /ME, only it is privately sent)

1.32 Command: /RAW

COMMAND:

/RAW <raw_irc_cmd>

PARAMETERS:

<raw_irc_cmd> a VALID command line to send to current server.

FUNCTION:

Sends <raw_irc_cmd> to the server. See the IRC RFC for more information.
RAW IRC commands are beyond the scope of this document.

1.33 Command: /WAIT

COMMAND:

/WAIT ON|OFF

PARAMETERS:

ON|OFF

FUNCTION:

To activate/deactivate the ARexx WAIT queue. If /WAIT is ON, then lines
received are queued (up to 256). If "/WAIT ON" is again executed while
line queueing is already ON, the 'buffer' is flushed of all pending lines.
If a "/WAIT OFF" is executed then line queueing is deactivated, and the
'buffer is flushed.

NOTES:

This command has NO value except to those using ARexx and ChatBox's WAIT
command inside of an ARexx script. It is meant for consistant flow
control.

It is recommended that you issue a "/WAIT ON" command at the beginning of
every ARexx script that has a WAIT loop (i.e. a BOT).

Definitely issue a "/WAIT OFF" command before exiting ANY script that uses
the WAIT command. No harm is done if this step is forgotten... However,
if line queueing is not turned off between the execution of scripts,
unpredictable results can be expected from the WAIT command, as the queue
buffer must be emptied of all pending stored lines (and only 256 can be
stored).

```
    Leaving the /WAIT in the ON "position" with no script running, merely
    wastes valuable CPU and Memory resources.

    Example:

-------------->snip<--------------

/* sample WAIT loop */

HOST = address()  /* get the calling host name */
address value HOST

CMD '/wait on'    /* turn the LINE queue on */

irc.CMD = 'NONE'

do while irc.CMD != '/endrexx'
    say 'Command for this line: ' || irc.CMD
    say 'Channel for this line: ' || irc.CHANNEL
    say 'Message for this line: ' || irc.LINE
end

CMD '/wait off'   /* turn the LINE queue off */

exit 0

-------------->snip<--------------
```

1.34 Command: /KICK

```
COMMAND:

/KICK [<channel>] <user> [<comment>]|[$]

PARAMETERS:

    <channel>  You must be on <channel> to kick, if <channel> is not
               specified, then the kick is automatically sent to the
               current channel.

    <user>     This must be specified, and the <user> must be on the
               <channel> that the KICK is being sent to.

    <comment>  Not required, but recommended.

    $          If a '$' is specified instead of comment, ChatBox will select a
               random kick message from 'cb.kick' if it is present.

FUNCTION:

    KICK's <user> off of <channel>

NOTES:
```

```
    You must have '+o' status on the channel to KICK.
```

1.35 Command: /KILL

```
COMMAND:

/KILL <nick> <comment>

PARAMETERS:

    <nick>    Nick of the user to KILL off of IRC (forced QUIT)

    <comment> Required comment clarifying the reason

FUNCTION:

    KILL's <nick> from server, causing a forced QUIT

NOTES:

    You must be an IRCop to use the KILL command.
```

1.36 Command: /TOPIC

```
COMMAND:

/TOPIC [<channel>] [<topic>]

PARAMETERS:

    <channel> Channel to change the topic on, if not specified then the
              command is sent to the current channel

    <topic>   The new topic string

FUNCTION:

    Changes the topic on <channel> to <topic>

NOTES:

    If the channel mode is set to '+t' (topic protection) then you must have
    channel operator status (+o) to change the topic.
```

1.37 Command: /TIME

```
COMMAND:

/TIME [<server>]
```

```
PARAMETERS:

    <server> If not specified, the TIME command is sent to the current
             server.

FUNCTION:

    Returns the current time on <server>
```

1.38 Command: /TRACE

```
COMMAND:

/TRACE [<server>]

PARAMETERS:

    <server> server to trace the route to

FUNCTION:

    TRACE is used to find the route to <server>
```

1.39 Command: /SERVER

```
COMMAND:

/SERVER [<server>]

PARAMETERS:

    <server> server to connect to, if not specified the current server
             name is returned.

FUNCTION:

    To connect to a server.  Upon reconnect, all active channels are rejoined.
```

1.40 Commands: /SIGNOFF, /SIGN, /QUIT

```
COMMAND:

/SIGNOFF [<comment>]
/QUIT [<comment>]

PARAMETERS:

    <comment> An optional comment, if not specified then "Leaving." is used.

FUNCTION:
```

```
    To QUIT IRC, and to quit ChatBox.
```

