Network Working Group   C. Kalt
Request for Comments: 2812   April 2000
Updates: 1459
Category: Informational

Internet Relay Chat: Client Protocol

Status of this Memo

This memo provides information for the Internet community. It does not specify an Internet standard of any kind. Distribution of this memo is unlimited.

Copyright Notice

IESG NOTE:

The IRC protocol itself enables several possibilities of transferring data between clients, and just like with other transfer mechanisms like email, the receiver of the data has to be careful about how the data is handled. For more information on security issues with the IRC protocol, see for example http://www.irchelp.org/irchelp/security/.

Abstract

The IRC (Internet Relay Chat) protocol is for use with text based conferencing; the simplest client being any socket program capable of connecting to the server.

This document defines the Client Protocol, and assumes that the reader is familiar with the IRC Architecture [IRC-ARCH].

Table of Contents

1. Labels

This section defines the identifiers used for the various components of the IRC protocol.

1.1 Servers

Servers are uniquely identified by their name, which has a maximum length of sixty three (63) characters. See the protocol grammar rules (section 2.3.1) for what may and may not be used in a server name.

1.2 Clients

For each client all servers MUST have the following information: a netwide unique identifier (whose format depends on the type of client) and the server which introduced the client.

1.2.1 Users

Each user is distinguished from other users by a unique nickname having a maximum length of nine (9) characters. See the protocol grammar rules (section 2.3.1) for what may and may not be used in a nickname.

While the maximum length is limited to nine characters, clients SHOULD accept longer strings as they may become used in future evolutions of the protocol.

1.2.1.1 Operators

To allow a reasonable amount of order to be kept within the IRC network, a special class of users (operators) is allowed to perform general maintenance functions on the network. Although the powers granted to an operator can be considered as 'dangerous', they are nonetheless often necessary. Operators SHOULD be able to perform basic network tasks such as disconnecting and reconnecting servers as needed. In recognition of this need, the protocol discussed herein provides for operators only to be able to perform such functions. See sections 3.1.8 (SQUIT) and 3.4.7 (CONNECT).

A more controversial power of operators is the ability to remove a user from the connected network by 'force', i.e., operators are able to close the connection between any client and server. The justification for this is very delicate since its abuse is both destructive and annoying, and its benefits close to inexistent. For further details on this type of action, see section 3.7.1 (KILL).

1.2.2 Services

Each service is distinguished from other services by a service name composed of a nickname and a server name. As for users, the nickname has a maximum length of nine (9) characters. See the protocol grammar rules (section 2.3.1) for what may and may not be used in a nickname.

1.3 Channels

Channels names are strings (beginning with a '&', '#', '+' or '!' character) of length up to fifty (50) characters. Apart from the requirement that the first character is either '&', '#', '+' or '!', the only restriction on a channel name is that it SHALL NOT contain any spaces (' '), a control G (^G or ASCII 7), a comma (','). Space is used as parameter separator and command is used as a list item separator by the protocol). A colon (':') can also be used as a delimiter for the channel mask. Channel names are case insensitive.

See the protocol grammar rules (section 2.3.1) for the exact syntax of a channel name.

Each prefix characterizes a different channel type. The definition of the channel types is not relevant to the client-server protocol and thus it is beyond the scope of this document. More details can be found in "Internet Relay Chat: Channel Management" [IRC-CHAN].

2. The IRC Client Specification

2.1 Overview

The protocol as described herein is for use only with client to server connections when the client registers as a user.

2.2 Character codes

No specific character set is specified. The protocol is based on a set of codes which are composed of eight (8) bits, making up an octet. Each message may be composed of any number of these octets; however, some octet values are used for control codes, which act as message delimiters.

Regardless of being an 8-bit protocol, the delimiters and keywords are such that protocol is mostly usable from US-ASCII terminal and a telnet connection.

Because of IRC’s Scandinavian origin, the characters {}|^ are considered to be the lower case equivalents of the characters []\˜, respectively. This is a critical issue when determining the equivalence of two nicknames or channel names.

2.3 Messages

Servers and clients send each other messages, which may or may not generate a reply. If the message contains a valid command, as described in later sections, the client should expect a reply as specified but it is not advised to wait forever for the reply; client to server and server to server communication is essentially asynchronous by nature.

Each IRC message may consist of up to three main parts: the prefix (OPTIONAL), the command, and the command parameters (maximum of fifteen (15)). The prefix, command, and all parameters are separated by one ASCII space character (0x20) each.

The presence of a prefix is indicated with a single leading ASCII colon character (’:’, 0x3b), which MUST be the first character of the message itself. There MUST be NO gap (whitespace) between the colon and the prefix. The prefix is used by servers to indicate the true origin of the message. If the prefix is missing from the message, it is assumed to have originated from the connection from which it was received from. Clients SHOULD NOT use a prefix when sending a message; if they use one, the only valid prefix is the registered nickname associated with the client.

The command MUST either be a valid IRC command or a three (3) digit number represented in ASCII text.

IRC messages are always lines of characters terminated with a CR-LF (Carriage Return - Line Feed) pair, and these messages SHALL NOT exceed 512 characters in length, counting all characters including the trailing CR-LF. Thus, there are 510 characters maximum allowed for the command and its parameters. There is no provision for continuation of message lines. See section 6 for more details about current implementations.

2.3.1 Message format in Augmented BNF

The protocol messages must be extracted from the contiguous stream of octets. The current solution is to designate two characters, CR and LF, as message separators. Empty messages are silently ignored, which permits use of the sequence CR-LF between messages without extra problems.

The extracted message is parsed into the components <prefix>, <command> and list of parameters (<params>).

The Augmented BNF representation for this is:

```
message    =  [ ":" prefix SPACE ] command [ params ] crlf
prefix     =  servername / ( nickname [ [ "!" user ] "@" host ] )
command    =  1*letter / 3digit
params     =  *14( SPACE middle ) [ SPACE ":" trailing ]
           =/ 14( SPACE middle ) [ SPACE [ ":" ] trailing ]

nospcrlfcl =  %x01-09 / %x0B-0C / %x0E-1F / %x21-39 / %x3B-FF
                ; any octet except NUL, CR, LF, " " and ":"
middle     =  nospcrlfcl *( ":" / nospcrlfcl )
trailing   =  *( ":" / " " / nospcrlfcl )

SPACE      =  %x20        ; space character
crlf       =  %x0D %x0A   ; "carriage return" "linefeed"
```

NOTES:

1) After extracting the parameter list, all parameters are equal whether matched by <middle> or <trailing>. <trailing> is just a syntactic trick to allow SPACE within the parameter.

2) The NUL (%x00) character is not special in message framing, and basically could end up inside a parameter, but it would cause extra complexities in normal C string handling. Therefore, NUL is not allowed within messages.

Most protocol messages specify additional semantics and syntax for the extracted parameter strings dictated by their position in the list. For example, many server commands will assume that the first parameter after the command is the list of targets, which can be described with:

```
target     =  nickname / server
msgtarget  =  msgto *( "," msgto )
msgto      =  channel / ( user [ "%" host ] "@" servername )
msgto      =/ ( user "%" host ) / targetmask
msgto      =/ nickname / ( nickname "!" user "@" host )
channel    =  ( "#" / "+" / ( "!" channelid ) / "&" ) chanstring
              [ ":" chanstring ]
servername =  hostname
host       =  hostname / hostaddr
hostname   =  shortname *( "." shortname )
shortname  =  ( letter / digit ) *( letter / digit / "-" )
              *( letter / digit )
                ; as specified in RFC 1123 [HNAME]
hostaddr   =  ip4addr / ip6addr
ip4addr    =  1*3digit "." 1*3digit "." 1*3digit "." 1*3digit
ip6addr    =  1*hexdigit 7( ":" 1*hexdigit )
ip6addr    =/ "0:0:0:0:0:" ( "0" / "FFFF" ) ":" ip4addr
nickname   =  ( letter / special ) *8( letter / digit / special / "-" )
targetmask =  ( "$" / "#" ) mask
                ; see details on allowed masks in section 3.3.1
chanstring =  %x01-07 / %x08-09 / %x0B-0C / %x0E-1F / %x21-2B
chanstring =/ %x2D-39 / %x3B-FF
                ; any octet except NUL, BELL, CR, LF, " ", "," and ":"
channelid  = 5( %x41-5A / digit )   ; 5( A-Z / 0-9 )
```

Other parameter syntaxes are:

```
  user       =  1*( %x01-09 / %x0B-0C / %x0E-1F / %x21-3F / %x41-FF )
                  ; any octet except NUL, CR, LF, " " and "@"
  key        =  1*23( %x01-05 / %x07-08 / %x0C / %x0E-1F / %x21-7F )
                  ; any 7-bit US_ASCII character,
                  ; except NUL, CR, LF, FF, h/v TABs, and " "
  letter     =  %x41-5A / %x61-7A       ; A-Z / a-z
  digit      =  %x30-39                 ; 0-9
  hexdigit   =  digit / "A" / "B" / "C" / "D" / "E" / "F"
  special    =  %x5B-60 / %x7B-7D
                   ; "[", "]", "\", "`", "_", "^", "{", "|", "}"
```

NOTES:

1) The <hostaddr> syntax is given here for the sole purpose of indicating the format to follow for IP addresses. This reflects the fact that the only available implementations of this protocol uses TCP/IP as underlying network protocol but is not meant to prevent other protocols to be used.

2) <hostname> has a maximum length of 63 characters. This is a limitation of the protocol as internet hostnames (in particular) can be longer. Such restriction is necessary because IRC messages are limited to 512 characters in length. Clients connecting from a host which name is longer than 63 characters are registered using the host (numeric) address instead of the host name.

3) Some parameters used in the following sections of this documents are not defined here as there is nothing specific about them besides the name that is used for convenience. These parameters follow the general syntax defined for <params>.

2.4 Numeric replies

Most of the messages sent to the server generate a reply of some sort. The most common reply is the numeric reply, used for both errors and normal replies. The numeric reply MUST be sent as one message consisting of the sender prefix, the three-digit numeric, and the target of the reply. A numeric reply is not allowed to originate from a client. In all other respects, a numeric reply is just like a normal message, except that the keyword is made up of 3 numeric digits rather than a string of letters. A list of different replies is supplied in section 5 (Replies).

2.5 Wildcard expressions

When wildcards are allowed in a string, it is referred as a "mask".