1.41 Command: /STATS

```
COMMAND:

/STATS [<query> [<server>]]

PARAMETERS:

    <query>  type of "c,h,i,k,l,m,o,y,u"

    <server> server to get STATS on

FUNCTION:

    To obtain information about a server about <query>  See the IRC RFC
    for more information on STATS
```

1.42 Command: /WHO

```
COMMAND:

/WHO [<name> [<o>]] [FULL]

PARAMETERS:

    <name> A nickname, user address, channel or pattern to query

    <o>    If specified then only IRCops are listed.

FUNCTION:

    Generates a query which returns a list of information matching the
    <name> pattern.  Wildcards are permitted.

    If FULL is specified, then the full whois information is displayed,
    otherwise just the channel, status, and address are displayed.
```

1.43 Command: /WHOIS

```
COMMAND:

/WHOIS [<server>] <nickmask>[,<nickmask>[,...]]

PARAMETERS:

    <server>   if specified, checks only <server>
```

```
    <nickmask> a wildcard to match, or a known nickname

FUNCTION:

    Returns info about <nickmask>
```

1.44 Command: /WHOWAS

```
COMMAND:

/WHOWAS <nick> [<count> [<server>]]

PARAMETERS:

    <nick>   a <nick> that no longer exists (because of a QUIT or NICK
             change)

    <count>  if specified, then only <count> replies will be returned
             if not specified, then all replies are sent.

    <server> server to query

FUNCTION:

    Returns info about a <nick> that no longer exists.
```

1.45 Command: /JOIN

```
COMMAND:

/JOIN <channel>

PARAMETERS:

    <channel> name of channel to join

FUNCTION:

    To JOIN a channel.  If the channel does not exist then you are given
    operator status on the newly created channel.
```

1.46 Command: /PING

```
COMMAND:

/PING <nick>|<channel>

PARAMETERS:

    <nick>    a nick to PING
```

```
    <channel> a channel to ping.  All users on <channel> will be sent a
              PING to reply to.

FUNCTION:

    Sends a CTCP of type PING to <nick> or <channel>.
```

1.47 Commands: /LEAVE, /L, /PART

```
COMMAND:

/LEAVE [<channel>]
/L [<channel>]
/PART [<channel>]

PARAMETERS:

    <channel> channel to leave, if not specified the current channel is
              left

FUNCTION:

    To exit from <channel>
```

1.48 Command: /LIST

```
COMMAND:

/LIST [<pattern>]|[<channel>{,<channel>}] [<server>] [MAX <max>] [MIN <min>]

PARAMETERS:

    <pattern> list channels matching <pattern>

    <channel> channel to list

    <server>  server to query

    MAX <max> list only channels that have less than or equal to 'max' number of
              people
    MIN <min> list only channels that have more than or equal to 'min' number of
              people

FUNCTION:

    Lists channels and their topics.  If <pattern> is specified then only
    channels that match the pattern and that are not secret or private (+s or
    +p) are listed.
```

1.49 Command: /LINKS

```
COMMAND:

/LINKS [[<remote server>] <server masg>]

PARAMETERS:

    See IRC RFC for more information.
```

1.50 Command: /NICK

```
COMMAND:

/NICK <nick>

PARAMETERS:

    <nick> New nickname.

FUNCTION:

    Change current nickname to <nick>
```

1.51 Command: /NOTICE

```
COMMAND:

/NOTICE <nick>|<channel> <message>

PARAMETERS:

    <nick>|<channel>  the nick or channel to send <message> to.

FUNCTION:

    Sends a notice (similar to a /msg) to <nick>, or <channel>
```

1.52 Command: /NAMES

```
COMMAND:

/NAMES [ON/OFF]|[<channel>{,<channel>}]

PARAMETERS:

    ON/OFF    to turn the names list ON or OFF.  The names list splits the
              current window on the right hand side.  It sports a scroll gadget,
              and two arrows. *** SEE NOTE BELOW ***
```

```
    <channel> to get the members names from.

FUNCTION:

    Lists the names for <channel>.  Assuming you are a> on the channel, or
    b> off the channel AND the +p/+s mode is not set.

NOTES:

    The names list can be used to supply nicks/names for commands that require a
    nick/name as input.  If a double-click is made on a nick/name, a /WHOIS will
    be issued.  /KICK also works, but only with the first name selected so as to
    avoid mass kick abuse that is certain to occur.

*** NOTE ***  THIS IS IMPORTANT!!!
    I keep getting 'bug reports' about how ChatBox does NOT sort the names list
    correctly.  I assure you, and I reassure you, and I stress, and I point out
    that I want to make it so VERY completely and utterly unmistakeably CLEAR
    that this is NOT a bug... I repeat, it is NOT A BUG.  I don't CARE if one
    thinks it IS a bug because it 'appears' to become 'unsorted'.  Here is the
    VERY BOTTOM LINE on ChatBox's sorting method, so READ IT, UNDERSTAND IT,
    and DON'T REPORT "BUGS" ABOUT IT!!!

    Here is how it works, and how it will ALWAYS work, if it BUGS you SOOOO
    much that it works this way, use GV, it's a VERY nice client, I hear.

    ChatBox SORTS the names upon YOUR entry into a channel, it also sorts NEW
    joins (when OTHER people join the channel).  It DOES NOT take goofy lead
    characters into consideration (like "_", "[", etc.)  It DOES NOT sort on
    nick changes, nor will it EVER (get that?  EVER!!!)

    SO, in conclusion, IF the names APPEAR to be out of sort, THEY ARE, and it
    is probably due to a NICK CHANGE.  OR, the person has a 'goofy' lead
    character in their nick.

    MY reasons (which are reasons enough):  In EARLY development of ChatBox I
    was sitting quietly on #Amiga, minding my own business, when suddenly about
    4 idiots joined the channel (1 idiot, with 4 crapbots, actually), they
    began NICK flooding.  NEEDLESS TO SAY, this created a TREMENDOUS load on
    the CPU, and on ChatBox.  I couldn't kick this person, I could do nothing
    but watch while everything on my machine slowed to a crawl.  Why did the
    CPU get so loaded down?  Well, let's see, could it have been the SORT
    ROUTINES???  Indeed, it was, hence, the nick change sorting WAS REMOVED,
    it IS NOT A BUG!

    Done.  Sorry for the flame, but the very same people are 'reminding' me of
    this bug every time they run across me, or remember I have an E-Mail
    address.  I guess I wasn't being clear enough. The above should pretty much
    set things straight.
```

1.53 Command: /ME

```
COMMAND:

/ME <action>
```

PARAMETERS:

<action> a personal action. i.e., if your nick is ChatBox and you entered:
'/me washes behind his ears.'
You'd see: (and so would all others on the channel)
-<ACTION>- ChatBox washes behind his ears.

FUNCTION:

For creative expression... A toy. A way to talk in the third person...

1.54 DCC.CHAT.AS225

SPECIAL NOTES ON DCC.CHAT.AS225

The DCC.CHAT executable included with ChatBox is NOT compatible with any other IRC Client. It was written with ChatBox, specifically, in mind.

One important feature of IRC is the ability to secure a CHAT transaction with any party connected to IRC. In designing ChatBox it was decided that having a separate 'DCC CHAT' window that used some other GUI was not acceptable.

DCC Chat uses ChatBox's GUI system. So, it will feel as though you are still using ChatBox IRC.

The design goal was met. Simplicity was the goal.

The advantage of having text send to a ChatBox window is clear. Why have a different type of GUI to have to get used to?

Osma "Tau" Ahvenlampi should be given GREAT credit for his contribution of DCC.CHAT.AS225 to the Amiga IRC community. It is an extraordinary alternative.

ChatBox's DCC.CHAT is based on his code. He offered the framework, and is given credit for designing the connection code (of which I know little about).

However, if you prefer having a separate, and distinct DCC client, then by all means you should use Tau's original version of DCC.CHAT.AS225.

1.55 Command: /MSG

COMMAND:

/MSG <nick>|<channel>[,<nick>|<channel>[,...]] <message>

PARAMETERS:

<nick>|<channel>|[<,><.>] are the recipient(s) of <message>

FUNCTION:

```
    To send a message to a user or channel.

NOTE:  If an '=' is placed before the <nick>, then ChatBox sends the text to
       the appropriate DCC Chat session.  i.e.:  /msg =SkyGuy <message>  would
       send <message> to SkyGuy over a DCC Chat connection (see section on

                 DCC.CHAT.AS225
                  for more information.  Also, if a ','
       is placed in the <nick> position, then the last person to send you a
       message is replied to.  If a '.' is placed there, then the last person
       you sent a message to is sent the current message.
```