For string matching purposes, the protocol allows the use of two special characters: '?' (%x3F) to match one and only one character, and '*' (%x2A) to match any number of any characters. These two characters can be escaped using the character '\' (%x5C).

The Augmented BNF syntax for this is:

```
 mask       =  *( nowild / noesc wildone / noesc wildmany )
 wildone    =  %x3F
 wildmany   =  %x2A
 nowild     =  %x01-29 / %x2B-3E / %x40-FF
                 ; any octet except NUL, "*", "?"
 noesc      =  %x01-5B / %x5D-FF
                 ; any octet except NUL and "\"
 matchone   =  %x01-FF
                 ; matches wildone
 matchmany  =  *matchone
                 ; matches wildmany
```

Examples:

```
a?c         ; Matches any string of 3 characters in length starting
            with "a" and ending with "c"

a*c         ; Matches any string of at least 2 characters in length
            starting with "a" and ending with "c"
```

3. Message Details

On the following pages there are descriptions of each message recognized by the IRC server and client. All commands described in this section MUST be implemented by any server for this protocol.

Where the reply ERR_NOSUCHSERVER is returned, it means that the target of the message could not be found. The server MUST NOT send any other replies after this error for that command.

The server to which a client is connected is required to parse the complete message, and return any appropriate errors.

If multiple parameters is presented, then each MUST be checked for validity and appropriate responses MUST be sent back to the client. In the case of incorrect messages which use parameter lists with comma as an item separator, a reply MUST be sent for each item.

3.1 Connection Registration

The commands described here are used to register a connection with an IRC server as a user as well as to correctly disconnect.

A "PASS" command is not required for a client connection to be registered, but it MUST precede the latter of the NICK/USER combination (for a user connection) or the SERVICE command (for a service connection). The RECOMMENDED order for a client to register is as follows:

```
                        1. Pass message
        2. Nick message                 2. Service message
        3. User message
```

Upon success, the client will receive an RPL_WELCOME (for users) or RPL_YOURESERVICE (for services) message indicating that the connection is now registered and known the to the entire IRC network. The reply message MUST contain the full client identifier upon which it was registered.

3.1.1 Password message

```
   Command: PASS
Parameters: <password>
```

The PASS command is used to set a ’connection password’. The optional password can and MUST be set before any attempt to register the connection is made. Currently this requires that user send a PASS command before sending the NICK/USER combination.

Numeric Replies:

```
        ERR_NEEDMOREPARAMS              ERR_ALREADYREGISTRED
```

Example:

```
        PASS secretpasswordhere
```

3.1.2 Nick message

```
   Command: NICK
Parameters: <nickname>
```

NICK command is used to give user a nickname or change the existing one.

Numeric Replies:

ERR_NONICKNAMEGIVEN	ERR_ERRONEUSNICKNAME
ERR_NICKNAMEINUSE	ERR_NICKCOLLISION
ERR_UNAVAILRESOURCE	ERR_RESTRICTED

Examples:

```
NICK Wiz                ; Introducing new nick "Wiz" if session is
                        still unregistered, or user changing his
                        nickname to "Wiz"

:WiZ!jto@tolsun.oulu.fi NICK Kilroy
                        ; Server telling that WiZ changed his
                        nickname to Kilroy.
```

3.1.3 User message

   Command: USER
Parameters: <user> <mode> <unused> <realname>

The USER command is used at the beginning of connection to specify the username, hostname and realname of a new user.

The <mode> parameter should be a numeric, and can be used to automatically set user modes when registering with the server. This parameter is a bitmask, with only 2 bits having any signification: if the bit 2 is set, the user mode ’w’ will be set and if the bit 3 is set, the user mode ’i’ will be set. (See Section 3.1.5 "User Modes").

The <realname> may contain space characters.

Numeric Replies:

ERR_NEEDMOREPARAMS	ERR_ALREADYREGISTRED

Example:

```
USER guest 0 * :Ronnie Reagan   ; User registering themselves with a
                                username of "guest" and real name
                                "Ronnie Reagan".

USER guest 8 * :Ronnie Reagan   ; User registering themselves with a
                                username of "guest" and real name
                                "Ronnie Reagan", and asking to be set
                                invisible.
```

3.1.4 Oper message

```
      Command: OPER
   Parameters: <name> <password>
```

A normal user uses the OPER command to obtain operator privileges. The combination of <name> and <password> are REQUIRED to gain Operator privileges. Upon success, the user will receive a MODE message (see section 3.1.5) indicating the new user modes.

Numeric Replies:

```
           ERR_NEEDMOREPARAMS              RPL_YOUREOPER
           ERR_NOOPERHOST                  ERR_PASSWDMISMATCH
```

Example:

```
   OPER foo bar                    ; Attempt to register as an operator
                                   using a username of "foo" and "bar"
                                   as the password.
```

3.1.5 User mode message

```
      Command: MODE
   Parameters: <nickname>
               *( ( "+" / "-" ) *( "i" / "w" / "o" / "O" / "r" ) )
```

The user MODE's are typically changes which affect either how the client is seen by others or what 'extra' messages the client is sent.

A user MODE command MUST only be accepted if both the sender of the message and the nickname given as a parameter are both the same. If no other parameter is given, then the server will return the current settings for the nick.

The available modes are as follows:

```
           a - user is flagged as away;
           i - marks a users as invisible;
           w - user receives wallops;
           r - restricted user connection;
           o - operator flag;
           O - local operator flag;
           s - marks a user for receipt of server notices.
```

Additional modes may be available later on.

The flag ’a’ SHALL NOT be toggled by the user using the MODE command, instead use of the AWAY command is REQUIRED.

If a user attempts to make themselves an operator using the "+o" or "+O" flag, the attempt SHOULD be ignored as users could bypass the authentication mechanisms of the OPER command. There is no restriction, however, on anyone ‘deopping’ themselves (using "-o" or "-O").

On the other hand, if a user attempts to make themselves unrestricted using the "-r" flag, the attempt SHOULD be ignored. There is no restriction, however, on anyone ‘deopping’ themselves (using "+r"). This flag is typically set by the server upon connection for administrative reasons. While the restrictions imposed are left up to the implementation, it is typical that a restricted user not be allowed to change nicknames, nor make use of the channel operator status on channels.

The flag ’s’ is obsolete but MAY still be used.

Numeric Replies:

```
        ERR_NEEDMOREPARAMS              ERR_USERSDONTMATCH
        ERR_UMODEUNKNOWNFLAG            RPL_UMODEIS
```

Examples:

```
MODE WiZ -w                     ; Command by WiZ to turn off
                                reception of WALLOPS messages.

MODE Angel +i                   ; Command from Angel to make herself
                                invisible.

MODE WiZ -o                     ; WiZ ’deopping’ (removing operator
                                status).
```

3.1.6 Service message

```
   Command: SERVICE
Parameters: <nickname> <reserved> <distribution> <type>
            <reserved> <info>
```

The SERVICE command to register a new service. Command parameters specify the service nickname, distribution, type and info of a new service.

The <distribution> parameter is used to specify the visibility of a service. The service may only be known to servers which have a name matching the distribution. For a matching server to have knowledge of the service, the network path between that server and the server on which the service is connected MUST be composed of servers which names all match the mask.

The <type> parameter is currently reserved for future usage.

Numeric Replies:

```
        ERR_ALREADYREGISTRED            ERR_NEEDMOREPARAMS
        ERR_ERRONEUSNICKNAME
        RPL_YOURESERVICE                RPL_YOURHOST
        RPL_MYINFO
```

Example:

```
SERVICE dict * *.fr 0 0 :French Dictionary ; Service registering
                                itself with a name of "dict".  This
                                service will only be available on
                                servers which name matches "*.fr".
```

3.1.7 Quit

```
   Command: QUIT
Parameters: [ <Quit Message> ]
```

A client session is terminated with a quit message. The server acknowledges this by sending an ERROR message to the client.

Numeric Replies:

        None.

Example:

```
QUIT :Gone to have lunch        ; Preferred message format.

:syrk!kalt@millennium.stealth.net QUIT :Gone to have lunch ; User
                                syrk has quit IRC to have lunch.
```

3.1.8 Squit

```
      Command: SQUIT
   Parameters: <server> <comment>
```

The SQUIT command is available only to operators. It is used to disconnect server links. Also servers can generate SQUIT messages on error conditions. A SQUIT message may also target a remote server connection. In this case, the SQUIT message will simply be sent to the remote server without affecting the servers in between the operator and the remote server.

The <comment> SHOULD be supplied by all operators who execute a SQUIT for a remote server. The server ordered to disconnect its peer generates a WALLOPS message with <comment> included, so that other users may be aware of the reason of this action.

Numeric replies:

```
        ERR_NOPRIVILEGES                ERR_NOSUCHSERVER
        ERR_NEEDMOREPARAMS
```

Examples:

```
   SQUIT tolsun.oulu.fi :Bad Link ?  ; Command to uplink of the server
                                   tolson.oulu.fi to terminate its
                                   connection with comment "Bad Link".

   :Trillian SQUIT cm22.eng.umd.edu :Server out of control ; Command
                                   from Trillian from to disconnect
                                   "cm22.eng.umd.edu" from the net with
                                   comment "Server out of control".
```

3.2 Channel operations

This group of messages is concerned with manipulating channels, their properties (channel modes), and their contents (typically users). For this reason, these messages SHALL NOT be made available to services.

All of these messages are requests which will or will not be granted by the server. The server MUST send a reply informing the user whether the request was granted, denied or generated an error. When the server grants the request, the message is typically sent back (eventually reformatted) to the user with the prefix set to the user itself.

The rules governing how channels are managed are enforced by the servers. These rules are beyond the scope of this document. More details are found in "Internet Relay Chat: Channel Management" [IRC-CHAN].

3.2.1 Join message

```
   Command: JOIN
Parameters: ( <channel> *( "," <channel> ) [ <key> *( "," <key> ) ] )
            / "0"
```

The JOIN command is used by a user to request to start listening to the specific channel. Servers MUST be able to parse arguments in the form of a list of target, but SHOULD NOT use lists when sending JOIN messages to clients.

Once a user has joined a channel, he receives information about all commands his server receives affecting the channel. This includes JOIN, MODE, KICK, PART, QUIT and of course PRIVMSG/NOTICE. This allows channel members to keep track of the other channel members, as well as channel modes.