1.56 Command: /MODE

```
COMMAND:

/MODE [<nick>|<channel>] {[+/-]<modes>} [<limit>] [<user>] [<ban mask>]

PARAMETERS:

    <nick>     Personal <nick> for user mode changes (i,o,w, etc)
    <channel>  for channel modes, or o,b,v, etc.
    [+/-]      + sets a flag ON, - sets it OFF
    <modes>    mode characters (i.e. i,o,b,v,t,n,s,p,l, etc.)
    <limit>    for mode +l (limit users), an ASCII number setting the limit of
               users on <channel>
    <user>     for modes: o,b,v
    <ban mask> ban address with wildcards for +/-b mode

FUNCTION:

    To change personal user modes, or channel modes.  Or to op/voice,
    deop/unvoice other users on a channel where you have ops.

NOTES:

    See the IRC-RFC for a full description of the modes, and their usage.
```

1.57 Command: /INVITE

```
COMMAND:

/INVITE <nick> [<channel>]

PARAMETERS:

    <nick>    The user to invite
    <channel> the channel to invite them to.  You do not have to be on <channel>
              to invite a user there.  If <channel> is not specified, then the
              current channel is assumed.
```

FUNCTION:

To send an invitation to another user to indicate you'd like them to join <channel>

1.58 Command: /INFO

COMMAND:

/INFO [<server>]

PARAMETERS:

<server> server to query information from, if not specified then current server joined is assumed.

FUNCTION:

To obtain further information on <server>.

NOTES:

See the IRC-RFC for more info on this command.

1.59 Command: /AWAY

COMMAND:

/AWAY [<comment>]

PARAMETERS:

<comment> message to appear when you are marked away. If not specified then the 'away' tag will be removed.

FUNCTION:

To notify others that you are away from keyboard (AFK) or not watching, etc.

1.60 Command: /ADMIN

COMMAND:

/ADMIN [<server>]

PARAMETERS:

<server> server to query information about it's administrat[ion]/[or]

```
FUNCTION:

    To obtain more information on about a server's adminitration
```

1.61 Command: /ALIAS

```
                COMMAND:

/ALIAS [<alias> [<command> [<parameters>]]]

PARAMETERS:

    <alias> the name to give this alias
    <command> the actual ChatBox command you'd like executed
    <parameters> the parameters appropriate for <command>

    If none of the parameters are specified, then all current aliases are
    listed.

FUNCTION:

    To supply shortcut aliases to a ChatBox IRC session.

EXAMPLES:

Example 1:  /ALIAS bye quit I'm OUTTA here!
Example 2:  /ALIAS test msg $C Useless Alias being tested.
Example 3:  /ALIAS umode mode $N
Example 4:  /ALIAS chop add +c *!*$1 $0
Example 5:  /ALIAS chg stat * $0 $1

For Example 1, if '/bye' was typed it would be the same as typing:
               /QUIT I'm OUTTA here!
For Example 2, if you were currently on #amiga, and typed '/test' it would be
               the same as entering:
               '/msg #amiga Useless Alias being tested'
               #amiga is substituted for $C
For Example 3, if your nick were 'ChatBox', and you typed '/umode +i', it would
               be the same as entering:
               '/mode ChatBox +i'   ChatBox is substituted for $N
For Example 4, if the following was entered:
               '/chop SASCMan jweb@*.phx.primenet.com'
               this would be the 'real' command:
               '/add +c *!*jweb@*.phx.primenet.com SASCMan'
For Example 5, if the following were entered:
               '/chg SASCMan +ckb'
               the actual command issued would be:
               '/stat * SASCMan +ckb'

$0-$9 are parsed as follows:

         $0   $1   $2   $3   $4       $9
/cmdname arg0 arg1 arg2 arg3 arg4 ... arg9

NOTES:
```

```
    $N    -  Substitution variable for current nickname
    $C    -  Substitution variable for current channel
    $0-$9 -  Substitution variables for arguments in command entered
    []    -  Substitute the rest of line.  This should always be specifed
             if an alias has any parameters.  If $0-$9 are used, then the
             rest of line is all arguments after the highest '$' token
             used.  i.e.:  if $3 was the highest used, then '[]' consists
             of all arguments from $4 to the last argument.

    An Alias file is automatically loaded at startup if it exists as
    "S:ircrc.als".  If the file does not exist as such, you can use

                /LOAD alias.filename
                .

SEE ALSO:


                Format for Alias file (ircrc.als)

                /LOAD
```

1.62 Command: /VERSION

```
COMMAND:

/VERSION [<server>]

PARAMETERS:

    <server>  server to query version information from

FUNCTION:

    To obtain version info on <server>
```

1.63 Command: /VOICE

```
COMMAND:

/VOICE [<channel>] <user> [<user> [<user>]]
/UNVOICE [<channel>] <user> [<user> [<user>]]

PARAMETERS:

    <channel> Channel to send /VOICE or /UNVOICE change to
    <user> user to give/remove +v (VOICE) status to.

FUNCTION:
```

```
    To give/remove voice status to/from <user>'s.  Allows/disallows users on a
    moderated channel (/mode +m) to speak.  Users without +v (voice) cannot
    send text to the channel, if +m is set.
```

1.64 Command: /CONNECT

```
COMMAND:

/CONNECT <target server> [<port> [<remote server>]]

PARAMETERS:

    <target server> server to CONNECT to
    <port>          port on <target server> to use
    <remote server> server to CONNECT to <target server>

FUNCTION:

    To (re)establish a connection between servers.  If <remote server> is
    specified, then an attempt is made to connect it to <target server>.

NOTES:

    See IRC-RFC for more information on CONNECT.  It is a restricted command,
    availble only to IRC operators.
```

1.65 Command: /CTCP

```
COMMAND:

/CTCP <nick>|<channel> <type> [<parameters>]

PARAMETERS:

    <nick>|<channel>  recipient of the CTCP msg
    <type>  type of CTCP (i.e.  PING, VERSION, CLIENTINFO, PID, etc.)
    <parameters> parameters specific to <type>

FUNCTION:

    To send query type msgs to <nick>|<channel>.  Responses usually provide more
    info, or access to functions available via <nick>.
```

1.66 Commands: /OP, /DEOP

```
COMMANDS:

/OP [<channel>] <user> [<user> [<user>]]
/DEOP [<channel>] <user> [<user> [<user>]]
```

```
PARAMETERS:

    <channel> if specified, then the mode change is sent to <channel>, otherwise
              current channel is assumed
    <user>    user to give/remove channel operator status to/from

FUNCTION:

    Gives/removes user +/-o (ops/deop) on <channel>.
    Same as /mode <channel> +/-o <user>.
```

1.67 Commands: /BAN, /UNBAN

```
COMMAND:

/BAN [<channel>] <user> [<user> [<user>]]
/UNBAN [<channel>] <user> [<user> [<user>]]

PARAMETERS:

    <channel> if specified, then the mode change is sent to <channel>, otherwise
              current channel is assumed
    <user>    user to set/remove ban on/from

FUNCTION:

    Sets/removes user +/-b (ban) on <channel>.
    Same as /mode <channel> +/-b <user>.

NOTES:

    ChatBox maintains an internal banlist, if a 'mode #channel +b' is requested
    (a banlist request).  To view the internal banlist:

    /ban $list

    To remove a ban from the list:

    /ban $rem #

    Where the '#' is not the pound sign, but the number of the ban you'd like
    to remove.

    See also:  /FRIENDS ON  for information on a special function of /ban and
               /unban
```

1.68 Command: /HELP

```
COMMAND:

/HELP [<command>] [FULL]
```

```
PARAMETERS:

    <command>  command to get /HELP on, if not specified, then all commands are
               listed.

FUNCTION

    To provide template help on <command>.  If FULL is specified, then the full
    help text will be displayed from ChatBox.guide, if the .guide is in the
    current directory.
```

1.69 Command: /EXEC

```
                COMMAND:

/EXEC <arexxscript>

PARAMETERS:

    <arexxscript>  an ARexx script/macro to invoke.

FUNCTION

    To start an ARexx script.
    See the section on
                ChatBox's ARexx Port
```

1.70 Command: /QUERY

```
COMMAND:

/QUERY [<nick>|<channel>]

PARAMETERS:

    <nick>|<channel>  Recipient of the query.  Causes all undirected input to be
                      sent to <nick> or <channel>.  If not specified, then the
                      /QUERY is cleared for that window.

FUNCTION:

    To allow a private/public conversation with <nick>/<channel> respectively
    without having to type /msg <nick>|<channel> each time.

NOTE:  If an '=' is placed before <nick>, then the query will be on the
       specified DCC Chat session specified by <nick>.  See the section on
       "DCC.CHAT.AS225" for more information.
```