If a JOIN is successful, the user receives a JOIN message as confirmation and is then sent the channel’s topic (using RPL_TOPIC) and the list of users who are on the channel (using RPL_NAMREPLY), which MUST include the user joining.

Note that this message accepts a special argument ("0"), which is a special request to leave all channels the user is currently a member of. The server will process this message as if the user had sent a PART command (See Section 3.2.2) for each channel he is a member of.

Numeric Replies:

```
        ERR_NEEDMOREPARAMS              ERR_BANNEDFROMCHAN
        ERR_INVITEONLYCHAN              ERR_BADCHANNELKEY
        ERR_CHANNELISFULL               ERR_BADCHANMASK
        ERR_NOSUCHCHANNEL               ERR_TOOMANYCHANNELS
        ERR_TOOMANYTARGETS              ERR_UNAVAILRESOURCE
        RPL_TOPIC
```

Examples:

```
JOIN #foobar                    ; Command to join channel #foobar.

JOIN &foo fubar                 ; Command to join channel &foo using
                                key "fubar".
```

```
   JOIN #foo,&bar fubar            ; Command to join channel #foo using
                                   key "fubar" and &bar using no key.

   JOIN #foo,#bar fubar,foobar     ; Command to join channel #foo using
                                   key "fubar", and channel #bar using
                                   key "foobar".

   JOIN #foo,#bar                  ; Command to join channels #foo and
                                   #bar.

   JOIN 0                          ; Leave all currently joined
                                   channels.

   :WiZ!jto@tolsun.oulu.fi JOIN #Twilight_zone ; JOIN message from WiZ
                                   on channel #Twilight_zone
```

3.2.2 Part message

```
      Command: PART
   Parameters: <channel> *( "," <channel> ) [ <Part Message> ]
```

The PART command causes the user sending the message to be removed from the list of active members for all given channels listed in the parameter string. If a "Part Message" is given, this will be sent instead of the default message, the nickname. This request is always granted by the server.

Servers MUST be able to parse arguments in the form of a list of target, but SHOULD NOT use lists when sending PART messages to clients.

Numeric Replies:

```
           ERR_NEEDMOREPARAMS              ERR_NOSUCHCHANNEL
           ERR_NOTONCHANNEL
```

Examples:

```
   PART #twilight_zone             ; Command to leave channel
                                   "#twilight_zone"

   PART #oz-ops,&group5            ; Command to leave both channels
                                   "&group5" and "#oz-ops".

   :WiZ!jto@tolsun.oulu.fi PART #playzone :I lost
                                   ; User WiZ leaving channel
                                   "#playzone" with the message "I
                                   lost".
```

3.2.3 Channel mode message

```
      Command: MODE
   Parameters: <channel> *( ( "-" / "+" ) *<modes> *<modeparams> )
```

The MODE command is provided so that users may query and change the characteristics of a channel. For more details on available modes and their uses, see "Internet Relay Chat: Channel Management" [IRC-CHAN]. Note that there is a maximum limit of three (3) changes per command for modes that take a parameter.

Numeric Replies:

```
           ERR_NEEDMOREPARAMS              ERR_KEYSET
           ERR_NOCHANMODES                 ERR_CHANOPRIVSNEEDED
           ERR_USERNOTINCHANNEL            ERR_UNKNOWNMODE
           RPL_CHANNELMODEIS
           RPL_BANLIST                     RPL_ENDOFBANLIST
           RPL_EXCEPTLIST                  RPL_ENDOFEXCEPTLIST
           RPL_INVITELIST                  RPL_ENDOFINVITELIST
           RPL_UNIQOPIS
```

The following examples are given to help understanding the syntax of the MODE command, but refer to modes defined in "Internet Relay Chat: Channel Management" [IRC-CHAN].

Examples:

```
   MODE #Finnish +imI *!*@*.fi     ; Command to make #Finnish channel
                                   moderated and 'invite-only' with user
                                   with a hostname matching *.fi
                                   automatically invited.

   MODE #Finnish +o Kilroy         ; Command to give 'chanop' privileges
                                   to Kilroy on channel #Finnish.

   MODE #Finnish +v Wiz            ; Command to allow WiZ to speak on
                                   #Finnish.

   MODE #Fins -s                   ; Command to remove 'secret' flag
                                   from channel #Fins.

   MODE #42 +k oulu                ; Command to set the channel key to
                                   "oulu".

   MODE #42 -k oulu                ; Command to remove the "oulu"
                                   channel key on channel "#42".
   MODE #eu-opers +l 10            ; Command to set the limit for the
                                   number of users on channel
                                   "#eu-opers" to 10.

   :WiZ!jto@tolsun.oulu.fi MODE #eu-opers -l
                                   ; User "WiZ" removing the limit for
                                   the number of users on channel "#eu-
                                   opers".

   MODE &oulu +b                   ; Command to list ban masks set for
                                   the channel "&oulu".

   MODE &oulu +b *!*@*             ; Command to prevent all users from
                                   joining.

   MODE &oulu +b *!*@*.edu +e *!*@*.bu.edu
                                   ; Command to prevent any user from a
                                   hostname matching *.edu from joining,
                                   except if matching *.bu.edu

   MODE #bu +be *!*@*.edu *!*@*.bu.edu
                                   ; Comment to prevent any user from a
                                   hostname matching *.edu from joining,
                                   except if matching *.bu.edu

   MODE #meditation e              ; Command to list exception masks set
                                   for the channel "#meditation".

   MODE #meditation I              ; Command to list invitations masks
                                   set for the channel "#meditation".

   MODE !12345ircd O               ; Command to ask who the channel
                                   creator for "!12345ircd" is
```

3.2.4 Topic message

```
      Command: TOPIC
   Parameters: <channel> [ <topic> ]
```

The TOPIC command is used to change or view the topic of a channel. The topic for channel <channel> is returned if there is no <topic> given. If the <topic> parameter is present, the topic for that channel will be changed, if this action is allowed for the user requesting it. If the <topic> parameter is an empty string, the topic for that channel will be removed.

Numeric Replies:

```
        ERR_NEEDMOREPARAMS              ERR_NOTONCHANNEL
        RPL_NOTOPIC                     RPL_TOPIC
        ERR_CHANOPRIVSNEEDED            ERR_NOCHANMODES
```

Examples:

```
:WiZ!jto@tolsun.oulu.fi TOPIC #test :New topic ; User Wiz setting the
                                topic.

TOPIC #test :another topic      ; Command to set the topic on #test
                                to "another topic".

TOPIC #test :                   ; Command to clear the topic on
                                #test.

TOPIC #test                     ; Command to check the topic for
                                #test.
```

3.2.5 Names message

```
   Command: NAMES
Parameters: [ <channel> *( "," <channel> ) [ <target> ] ]
```

By using the NAMES command, a user can list all nicknames that are visible to him. For more details on what is visible and what is not, see "Internet Relay Chat: Channel Management" [IRC-CHAN]. The <channel> parameter specifies which channel(s) to return information about. There is no error reply for bad channel names.

If no <channel> parameter is given, a list of all channels and their occupants is returned. At the end of this list, a list of users who are visible but either not on any channel or not on a visible channel are listed as being on ‘channel’ "*".

If the <target> parameter is specified, the request is forwarded to that server which will generate the reply.

Wildcards are allowed in the <target> parameter.

Numerics:

```
        ERR_TOOMANYMATCHES              ERR_NOSUCHSERVER
        RPL_NAMREPLY                    RPL_ENDOFNAMES
```

Examples:

```
NAMES #twilight_zone,#42        ; Command to list visible users on
                                #twilight_zone and #42

NAMES                           ; Command to list all visible
                                channels and users
```

3.2.6 List message

```
   Command: LIST
Parameters: [ <channel> *( "," <channel> ) [ <target> ] ]
```

The list command is used to list channels and their topics. If the <channel> parameter is used, only the status of that channel is displayed.

If the <target> parameter is specified, the request is forwarded to that server which will generate the reply.

Wildcards are allowed in the <target> parameter.

Numeric Replies:

```
        ERR_TOOMANYMATCHES              ERR_NOSUCHSERVER
        RPL_LIST                        RPL_LISTEND
```

Examples:

```
LIST                            ; Command to list all channels.

LIST #twilight_zone,#42         ; Command to list channels
                                #twilight_zone and #42
```

3.2.7 Invite message

```
   Command: INVITE
Parameters: <nickname> <channel>
```

The INVITE command is used to invite a user to a channel. The parameter <nickname> is the nickname of the person to be invited to the target channel <channel>. There is no requirement that the channel the target user is being invited to must exist or be a valid channel. However, if the channel exists, only members of the channel are allowed to invite other users. When the channel has invite-only flag set, only channel operators may issue INVITE command.

Only the user inviting and the user being invited will receive notification of the invitation. Other channel members are not notified. (This is unlike the MODE changes, and is occasionally the source of trouble for users.)

Numeric Replies:

```
        ERR_NEEDMOREPARAMS              ERR_NOSUCHNICK
        ERR_NOTONCHANNEL                ERR_USERONCHANNEL
        ERR_CHANOPRIVSNEEDED
        RPL_INVITING                    RPL_AWAY
```

Examples:

```
:Angel!wings@irc.org INVITE Wiz #Dust

                                ; Message to WiZ when he has been
                                invited by user Angel to channel
                                #Dust

INVITE Wiz #Twilight_Zone       ; Command to invite WiZ to
                                #Twilight_zone
```

3.2.8 Kick command

```
   Command: KICK
Parameters: <channel> *( "," <channel> ) <user> *( "," <user> )
            [<comment>]
```

The KICK command can be used to request the forced removal of a user from a channel. It causes the <user> to PART from the <channel> by force. For the message to be syntactically correct, there MUST be either one channel parameter and multiple user parameter, or as many channel parameters as there are user parameters. If a "comment" is given, this will be sent instead of the default message, the nickname of the user issuing the KICK.

The server MUST NOT send KICK messages with multiple channels or users to clients. This is necessarily to maintain backward compatibility with old client software.