1.71 Command: /NEWWIN

```
COMMAND:

/NEWWIN

PARAMETERS:

    None.

FUNCTION:

    To open a new ChatBox channel window, without joining a channel.  To use as
    a query/msg/crap window, for example.
```

1.72 Command: /MSGWIN

```
COMMAND:

/MSGWIN

PARAMETERS:

    None.

FUNCTION:

    To direct private messages/notifications to window in which /MSGWIN was
    executed.
```

1.73 Command: /ERRWIN

```
COMMAND:

/ERRWIN

PARAMETERS:

    None.

FUNCTION:

    To direct error messages/notifications to window in which /ERRWIN was
    executed.
```

1.74 Command: /ADD

```
COMMAND:

/ADD <flags> <address> [<nick>]
```

PARAMETERS:

<flags> flags to set/unset (+/-) or ALL (see NOTES below)
<address> address of the 'friend', wildcards are permitted. (standard IRC address format)
<nick> if specified allows easier access to the /FRIENDS list

FUNCTION:

To 'add' a friend to the '/FRIENDS' list. Giving them access to certain ChatBox features in the friends architecture.

NOTES:

Flags and there uses:

- c - CHOPS, or channel operator status on /FRIENDS list. With this flag set on a friend, they will be auto-opped when they join a channel with /FRIENDS ON
- k - KICK, or KICK status. With this flag set on a friend they can use ChatBox's kick to remove a user from the current channel (bot like function)
- b - BAN, or ban status. With this flag set on a friend they can use ChatBox's smart ban facilities (see notes on bans below)
- o - OPS, or op status, with this flag set this user can give OPs to other users that are on the friends list. But will not allow OP'ing non-friends (good for when no ops are live, and more ops are needed. for example, when the chatbox session is started with no ops in channel, and no one is paying attention except for a user who has +o on his friend status.)
- a - ADD, or add status. With this flag set this user can /ADD more friends to the ChatBox list (remotely)
- r - REMOVE, or remove status. With this flag set for a friend, they can remove a user from the ChatBox friends list (remotely)
- l - LIST, or list status. Allows a friend with this flag set to list out the current friends list (this requires MEGA bandwidth, and usually results in FLOOD kills when friends lists are large)
- s - STAT, with this flag set a user can query ChatBox for info on a friend (to modify)

FRIENDS COMMANDS:

%chop	- no parameters
%kick <nick> <comment>	- same as /kick, except no channel is specified
%ban <nick>|<ban mask>	- same as /ban
%op <nick>	- same as /op
%add <flags> <address> <nick>	- same as /add
%remove <*>|<address> <*>|<nick>	- same as /remove
%list	- no parameters
%stat <*>|<address> <*>|<nick> <flags>	- same as /stat

EXAMPLE:

/ADD +ckbo *!jweb@*primenet.com SASCMan
The above adds SASCMan (jweb@primenet.com) to the friends list (this is not saved to disk) with flags of +ckbo

```
    /ADD ALL *!ding@*.silly.world.com SillyMan
    The above adds SillyMan to the friends list with ALL flags set

SEE ALSO:

    /FRIENDS, /STAT, /REM, /FREE
```

1.75 Command: /REM

```
COMMAND:

/REM <*>|<address> <*>|<nick>

PARAMETERS:

    <address> address specification of friend to remove
    <nick>    irc name of friend to remove.
    <*>       place holder.  If address is not specified, a '*' should be used
              as a place holder.  (note, do not edit ARexx scripts to conform
              to this, as it will most certainly change as soon as I devise a
              more intuitive way of handling this)

FUNCTION:

    Removes a friend from the current, in memory, friends list.  Does not remove
    it from the friends file

EXAMPLE:

    /REM #?!jweb@#?.phx.primenet.com   - removes address (friend) from current
                                         list in memory
    /REM * SASCMan                     - removes SASCMan from the current list
```

1.76 Command: /STAT

```
COMMAND:

/STAT <*>|<address> [<nick>] [<flags>]

PARAMETERS:

    <address> address of friend to get info on or change status of
    <nick> nick of friend, if <address> is not used (with '*' as a place holder)
    <flags> new flags to set/unset for friend specified.
    <*) a place holder for <address> in case <nick> is used to find the friend.
    IF no parameters are given, then the current friends list will be output to
    the window.

FUNCTION:

    To list/change/query info/status of a friend.
```

```
EXAMPLE:

    /STAT * SASCMan                       - Shows current status info on
                                            SASCMan
    /STAT #?!root@#?.slip.uiuc.edu        - Lists current status info on
                                            specified address
    /STAT #?!silly@#?.corny.edu * +c-k     - Sets status to +c, -k for address
                                            specified
    /STAT * SASCMan +ck-b-o               - Sets status +c, +k, -b, -o for
                                            SASCMan
    /STAT                                 - Lists out current Friends list.
```