Numeric Replies:

```
        ERR_NEEDMOREPARAMS              ERR_NOSUCHCHANNEL
        ERR_BADCHANMASK                 ERR_CHANOPRIVSNEEDED
        ERR_USERNOTINCHANNEL            ERR_NOTONCHANNEL
```

Examples:

```
   KICK &Melbourne Matthew         ; Command to kick Matthew from
                                   &Melbourne

   KICK #Finnish John :Speaking English
                                   ; Command to kick John from #Finnish
                                   using "Speaking English" as the
                                   reason (comment).

   :WiZ!jto@tolsun.oulu.fi KICK #Finnish John
                                   ; KICK message on channel #Finnish
                                   from WiZ to remove John from channel
```

3.3 Sending messages

The main purpose of the IRC protocol is to provide a base for clients to communicate with each other. PRIVMSG, NOTICE and SQUERY (described in Section 3.5 on Service Query and Commands) are the only messages available which actually perform delivery of a text message from one client to another - the rest just make it possible and try to ensure it happens in a reliable and structured manner.

3.3.1 Private messages

```
      Command: PRIVMSG
   Parameters: <msgtarget> <text to be sent>
```

PRIVMSG is used to send private messages between users, as well as to send messages to channels. <msgtarget> is usually the nickname of the recipient of the message, or a channel name.

The <msgtarget> parameter may also be a host mask (#<mask>) or server mask ($<mask>). In both cases the server will only send the PRIVMSG to those who have a server or host matching the mask. The mask MUST have at least 1 (one) "." in it and no wildcards following the last ".". This requirement exists to prevent people sending messages to "#*" or "$*", which would broadcast to all users. Wildcards are the '*' and '?' characters. This extension to the PRIVMSG command is only available to operators.

Numeric Replies:

```
           ERR_NORECIPIENT                 ERR_NOTEXTTOSEND
           ERR_CANNOTSENDTOCHAN            ERR_NOTOPLEVEL
           ERR_WILDTOPLEVEL                ERR_TOOMANYTARGETS
           ERR_NOSUCHNICK
           RPL_AWAY
```

Examples:

```
:Angel!wings@irc.org PRIVMSG Wiz :Are you receiving this message ?
                                ; Message from Angel to Wiz.

PRIVMSG Angel :yes I'm receiving it !
                                ; Command to send a message to Angel.

PRIVMSG jto@tolsun.oulu.fi :Hello !
                                ; Command to send a message to a user
                                on server tolsun.oulu.fi with
                                username of "jto".

PRIVMSG kalt%millennium.stealth.net@irc.stealth.net :Are you a frog?
                                ; Message to a user on server
                                irc.stealth.net with username of
                                "kalt", and connected from the host
                                millennium.stealth.net.

PRIVMSG kalt%millennium.stealth.net :Do you like cheese?
                                ; Message to a user on the local
                                server with username of "kalt", and
                                connected from the host
                                millennium.stealth.net.

PRIVMSG Wiz!jto@tolsun.oulu.fi :Hello !
                                ; Message to the user with nickname
                                Wiz who is connected from the host
                                tolsun.oulu.fi and has the username
                                "jto".

PRIVMSG $*.fi :Server tolsun.oulu.fi rebooting.
                                ; Message to everyone on a server
                                which has a name matching *.fi.

PRIVMSG #*.edu :NSFNet is undergoing work, expect interruptions
                                ; Message to all users who come from
                                a host which has a name matching
                                *.edu.
```

3.3.2 Notice

```
   Command: NOTICE
Parameters: <msgtarget> <text>
```

The NOTICE command is used similarly to PRIVMSG. The difference between NOTICE and PRIVMSG is that automatic replies MUST NEVER be sent in response to a NOTICE message. This rule applies to servers too - they MUST NOT send any error reply back to the client on receipt of a notice. The object of this rule is to avoid loops between clients automatically sending something in response to something it received.

This command is available to services as well as users.

This is typically used by services, and automatons (clients with either an AI or other interactive program controlling their actions).

See PRIVMSG for more details on replies and examples.

3.4 Server queries and commands

The server query group of commands has been designed to return information about any server which is connected to the network.

In these queries, where a parameter appears as <target>, wildcard masks are usually valid. For each parameter, however, only one query and set of replies is to be generated. In most cases, if a nickname is given, it will mean the server to which the user is connected.

These messages typically have little value for services, it is therefore RECOMMENDED to forbid services from using them.

3.4.1 Motd message

```
   Command: MOTD
Parameters: [ <target> ]
```

The MOTD command is used to get the "Message Of The Day" of the given server, or current server if <target> is omitted.

Wildcards are allowed in the <target> parameter.

```
Numeric Replies:
        RPL_MOTDSTART                   RPL_MOTD
        RPL_ENDOFMOTD                   ERR_NOMOTD
```

3.4.2 Lusers message

```
   Command: LUSERS
Parameters: [ <mask> [ <target> ] ]
```

The LUSERS command is used to get statistics about the size of the IRC network. If no parameter is given, the reply will be about the whole net. If a <mask> is specified, then the reply will only concern the part of the network formed by the servers matching the mask. Finally, if the <target> parameter is specified, the request is forwarded to that server which will generate the reply.

Wildcards are allowed in the <target> parameter.

Numeric Replies:

```
        RPL_LUSERCLIENT                 RPL_LUSEROP
        RPL_LUSERUNKOWN                 RPL_LUSERCHANNELS
        RPL_LUSERME                     ERR_NOSUCHSERVER
```

3.4.3 Version message

```
      Command: VERSION
   Parameters: [ <target> ]
```

The VERSION command is used to query the version of the server program. An optional parameter <target> is used to query the version of the server program which a client is not directly connected to.

Wildcards are allowed in the <target> parameter.

Numeric Replies:

```
        ERR_NOSUCHSERVER                RPL_VERSION
```

Examples:

```
   VERSION tolsun.oulu.fi          ; Command to check the version of
                                   server "tolsun.oulu.fi".
```

3.4.4 Stats message

```
      Command: STATS
   Parameters: [ <query> [ <target> ] ]
```

The stats command is used to query statistics of certain server. If <query> parameter is omitted, only the end of stats reply is sent back.

A query may be given for any single letter which is only checked by the destination server and is otherwise passed on by intermediate servers, ignored and unaltered.

Wildcards are allowed in the <target> parameter.

Except for the ones below, the list of valid queries is implementation dependent. The standard queries below SHOULD be supported by the server:

```
         l - returns a list of the server’s connections, showing how
             long each connection has been established and the
             traffic over that connection in Kbytes and messages for
             each direction;
         m - returns the usage count for each of commands supported
             by the server; commands for which the usage count is
             zero MAY be omitted;
         o - returns a list of configured privileged users,
             operators;
         u - returns a string showing how long the server has been
             up.
```

It is also RECOMMENDED that client and server access configuration be published this way.

Numeric Replies:

```
        ERR_NOSUCHSERVER
        RPL_STATSLINKINFO                RPL_STATSUPTIME
        RPL_STATSCOMMANDS                RPL_STATSOLINE
        RPL_ENDOFSTATS
```

Examples:

```
STATS m                         ; Command to check the command usage
                                for the server you are connected to
```

3.4.5 Links message

```
   Command: LINKS
Parameters: [ [ <remote server> ] <server mask> ]
```

With LINKS, a user can list all servernames, which are known by the server answering the query. The returned list of servers MUST match the mask, or if no mask is given, the full list is returned.

If <remote server> is given in addition to <server mask>, the LINKS command is forwarded to the first server found that matches that name (if any), and that server is then required to answer the query.

Numeric Replies:

```
        ERR_NOSUCHSERVER
        RPL_LINKS                        RPL_ENDOFLINKS
```

Examples:

```
   LINKS *.au                      ; Command to list all servers which
                                   have a name that matches *.au;

   LINKS *.edu *.bu.edu            ; Command to list servers matching
                                   *.bu.edu as seen by the first server
                                   matching *.edu.
```

3.4.6 Time message

```
      Command: TIME
   Parameters: [ <target> ]
```

The time command is used to query local time from the specified server. If the <target> parameter is not given, the server receiving the command must reply to the query.

Wildcards are allowed in the <target> parameter.

Numeric Replies:

```
           ERR_NOSUCHSERVER              RPL_TIME
```

Examples:

```
   TIME tolsun.oulu.fi             ; check the time on the server
                                   "tolson.oulu.fi"
```

3.4.7 Connect message

```
      Command: CONNECT
   Parameters: <target server> <port> [ <remote server> ]
```

The CONNECT command can be used to request a server to try to establish a new connection to another server immediately. CONNECT is a privileged command and SHOULD be available only to IRC Operators. If a <remote server> is given and its mask doesn’t match name of the parsing server, the CONNECT attempt is sent to the first match of remote server. Otherwise the CONNECT attempt is made by the server processing the request.

The server receiving a remote CONNECT command SHOULD generate a WALLOPS message describing the source and target of the request.

Numeric Replies:

```
           ERR_NOSUCHSERVER              ERR_NOPRIVILEGES
           ERR_NEEDMOREPARAMS
```

Examples:

```
CONNECT tolsun.oulu.fi 6667     ; Command to attempt to connect local
                                server to tolsun.oulu.fi on port 6667
```

3.4.8 Trace message

```
   Command: TRACE
Parameters: [ <target> ]
```

TRACE command is used to find the route to specific server and information about its peers. Each server that processes this command MUST report to the sender about it. The replies from pass-through links form a chain, which shows route to destination. After sending this reply back, the query MUST be sent to the next server until given <target> server is reached.

TRACE command is used to find the route to specific server. Each server that processes this message MUST tell the sender about it by sending a reply indicating it is a pass-through link, forming a chain of replies. After sending this reply back, it MUST then send the TRACE message to the next server until given server is reached. If the <target> parameter is omitted, it is RECOMMENDED that TRACE command sends a message to the sender telling which servers the local server has direct connection to.

If the destination given by <target> is an actual server, the destination server is REQUIRED to report all servers, services and operators which are connected to it; if the command was issued by an operator, the server MAY also report all users which are connected to it. If the destination given by <target> is a nickname, then only a reply for that nickname is given. If the <target> parameter is omitted, it is RECOMMENDED that the TRACE command is parsed as targeted to the processing server.

Wildcards are allowed in the <target> parameter.

Numeric Replies:

        ERR_NOSUCHSERVER

   If the TRACE message is destined for another server, all intermediate servers must return a RPL_TRACELINK reply to indicate that the TRACE passed through it and where it is going next.