1.77 Command: /FRIENDS

```
                COMMAND:

/FRIENDS <ON/OFF/WHO>

PARAMETERS:

    <ON/OFF/WHO> to activate or deactivate the friends list for the window in
                 which ON/OFF was specified

FUNCTION:

    To make a friends list active or non-active.

NOTES:

    If 'WHO' is specified, the friends list is left OFF, but the WHO list
    is loaded so that smart banning can be used.

    When a friends list is turned 'ON' a /WHO #channel is done to grab the
    address of all users currently on the channel, so there may be a slight
    delay and then an "End of WHO" message.  With /FRIENDS ON, a special
    ChatBox feature is enabled (regardless of whether the list actually
    has any 'friends' in it or not).  /BAN becomes 'smart'.  For example:

    A user is DorkMan (~dork@slip34.ding.dong.com).
    And '/ban DorkMan' is entered...  ChatBox will set the ban as follows:
    /mode #channel +b *!*dork@*.ding.dong.com

SEE ALSO:

    /ADD, /REM, /STAT, /FREE,

                Format for FRIENDS files
```

1.78 Command: /FREE

```
COMMAND:

/FREE ALIAS|FRIENDS

PARAMETERS:

    ALIAS   the keyword ALIAS when used (ie: /free ALIAS) free's the alias list
            from memory
    FRIENDS the keyword FRIENDS when used (ie: /free FRIENDS) free's the friends
            list from memory

FUNCTION:

    To free up memory resources by eliminating /ALIAS and /FRIENDS lists.
```

1.79 Command: /LOG

```
COMMAND:

/LOG [<file>] [<channel>] [CLOSE|OPEN]

PARAMETERS:

    <file>       file to create to keep logs (if not specified, <channel> name
                 is used as file name)
    <channel>    channel to creat log file for  (if not specified, current
                 channel is assumed)
    [CLOSE|OPEN] Keyword to OPEN or CLOSE the file for <channel>

FUNCTION:

    To create a log file for <channel> or the current channel.
```

1.80 Example /FRIENDS file

```
The format for a /FRIENDS file is VERY simple.  All lines beginning with a
command starter are executed by ChatBox's interpreter.  An example file:

--------------------> cut here <----------------------

Comment line
# or another way for a comment line
! or another
/add +ckbo #?!silly@#?.willy.wong.edu DingDong
/add +c #?!jweb@#?.phx.primenet.com SASCMan
/add ALL #?!root@#?.this.that.com Lost

#another comment

/add +ck #?!joe@#?.blo.com JoeBlo
```

```
------------------------>end<-----------------------

Any questions?

Save this file anywhere, and load it with:

/FRIENDS PATH:<friends.file>

Don't forget to activate the friends list with:

/FRIENDS ON
```

1.81 Example /ALIAS file

```
The format for an /ALIAS file is VERY simple.  All lines beginning with a
command starter are executed by ChatBox's interpreter.  An example file:

--------------------> cut here <----------------------

Comment line
# or another way for a comment line
! or another
/alias banlist mode $0 +b
/alias chop add +c *!*$1 $0
/alias tops add ALL *!*$1 $0
/alias umode mode $N []
/alias pme describe $0 []

#another comment
/alias vers ctcp $0 VERSION
/alias finger ctcp $0 FINGER
/alias send dcc send $0 $1
/alias move dcc move $0 $1
/alias chat dcc chat $0
/alias dmsg =$0 []
/alias dclose dcc $0 CLOSE

------------------------>end<-----------------------

Ez nuff?

NOTE:  Only /ALIAS commands will be parsed.
```

1.82 Command: /LOAD

```
COMMAND:

/LOAD <cmdfile>

PARAMETERS:

    <cmdfile> a file consisting of ChatBox command lines (i.e.:  alias file,
```

```
              'friends' file)

FUNCTION:

    To load and execute a ChatBox command file.

NOTES:

    Not all commands can be issued in a command file.  Currently, only
    the following commands are available:

    /ADD
    /ALIAS
    /AWAY
    /BAN     /UNBAN
    /FRIENDS
    /JOIN
    /MODE
    /NICK
    /OP      /DEOP
    /VOICE   /UNVOICE
    /NEWWIN
```