        RPL_TRACELINK

A TRACE reply may be composed of any number of the following numeric replies.

```
        RPL_TRACECONNECTING           RPL_TRACEHANDSHAKE
        RPL_TRACEUNKNOWN              RPL_TRACEOPERATOR
        RPL_TRACEUSER                 RPL_TRACESERVER
        RPL_TRACESERVICE              RPL_TRACENEWTYPE
        RPL_TRACECLASS                RPL_TRACELOG
        RPL_TRACEEND
```

Examples:

```
TRACE *.oulu.fi                 ; TRACE to a server matching
                                *.oulu.fi
```

3.4.9 Admin command

```
   Command: ADMIN
Parameters: [ <target> ]
```

The admin command is used to find information about the administrator of the given server, or current server if <target> parameter is omitted. Each server MUST have the ability to forward ADMIN messages to other servers.

Wildcards are allowed in the <target> parameter.

Numeric Replies:

```
        ERR_NOSUCHSERVER
        RPL_ADMINME                   RPL_ADMINLOC1
        RPL_ADMINLOC2                 RPL_ADMINEMAIL
```

Examples:

```
ADMIN tolsun.oulu.fi            ; request an ADMIN reply from
                                tolsun.oulu.fi

ADMIN syrk                      ; ADMIN request for the server to
                                which the user syrk is connected
```

3.4.10 Info command

```
      Command: INFO
   Parameters: [ <target> ]
```

The INFO command is REQUIRED to return information describing the server: its version, when it was compiled, the patchlevel, when it was started, and any other miscellaneous information which may be considered to be relevant.

Wildcards are allowed in the <target> parameter.

Numeric Replies:

```
           ERR_NOSUCHSERVER
           RPL_INFO                        RPL_ENDOFINFO
```

Examples:

```
   INFO csd.bu.edu                 ; request an INFO reply from
                                   csd.bu.edu

   INFO Angel                      ; request info from the server that
                                   Angel is connected to.
```

3.5 Service Query and Commands

The service query group of commands has been designed to return information about any service which is connected to the network.

3.5.1 Servlist message

```
      Command: SERVLIST
   Parameters: [ <mask> [ <type> ] ]
```

The SERVLIST command is used to list services currently connected to the network and visible to the user issuing the command. The optional parameters may be used to restrict the result of the query (to matching services names, and services type).

Numeric Replies:

```
           RPL_SERVLIST                    RPL_SERVLISTEND
```

3.5.2 Squery

```
      Command: SQUERY
   Parameters: <servicename> <text>
```

The SQUERY command is used similarly to PRIVMSG. The only difference is that the recipient MUST be a service. This is the only way for a text message to be delivered to a service.

See PRIVMSG for more details on replies and example.

Examples:

```
   SQUERY irchelp :HELP privmsg
                                   ; Message to the service with
                                   nickname irchelp.

   SQUERY dict@irc.fr :fr2en blaireau
                                   ; Message to the service with name
                                   dict@irc.fr.
```

3.6 User based queries

User queries are a group of commands which are primarily concerned with finding details on a particular user or group users. When using wildcards with any of these commands, if they match, they will only return information on users who are ’visible’ to you. The visibility of a user is determined as a combination of the user’s mode and the common set of channels you are both on.

Although services SHOULD NOT be using this class of message, they are allowed to.

3.6.1 Who query

```
      Command: WHO
   Parameters: [ <mask> [ "o" ] ]
```

The WHO command is used by a client to generate a query which returns a list of information which ’matches’ the <mask> parameter given by the client. In the absence of the <mask> parameter, all visible (users who aren’t invisible (user mode +i) and who don’t have a common channel with the requesting client) are listed. The same result can be achieved by using a <mask> of "0" or any wildcard which will end up matching every visible user.

The <mask> passed to WHO is matched against users’ host, server, real name and nickname if the channel <mask> cannot be found.

If the "o" parameter is passed only operators are returned according to the <mask> supplied.

Numeric Replies:

```
        ERR_NOSUCHSERVER
        RPL_WHOREPLY                  RPL_ENDOFWHO
```

Examples:

```
WHO *.fi                        ; Command to list all users who match
                                against "*.fi".

WHO jto* o                      ; Command to list all users with a
                                match against "jto*" if they are an
                                operator.
```

3.6.2 Whois query

```
   Command: WHOIS
Parameters: [ <target> ] <mask> *( "," <mask> )
```

This command is used to query information about particular user. The server will answer this command with several numeric messages indicating different statuses of each user which matches the mask (if you are entitled to see them). If no wildcard is present in the <mask>, any information about that nick which you are allowed to see is presented.

If the <target> parameter is specified, it sends the query to a specific server. It is useful if you want to know how long the user in question has been idle as only local server (i.e., the server the user is directly connected to) knows that information, while everything else is globally known.

Wildcards are allowed in the <target> parameter.

Numeric Replies:

```
        ERR_NOSUCHSERVER              ERR_NONICKNAMEGIVEN
        RPL_WHOISUSER                 RPL_WHOISCHANNELS
        RPL_WHOISCHANNELS             RPL_WHOISSERVER
        RPL_AWAY                      RPL_WHOISOPERATOR
        RPL_WHOISIDLE                 ERR_NOSUCHNICK
        RPL_ENDOFWHOIS
```

Examples:

```
WHOIS wiz                       ; return available user information
                                about nick WiZ

WHOIS eff.org trillian          ; ask server eff.org for user
                                information  about trillian
```

3.6.3 Whowas

```
   Command: WHOWAS
Parameters: <nickname> *( "," <nickname> ) [ <count> [ <target> ] ]
```

Whowas asks for information about a nickname which no longer exists. This may either be due to a nickname change or the user leaving IRC. In response to this query, the server searches through its nickname history, looking for any nicks which are lexically the same (no wild card matching here). The history is searched backward, returning the most recent entry first. If there are multiple entries, up to <count> replies will be returned (or all of them if no <count> parameter is given). If a non-positive number is passed as being <count>, then a full search is done.

Wildcards are allowed in the <target> parameter.

Numeric Replies:

```
        ERR_NONICKNAMEGIVEN           ERR_WASNOSUCHNICK
        RPL_WHOWASUSER                RPL_WHOISSERVER
        RPL_ENDOFWHOWAS
```

Examples:

```
WHOWAS Wiz                      ; return all information in the nick
                                history about nick "WiZ";

WHOWAS Mermaid 9                ; return at most, the 9 most recent
                                entries in the nick history for
                                "Mermaid";

WHOWAS Trillian 1 *.edu         ; return the most recent history for
                                "Trillian" from the first server
                                found to match "*.edu".
```

3.7 Miscellaneous messages

Messages in this category do not fit into any of the above categories but are nonetheless still a part of and REQUIRED by the protocol.

3.7.1 Kill message

```
   Command: KILL
Parameters: <nickname> <comment>
```

The KILL command is used to cause a client-server connection to be closed by the server which has the actual connection. Servers generate KILL messages on nickname collisions. It MAY also be available available to users who have the operator status.

Clients which have automatic reconnect algorithms effectively make this command useless since the disconnection is only brief. It does however break the flow of data and can be used to stop large amounts of 'flooding' from abusive users or accidents. Abusive users usually don't care as they will reconnect promptly and resume their abusive behaviour. To prevent this command from being abused, any user may elect to receive KILL messages generated for others to keep an 'eye' on would be trouble spots.

In an arena where nicknames are REQUIRED to be globally unique at all times, KILL messages are sent whenever 'duplicates' are detected (that is an attempt to register two users with the same nickname) in the hope that both of them will disappear and only 1 reappear.

When a client is removed as the result of a KILL message, the server SHOULD add the nickname to the list of unavailable nicknames in an attempt to avoid clients to reuse this name immediately which is usually the pattern of abusive behaviour often leading to useless "KILL loops". See the "IRC Server Protocol" document [IRC-SERVER] for more information on this procedure.

The comment given MUST reflect the actual reason for the KILL. For server-generated KILLs it usually is made up of details concerning the origins of the two conflicting nicknames. For users it is left up to them to provide an adequate reason to satisfy others who see it. To prevent/discourage fake KILLs from being generated to hide the identify of the KILLer, the comment also shows a 'kill-path' which is updated by each server it passes through, each prepending its name to the path.

Numeric Replies:

```
        ERR_NOPRIVILEGES              ERR_NEEDMOREPARAMS
        ERR_NOSUCHNICK                ERR_CANTKILLSERVER
```

NOTE:
It is RECOMMENDED that only Operators be allowed to kill other users with KILL command. This command has been the subject of many controversies over the years, and along with the above recommendation, it is also widely recognized that not even operators should be allowed to kill users on remote servers.

3.7.2 Ping message

```
      Command: PING
   Parameters: <server1> [ <server2> ]
```

The PING command is used to test the presence of an active client or server at the other end of the connection. Servers send a PING message at regular intervals if no other activity detected coming from a connection. If a connection fails to respond to a PING message within a set amount of time, that connection is closed. A PING message MAY be sent even if the connection is active.

When a PING message is received, the appropriate PONG message MUST be sent as reply to <server1> (server which sent the PING message out) as soon as possible. If the <server2> parameter is specified, it represents the target of the ping, and the message gets forwarded there.

Numeric Replies:

```
           ERR_NOORIGIN                  ERR_NOSUCHSERVER
```

Examples:

```
   PING tolsun.oulu.fi             ; Command to send a PING message to
                                   server

   PING WiZ tolsun.oulu.fi         ; Command from WiZ to send a PING
                                   message to server "tolsun.oulu.fi"

   PING :irc.funet.fi              ; Ping message sent by server
                                   "irc.funet.fi"
```

3.7.3 Pong message

```
      Command: PONG
   Parameters: <server> [ <server2> ]
```

PONG message is a reply to ping message. If parameter <server2> is given, this message MUST be forwarded to given target. The <server> parameter is the name of the entity who has responded to PING message and generated this message.

Numeric Replies:

```
           ERR_NOORIGIN                  ERR_NOSUCHSERVER
```

Example:

```
   PONG csd.bu.edu tolsun.oulu.fi  ; PONG message from csd.bu.edu to
                                   tolsun.oulu.fi
```

3.7.4 Error

```
      Command: ERROR
   Parameters: <error message>
```

The ERROR command is for use by servers when reporting a serious or fatal error to its peers. It may also be sent from one server to another but MUST NOT be accepted from any normal unknown clients.

Only an ERROR message SHOULD be used for reporting errors which occur with a server-to-server link. An ERROR message is sent to the server at the other end (which reports it to appropriate local users and logs) and to appropriate local users and logs. It is not to be passed onto any other servers by a server if it is received from a server.

The ERROR message is also used before terminating a client connection.

When a server sends a received ERROR message to its operators, the message SHOULD be encapsulated inside a NOTICE message, indicating that the client was not responsible for the error.

Numerics:

None.

Examples:

```
ERROR :Server *.fi already exists ; ERROR message to the other server
                                which caused this error.

NOTICE WiZ :ERROR from csd.bu.edu -- Server *.fi already exists
                                ; Same ERROR message as above but
                                sent to user WiZ on the other server.
```

4. Optional features

This section describes OPTIONAL messages. They are not required in a working server implementation of the protocol described herein. In the absence of the feature, an error reply message MUST be generated or an unknown command error. If the message is destined for another server to answer then it MUST be passed on (elementary parsing REQUIRED) The allocated numerics for this are listed with the messages below.

From this section, only the USERHOST and ISON messages are available to services.

4.1 Away

```
   Command: AWAY
Parameters: [ <text> ]
```

With the AWAY command, clients can set an automatic reply string for any PRIVMSG commands directed at them (not to a channel they are on). The server sends an automatic reply to the client sending the PRIVMSG command. The only replying server is the one to which the sending client is connected to.

The AWAY command is used either with one parameter, to set an AWAY message, or with no parameters, to remove the AWAY message.

Because of its high cost (memory and bandwidth wise), the AWAY message SHOULD only be used for client-server communication. A server MAY choose to silently ignore AWAY messages received from other servers. To update the away status of a client across servers, the user mode ’a’ SHOULD be used instead. (See Section 3.1.5)

Numeric Replies:

```
        RPL_UNAWAY                      RPL_NOWAWAY
```

Example:

```
AWAY :Gone to lunch.  Back in 5 ; Command to set away message to
                                "Gone to lunch.  Back in 5".
```

4.2 Rehash message

```
   Command: REHASH
Parameters: None
```

The rehash command is an administrative command which can be used by an operator to force the server to re-read and process its configuration file.

Numeric Replies:

```
        RPL_REHASHING                 ERR_NOPRIVILEGES
```

Example:

```
REHASH                          ; message from user with operator
                                status to server asking it to reread
                                its configuration file.
```

4.3 Die message

```
   Command: DIE
Parameters: None
```

An operator can use the DIE command to shutdown the server. This message is optional since it may be viewed as a risk to allow arbitrary people to connect to a server as an operator and execute this command.

The DIE command MUST always be fully processed by the server to which the sending client is connected and MUST NOT be passed onto other connected servers.

Numeric Replies:

```
        ERR_NOPRIVILEGES
```

Example:

```
DIE                             ; no parameters required.
```

4.4 Restart message

```
      Command: RESTART
   Parameters: None
```

An operator can use the restart command to force the server to restart itself. This message is optional since it may be viewed as a risk to allow arbitrary people to connect to a server as an operator and execute this command, causing (at least) a disruption to service.

The RESTART command MUST always be fully processed by the server to which the sending client is connected and MUST NOT be passed onto other connected servers.

Numeric Replies:

```
           ERR_NOPRIVILEGES
```

Example:

```
   RESTART                         ; no parameters required.
```

4.5 Summon message

```
      Command: SUMMON
   Parameters: <user> [ <target> [ <channel> ] ]
```

The SUMMON command can be used to give users who are on a host running an IRC server a message asking them to please join IRC. This message is only sent if the target server (a) has SUMMON enabled, (b) the user is logged in and (c) the server process can write to the user’s tty (or similar).

If no <server> parameter is given it tries to summon <user> from the server the client is connected to is assumed as the target.

If summon is not enabled in a server, it MUST return the ERR_SUMMONDISABLED numeric.

Numeric Replies:

```
           ERR_NORECIPIENT               ERR_FILEERROR
           ERR_NOLOGIN                   ERR_NOSUCHSERVER
           ERR_SUMMONDISABLED            RPL_SUMMONING
```

Examples:

```
SUMMON jto                      ; summon user jto on the server's
                                host

SUMMON jto tolsun.oulu.fi       ; summon user jto on the host which a
                                server named "tolsun.oulu.fi" is
                                running.
```

4.6 Users

```
   Command: USERS
Parameters: [ <target> ]
```

The USERS command returns a list of users logged into the server in a format similar to the UNIX commands who(1), rusers(1) and finger(1). If disabled, the correct numeric MUST be returned to indicate this.

Because of the security implications of such a command, it SHOULD be disabled by default in server implementations. Enabling it SHOULD require recompiling the server or some equivalent change rather than simply toggling an option and restarting the server. The procedure to enable this command SHOULD also include suitable large comments.

Numeric Replies:

```
        ERR_NOSUCHSERVER              ERR_FILEERROR
        RPL_USERSSTART                RPL_USERS
        RPL_NOUSERS                   RPL_ENDOFUSERS
        ERR_USERSDISABLED
```

Disabled Reply:

```
        ERR_USERSDISABLED
```

Example:

```
USERS eff.org                   ; request a list of users logged in
                                on server eff.org
```

4.7 Operwall message

```
   Command: WALLOPS
Parameters: <Text to be sent>
```

The WALLOPS command is used to send a message to all currently connected users who have set the 'w' user mode for themselves. (See Section 3.1.5 "User modes").

After implementing WALLOPS as a user command it was found that it was often and commonly abused as a means of sending a message to a lot of people. Due to this, it is RECOMMENDED that the implementation of WALLOPS allows and recognizes only servers as the originators of WALLOPS.

Numeric Replies:

```
        ERR_NEEDMOREPARAMS
```

Example:

```
:csd.bu.edu WALLOPS :Connect '*.uiuc.edu 6667' from Joshua ; WALLOPS
                                message from csd.bu.edu announcing a
                                CONNECT message it received from
                                Joshua and acted upon.
```

4.8 Userhost message

```
   Command: USERHOST
Parameters: <nickname> *( SPACE <nickname> )
```

The USERHOST command takes a list of up to 5 nicknames, each separated by a space character and returns a list of information about each nickname that it found. The returned list has each reply separated by a space.

Numeric Replies:

```
        RPL_USERHOST                  ERR_NEEDMOREPARAMS
```

Example:

```
USERHOST Wiz Michael syrk       ; USERHOST request for information on
                                nicks "Wiz", "Michael", and "syrk"

:ircd.stealth.net 302 yournick :syrk=+syrk@millennium.stealth.net
                                ; Reply for user syrk
```

4.9 Ison message

```
   Command: ISON
Parameters: <nickname> *( SPACE <nickname> )
```

The ISON command was implemented to provide a quick and efficient means to get a response about whether a given nickname was currently on IRC. ISON only takes one (1) type of parameter: a space-separated list of nicks. For each nickname in the list that is present, the server adds that to its reply string. Thus the reply string may return empty (none of the given nicks are present), an exact copy of the parameter string (all of them present) or any other subset of the set of nicks given in the parameter. The only limit on the number of nicks that may be checked is that the combined length MUST NOT be too large as to cause the server to chop it off so it fits in 512 characters.

ISON is only processed by the server local to the client sending the command and thus not passed onto other servers for further processing.

Numeric Replies:

```
        RPL_ISON                      ERR_NEEDMOREPARAMS
```

Example:

```
ISON phone trillian WiZ jarlek Avalon Angel Monstah syrk
                                ; Sample ISON request for 7 nicks.
```

5. Replies

The following is a list of numeric replies which are generated in response to the commands given above. Each numeric is given with its number, name and reply string.

5.1 Command responses

Numerics in the range from 001 to 099 are used for client-server connections only and should never travel between servers. Replies generated in the response to commands are found in the range from 200 to 399.

```
    001    RPL_WELCOME
           "Welcome to the Internet Relay Network
            <nick>!<user>@<host>"
    002    RPL_YOURHOST
           "Your host is <servername>, running version <ver>"
    003    RPL_CREATED
           "This server was created <date>"
    004    RPL_MYINFO
           "<servername> <version> <available user modes>
            <available channel modes>"
```