1.83 Command: /DCC

```
COMMAND:

/DCC <type> <nick> [<parameters>]

PARAMETERS:

    <type>       CHAT/SEND/MOVE, currently none of these are built in and
                 require DCC.send and DCC.chat to be installed in DCC:
    <nick>       recipient of DCC
    <parameters> filename for SEND/MOVE

FUNCTION:

    To establish a direct socket connection with <nick> for a more secure
    transaction.

SPECIAL NOTES:

    DCC Chat has now been 'personalized' for ChatBox.  The version included with
    ChatBox (DCC.CHAT.AS225) will only work with ChatBox.  Do not try to use it
    with other IRC clients, it is futile. ;)

    ChatBox's DCC system behaves in a fashion similar to IrcII.  When you wish
    to 'send' text to a connected party you prepend his/her nickname with '='
    to tell ChatBox that this message is intended for the recipient via a DCC
    Chat connection.

    The recommended method is to open a clear window with /NEWWIN, and use
    "/QUERY =nick" to carry on the transaction.  However, you may use ChatBox's
    /ALIAS system, or more simply use "/MSG =nick <message>".
```

```
Example /ALIAS for using DCC:

/ALIAS DMSG MSG =$0 []

With the above alias you can type:  /dmsg SkyD00d Hey!
And the text will be sent to =SkyD00d.

To close a DCC Chat session:  /DCC <nick> CLOSE
```

1.84 The Future of ChatBox

```
Features on ChatBox will be improving, and new ones will be added.  Currently,
I'm working on the following:

Completed Preferences Facility
More Menu Items (thus, keyboard shortcuts)
New DCC clients
Debug output window (displaying RAW server communications)
8SVX Beeping
External player beeping
Banlist editing via GUI facility
Server list maintenance (via preferences and 'on-the-fly' popup)
Online Help/FAQ system for CB commands and ARexx...
WHOIS popup GUI facility (for GUI based kicking, banning, etc.)

Planned Features:

Multi-Threaded support (via a separate Socket Task)
Iconization
More communication between CB and other tasks (a public MsgPort)
Visual communications (via a Graphic versus just a 'nick')
```

1.85 ChatBox's Author

```
My name is Jeff Webster, known as SASCMan on IRC.

I am a 26 year old software developer living in Phoenix, AZ.  I develop
software exclusively for the Amiga line of computers.  My specialty is
Operations Development.  I have designed two major operating systems
using the Amiga.  My first large project was ProVU-OS.  ProVU is an OS
that allows individuals in reconstruction to customize and expand the
underlying previewing system.  ProVU is mainly used by cosmetic surgeons,
interior decorators/designers, and reconstructive modelers to allow their
clients/patients to view the changes to be made, before they are actually
made.  ProVU is under constant development, always being improved.

My other project is Ami-POS.  Yes, Point Of Sale on the Amiga.  With Ami-POS
we broke new ground with Amiga networking hardware and software, and were
able to set up elaborate LAN (Local Area Networks) for small retailers (mostly
book and software houses).  Ami-POS is currently a complete project, undergoing
bug fixes only.
```

JDW Developments, my company, is moving forward to developling software for the general public and moving away from private business. The Internet has opened up a whole new world of opportunities.

My company consists of 7 Developers, including myself. Two of my developers are hardware specialists as well as being well versed in applications development. I do not have permission to mention their names here, so I will mention only first names: Drake, Jared, Gary, Stephen, Nicole, and Diane.

Drake is my hardware specialist, he designs network protocols as well as the hardware to communicate on.

Jared is my UNIX/LINUX specialist, he designs multi-user security systems as well as contributing to our network literacy.

Gary is my hacker, he can take any piece of hardware, and modify it to work safely on an Amiga. He has a degree in computer science, and is my partner from day one.

Stephen is my newest partner. He is a specialist in HTML, VRML and studying to be versed in Java. He is our parser developer (he designed the concept for ChatBox's new parser)

Nicole is my team leader, she develops support routines, specializing in GUI and visual interface construction and communication.

Diane is my style-guide compliant QUEEN. She makes sure that everything has the right 'feel' to it. She designs ALL of JDW Developments GUI's.

Please note that ChatBox is a solo effort, with the exception of a few suggestions and small enhancements here and there (i.e. Stephen's line parser)