- The server sends Replies 001 to 004 to a user upon successful registration.

```
       005    RPL_BOUNCE
              "Try server <server name>, port <port number>"

         - Sent by the server to a user to suggest an alternative
           server.  This is often used when the connection is
           refused because the server is already full.

       302    RPL_USERHOST
              ":*1<reply> *( " " <reply> )"

         - Reply format used by USERHOST to list replies to
           the query list.  The reply string is composed as
           follows:

           reply = nickname [ "*" ] "=" ( "+" / "-" ) hostname

           The '*' indicates whether the client has registered
           as an Operator.  The '-' or '+' characters represent
           whether the client has set an AWAY message or not
           respectively.

       303    RPL_ISON
              ":*1<nick> *( " " <nick> )"

         - Reply format used by ISON to list replies to the
           query list.

       301    RPL_AWAY
              "<nick> :<away message>"
       305    RPL_UNAWAY
              ":You are no longer marked as being away"
       306    RPL_NOWAWAY
              ":You have been marked as being away"

         - These replies are used with the AWAY command (if
           allowed).  RPL_AWAY is sent to any client sending a
           PRIVMSG to a client which is away.  RPL_AWAY is only
           sent by the server to which the client is connected.
           Replies RPL_UNAWAY and RPL_NOWAWAY are sent when the
           client removes and sets an AWAY message.

       311    RPL_WHOISUSER
              "<nick> <user> <host> * :<real name>"
       312    RPL_WHOISSERVER
              "<nick> <server> :<server info>"
       313    RPL_WHOISOPERATOR
              "<nick> :is an IRC operator"
       317    RPL_WHOISIDLE
              "<nick> <integer> :seconds idle"
       318    RPL_ENDOFWHOIS
              "<nick> :End of WHOIS list"
       319    RPL_WHOISCHANNELS
              "<nick> :*( ( "@" / "+" ) <channel> " " )"

         - Replies 311 - 313, 317 - 319 are all replies
           generated in response to a WHOIS message.  Given that
           there are enough parameters present, the answering
           server MUST either formulate a reply out of the above
           numerics (if the query nick is found) or return an
           error reply.  The '*' in RPL_WHOISUSER is there as
           the literal character and not as a wild card.  For
           each reply set, only RPL_WHOISCHANNELS may appear
           more than once (for long lists of channel names).
           The '@' and '+' characters next to the channel name
           indicate whether a client is a channel operator or
           has been granted permission to speak on a moderated
           channel.  The RPL_ENDOFWHOIS reply is used to mark
           the end of processing a WHOIS message.

       314    RPL_WHOWASUSER
              "<nick> <user> <host> * :<real name>"
       369    RPL_ENDOFWHOWAS
              "<nick> :End of WHOWAS"

         - When replying to a WHOWAS message, a server MUST use
           the replies RPL_WHOWASUSER, RPL_WHOISSERVER or
           ERR_WASNOSUCHNICK for each nickname in the presented
           list.  At the end of all reply batches, there MUST
           be RPL_ENDOFWHOWAS (even if there was only one reply
           and it was an error).

       321    RPL_LISTSTART
              Obsolete. Not used.

       322    RPL_LIST
              "<channel> <# visible> :<topic>"
       323    RPL_LISTEND
              ":End of LIST"

         - Replies RPL_LIST, RPL_LISTEND mark the actual replies
           with data and end of the server's response to a LIST
           command.  If there are no channels available to return,
           only the end reply MUST be sent.
       325    RPL_UNIQOPIS
              "<channel> <nickname>"

       324    RPL_CHANNELMODEIS
              "<channel> <mode> <mode params>"

       331    RPL_NOTOPIC
              "<channel> :No topic is set"
       332    RPL_TOPIC
              "<channel> :<topic>"

         - When sending a TOPIC message to determine the
           channel topic, one of two replies is sent.  If
           the topic is set, RPL_TOPIC is sent back else
           RPL_NOTOPIC.

       341    RPL_INVITING
              "<channel> <nick>"

         - Returned by the server to indicate that the
           attempted INVITE message was successful and is
           being passed onto the end client.

       342    RPL_SUMMONING
              "<user> :Summoning user to IRC"

         - Returned by a server answering a SUMMON message to
           indicate that it is summoning that user.

       346    RPL_INVITELIST
              "<channel> <invitemask>"
       347    RPL_ENDOFINVITELIST
              "<channel> :End of channel invite list"

         - When listing the 'invitations masks' for a given channel,
           a server is required to send the list back using the
           RPL_INVITELIST and RPL_ENDOFINVITELIST messages.  A
           separate RPL_INVITELIST is sent for each active mask.
           After the masks have been listed (or if none present) a
           RPL_ENDOFINVITELIST MUST be sent.

       348    RPL_EXCEPTLIST
              "<channel> <exceptionmask>"
       349    RPL_ENDOFEXCEPTLIST
              "<channel> :End of channel exception list"
```

```
          - When listing the 'exception masks' for a given channel,
            a server is required to send the list back using the
            RPL_EXCEPTLIST and RPL_ENDOFEXCEPTLIST messages.  A
            separate RPL_EXCEPTLIST is sent for each active mask.
            After the masks have been listed (or if none present)
            a RPL_ENDOFEXCEPTLIST MUST be sent.

       351    RPL_VERSION
              "<version>.<debuglevel> <server> :<comments>"

          - Reply by the server showing its version details.
            The <version> is the version of the software being
            used (including any patchlevel revisions) and the
            <debuglevel> is used to indicate if the server is
            running in "debug mode".

            The "comments" field may contain any comments about
            the version or further version details.

       352    RPL_WHOREPLY
              "<channel> <user> <host> <server> <nick>
              ( "H" / "G" > ["*"] [ ( "@" / "+" ) ]
              :<hopcount> <real name>"

       315    RPL_ENDOFWHO
              "<name> :End of WHO list"

          - The RPL_WHOREPLY and RPL_ENDOFWHO pair are used
            to answer a WHO message.  The RPL_WHOREPLY is only
            sent if there is an appropriate match to the WHO
            query.  If there is a list of parameters supplied
            with a WHO message, a RPL_ENDOFWHO MUST be sent
            after processing each list item with <name> being
            the item.

       353    RPL_NAMREPLY
              "( "=" / "*" / "@" ) <channel>
               :[ "@" / "+" ] <nick> *( " " [ "@" / "+" ] <nick> )
          - "@" is used for secret channels, "*" for private
            channels, and "=" for others (public channels).

       366    RPL_ENDOFNAMES
              "<channel> :End of NAMES list"

          - To reply to a NAMES message, a reply pair consisting
            of RPL_NAMREPLY and RPL_ENDOFNAMES is sent by the
            server back to the client.  If there is no channel
            found as in the query, then only RPL_ENDOFNAMES is
```

```
           returned.  The exception to this is when a NAMES
           message is sent with no parameters and all visible
           channels and contents are sent back in a series of
           RPL_NAMEREPLY messages with a RPL_ENDOFNAMES to mark
           the end.

       364    RPL_LINKS
              "<mask> <server> :<hopcount> <server info>"
       365    RPL_ENDOFLINKS
              "<mask> :End of LINKS list"

         - In replying to the LINKS message, a server MUST send
           replies back using the RPL_LINKS numeric and mark the
           end of the list using an RPL_ENDOFLINKS reply.

       367    RPL_BANLIST
              "<channel> <banmask>"
       368    RPL_ENDOFBANLIST
              "<channel> :End of channel ban list"

         - When listing the active 'bans' for a given channel,
           a server is required to send the list back using the
           RPL_BANLIST and RPL_ENDOFBANLIST messages.  A separate
           RPL_BANLIST is sent for each active banmask.  After the
           banmasks have been listed (or if none present) a
           RPL_ENDOFBANLIST MUST be sent.

       371    RPL_INFO
              ":<string>"
       374    RPL_ENDOFINFO
              ":End of INFO list"

         - A server responding to an INFO message is required to
           send all its 'info' in a series of RPL_INFO messages
           with a RPL_ENDOFINFO reply to indicate the end of the
           replies.

       375    RPL_MOTDSTART
              ":- <server> Message of the day - "
       372    RPL_MOTD
              ":- <text>"
       376    RPL_ENDOFMOTD
              ":End of MOTD command"

         - When responding to the MOTD message and the MOTD file
           is found, the file is displayed line by line, with
           each line no longer than 80 characters, using
           RPL_MOTD format replies.  These MUST be surrounded
           by a RPL_MOTDSTART (before the RPL_MOTDs) and an
           RPL_ENDOFMOTD (after).

       381    RPL_YOUREOPER
              ":You are now an IRC operator"

         - RPL_YOUREOPER is sent back to a client which has
           just successfully issued an OPER message and gained
           operator status.

       382    RPL_REHASHING
              "<config file> :Rehashing"

         - If the REHASH option is used and an operator sends
           a REHASH message, an RPL_REHASHING is sent back to
           the operator.

       383    RPL_YOURESERVICE
              "You are service <servicename>"

         - Sent by the server to a service upon successful
           registration.

       391    RPL_TIME
              "<server> :<string showing server’s local time>"

         - When replying to the TIME message, a server MUST send
           the reply using the RPL_TIME format above.  The string
           showing the time need only contain the correct day and
           time there.  There is no further requirement for the
           time string.

       392    RPL_USERSSTART
              ":UserID   Terminal  Host"
       393    RPL_USERS
              ":<username> <ttyline> <hostname>"
       394    RPL_ENDOFUSERS
              ":End of users"
       395    RPL_NOUSERS
              ":Nobody logged in"

         - If the USERS message is handled by a server, the
           replies RPL_USERSTART, RPL_USERS, RPL_ENDOFUSERS and
           RPL_NOUSERS are used.  RPL_USERSSTART MUST be sent
           first, following by either a sequence of RPL_USERS
           or a single RPL_NOUSER.  Following this is
           RPL_ENDOFUSERS.
       200    RPL_TRACELINK
              "Link <version & debug level> <destination>
               <next server> V<protocol version>
               <link uptime in seconds> <backstream sendq>
               <upstream sendq>"
       201    RPL_TRACECONNECTING
              "Try. <class> <server>"
       202    RPL_TRACEHANDSHAKE
              "H.S. <class> <server>"
       203    RPL_TRACEUNKNOWN
              "???? <class> [<client IP address in dot form>]"
       204    RPL_TRACEOPERATOR
              "Oper <class> <nick>"
       205    RPL_TRACEUSER
              "User <class> <nick>"
       206    RPL_TRACESERVER
              "Serv <class> <int>S <int>C <server>
               <nick!user|*!*>@<host|server> V<protocol version>"
       207    RPL_TRACESERVICE
              "Service <class> <name> <type> <active type>"
       208    RPL_TRACENEWTYPE
              "<newtype> 0 <client name>"
       209    RPL_TRACECLASS
              "Class <class> <count>"
       210    RPL_TRACERECONNECT
              Unused.
       261    RPL_TRACELOG
              "File <logfile> <debug level>"
       262    RPL_TRACEEND
              "<server name> <version & debug level> :End of TRACE"
```

- The RPL_TRACE* are all returned by the server in response to the TRACE message. How many are returned is dependent on the TRACE message and whether it was sent by an operator or not. There is no predefined order for which occurs first. Replies RPL_TRACEUNKNOWN, RPL_TRACECONNECTING and RPL_TRACEHANDSHAKE are all used for connections which have not been fully established and are either unknown, still attempting to connect or in the process of completing the ’server handshake’. RPL_TRACELINK is sent by any server which handles a TRACE message and has to pass it on to another server. The list of RPL_TRACELINKs sent in response to a TRACE command traversing the IRC network should reflect the actual connectivity of the servers themselves along that path.

```
           RPL_TRACENEWTYPE is to be used for any connection
           which does not fit in the other categories but is
           being displayed anyway.
           RPL_TRACEEND is sent to indicate the end of the list.

       211    RPL_STATSLINKINFO
              "<linkname> <sendq> <sent messages>
               <sent Kbytes> <received messages>
               <received Kbytes> <time open>"

         - reports statistics on a connection.  <linkname>
           identifies the particular connection, <sendq> is
           the amount of data that is queued and waiting to be
           sent <sent messages> the number of messages sent,
           and <sent Kbytes> the amount of data sent, in
           Kbytes. <received messages> and <received Kbytes>
           are the equivalent of <sent messages> and <sent
           Kbytes> for received data, respectively.  <time
           open> indicates how long ago the connection was
           opened, in seconds.

       212    RPL_STATSCOMMANDS
              "<command> <count> <byte count> <remote count>"

         - reports statistics on commands usage.

       219    RPL_ENDOFSTATS
              "<stats letter> :End of STATS report"

       242    RPL_STATSUPTIME
              ":Server Up %d days %d:%02d:%02d"

         - reports the server uptime.

       243    RPL_STATSOLINE
              "O <hostmask> * <name>"

         - reports the allowed hosts from where user may become IRC
           operators.

       221    RPL_UMODEIS
              "<user mode string>"

         - To answer a query about a client’s own mode,
           RPL_UMODEIS is sent back.

       234    RPL_SERVLIST
              "<name> <server> <mask> <type> <hopcount> <info>"
       235    RPL_SERVLISTEND
              "<mask> <type> :End of service listing"

         - When listing services in reply to a SERVLIST message,
           a server is required to send the list back using the
           RPL_SERVLIST and RPL_SERVLISTEND messages.  A separate
           RPL_SERVLIST is sent for each service.  After the
           services have been listed (or if none present) a
           RPL_SERVLISTEND MUST be sent.

       251    RPL_LUSERCLIENT
              ":There are <integer> users and <integer>
               services on <integer> servers"
       252    RPL_LUSEROP
              "<integer> :operator(s) online"
       253    RPL_LUSERUNKNOWN
              "<integer> :unknown connection(s)"
       254    RPL_LUSERCHANNELS
              "<integer> :channels formed"
       255    RPL_LUSERME
              ":I have <integer> clients and <integer>
                servers"

         - In processing an LUSERS message, the server
           sends a set of replies from RPL_LUSERCLIENT,
           RPL_LUSEROP, RPL_USERUNKNOWN,
           RPL_LUSERCHANNELS and RPL_LUSERME.  When
           replying, a server MUST send back
           RPL_LUSERCLIENT and RPL_LUSERME.  The other
           replies are only sent back if a non-zero count
           is found for them.

       256    RPL_ADMINME
              "<server> :Administrative info"
       257    RPL_ADMINLOC1
              ":<admin info>"
       258    RPL_ADMINLOC2
              ":<admin info>"
       259    RPL_ADMINEMAIL
              ":<admin info>"

         - When replying to an ADMIN message, a server
           is expected to use replies RPL_ADMINME
           through to RPL_ADMINEMAIL and provide a text
           message with each.  For RPL_ADMINLOC1 a
           description of what city, state and country
           the server is in is expected, followed by
           details of the institution (RPL_ADMINLOC2)
           and finally the administrative contact for the
           server (an email address here is REQUIRED)
           in RPL_ADMINEMAIL.

       263    RPL_TRYAGAIN
              "<command> :Please wait a while and try again."

         - When a server drops a command without processing it,
           it MUST use the reply RPL_TRYAGAIN to inform the
           originating client.
```

5.2 Error Replies

```
       Error replies are found in the range from 400 to 599.

       401    ERR_NOSUCHNICK
              "<nickname> :No such nick/channel"

          - Used to indicate the nickname parameter supplied to a
            command is currently unused.

       402    ERR_NOSUCHSERVER
              "<server name> :No such server"

         - Used to indicate the server name given currently
           does not exist.

       403    ERR_NOSUCHCHANNEL
              "<channel name> :No such channel"

         - Used to indicate the given channel name is invalid.

       404    ERR_CANNOTSENDTOCHAN
              "<channel name> :Cannot send to channel"

         - Sent to a user who is either (a) not on a channel
           which is mode +n or (b) not a chanop (or mode +v) on
           a channel which has mode +m set or where the user is
           banned and is trying to send a PRIVMSG message to
           that channel.

       405    ERR_TOOMANYCHANNELS
              "<channel name> :You have joined too many channels"

         - Sent to a user when they have joined the maximum
           number of allowed channels and they try to join
           another channel.
       406    ERR_WASNOSUCHNICK
              "<nickname> :There was no such nickname"

         - Returned by WHOWAS to indicate there is no history
           information for that nickname.

       407    ERR_TOOMANYTARGETS
              "<target> :<error code> recipients. <abort message>"

         - Returned to a client which is attempting to send a
           PRIVMSG/NOTICE using the user@host destination format
           and for a user@host which has several occurrences.

         - Returned to a client which trying to send a
           PRIVMSG/NOTICE to too many recipients.

         - Returned to a client which is attempting to JOIN a safe
           channel using the shortname when there are more than one
           such channel.

       408    ERR_NOSUCHSERVICE
              "<service name> :No such service"

         - Returned to a client which is attempting to send a SQUERY
           to a service which does not exist.

       409    ERR_NOORIGIN
              ":No origin specified"

         - PING or PONG message missing the originator parameter.

       411    ERR_NORECIPIENT
              ":No recipient given (<command>)"
       412    ERR_NOTEXTTOSEND
              ":No text to send"
       413    ERR_NOTOPLEVEL
              "<mask> :No toplevel domain specified"
       414    ERR_WILDTOPLEVEL
              "<mask> :Wildcard in toplevel domain"
       415    ERR_BADMASK
              "<mask> :Bad Server/host mask"

         - 412 - 415 are returned by PRIVMSG to indicate that
           the message wasn't delivered for some reason.
           ERR_NOTOPLEVEL and ERR_WILDTOPLEVEL are errors that
           are returned when an invalid use of
           "PRIVMSG $<server>" or "PRIVMSG #<host>" is attempted.
       421    ERR_UNKNOWNCOMMAND
              "<command> :Unknown command"

         - Returned to a registered client to indicate that the
           command sent is unknown by the server.

       422    ERR_NOMOTD
              ":MOTD File is missing"

         - Server's MOTD file could not be opened by the server.

       423    ERR_NOADMININFO
              "<server> :No administrative info available"

         - Returned by a server in response to an ADMIN message
           when there is an error in finding the appropriate
           information.

       424    ERR_FILEERROR
              ":File error doing <file op> on <file>"

         - Generic error message used to report a failed file
           operation during the processing of a message.

       431    ERR_NONICKNAMEGIVEN
              ":No nickname given"

         - Returned when a nickname parameter expected for a
           command and isn't found.

       432    ERR_ERRONEUSNICKNAME
              "<nick> :Erroneous nickname"

         - Returned after receiving a NICK message which contains
           characters which do not fall in the defined set.  See
           section 2.3.1 for details on valid nicknames.

       433    ERR_NICKNAMEINUSE
              "<nick> :Nickname is already in use"

         - Returned when a NICK message is processed that results
           in an attempt to change to a currently existing
           nickname.
       436    ERR_NICKCOLLISION
              "<nick> :Nickname collision KILL from <user>@<host>"

         - Returned by a server to a client when it detects a
           nickname collision (registered of a NICK that
           already exists by another server).

       437    ERR_UNAVAILRESOURCE
              "<nick/channel> :Nick/channel is temporarily unavailable"

         - Returned by a server to a user trying to join a channel
           currently blocked by the channel delay mechanism.

         - Returned by a server to a user trying to change nickname
           when the desired nickname is blocked by the nick delay
           mechanism.

       441    ERR_USERNOTINCHANNEL
              "<nick> <channel> :They aren't on that channel"

         - Returned by the server to indicate that the target
           user of the command is not on the given channel.

       442    ERR_NOTONCHANNEL
              "<channel> :You're not on that channel"

         - Returned by the server whenever a client tries to
           perform a channel affecting command for which the
           client isn't a member.

       443    ERR_USERONCHANNEL
              "<user> <channel> :is already on channel"

         - Returned when a client tries to invite a user to a
           channel they are already on.

       444    ERR_NOLOGIN
              "<user> :User not logged in"

         - Returned by the summon after a SUMMON command for a
           user was unable to be performed since they were not
           logged in.
       445    ERR_SUMMONDISABLED
              ":SUMMON has been disabled"

         - Returned as a response to the SUMMON command.  MUST be
           returned by any server which doesn't implement it.

       446    ERR_USERSDISABLED
              ":USERS has been disabled"

         - Returned as a response to the USERS command.  MUST be
           returned by any server which does not implement it.

       451    ERR_NOTREGISTERED
              ":You have not registered"

         - Returned by the server to indicate that the client
           MUST be registered before the server will allow it
           to be parsed in detail.

       461    ERR_NEEDMOREPARAMS
              "<command> :Not enough parameters"

         - Returned by the server by numerous commands to
           indicate to the client that it didn't supply enough
           parameters.

       462    ERR_ALREADYREGISTRED
              ":Unauthorized command (already registered)"

         - Returned by the server to any link which tries to
           change part of the registered details (such as
           password or user details from second USER message).

       463    ERR_NOPERMFORHOST
              ":Your host isn't among the privileged"

         - Returned to a client which attempts to register with
           a server which does not been setup to allow
           connections from the host the attempted connection
           is tried.

       464    ERR_PASSWDMISMATCH
              ":Password incorrect"

         - Returned to indicate a failed attempt at registering
           a connection for which a password was required and
           was either not given or incorrect.
       465    ERR_YOUREBANNEDCREEP
              ":You are banned from this server"

         - Returned after an attempt to connect and register
           yourself with a server which has been setup to
           explicitly deny connections to you.

       466    ERR_YOUWILLBEBANNED

         - Sent by a server to a user to inform that access to the
           server will soon be denied.

       467    ERR_KEYSET
              "<channel> :Channel key already set"
       471    ERR_CHANNELISFULL
              "<channel> :Cannot join channel (+l)"
       472    ERR_UNKNOWNMODE
              "<char> :is unknown mode char to me for <channel>"
       473    ERR_INVITEONLYCHAN
              "<channel> :Cannot join channel (+i)"
       474    ERR_BANNEDFROMCHAN
              "<channel> :Cannot join channel (+b)"
       475    ERR_BADCHANNELKEY
              "<channel> :Cannot join channel (+k)"
       476    ERR_BADCHANMASK
              "<channel> :Bad Channel Mask"
       477    ERR_NOCHANMODES
              "<channel> :Channel doesn’t support modes"
       478    ERR_BANLISTFULL
              "<channel> <char> :Channel list is full"

       481    ERR_NOPRIVILEGES
              ":Permission Denied- You’re not an IRC operator"

         - Any command requiring operator privileges to operate
           MUST return this error to indicate the attempt was
           unsuccessful.

       482    ERR_CHANOPRIVSNEEDED
              "<channel> :You’re not channel operator"

         - Any command requiring ’chanop’ privileges (such as
           MODE messages) MUST return this error if the client
           making the attempt is not a chanop on the specified
           channel.
       483    ERR_CANTKILLSERVER
              ":You can't kill a server!"

         - Any attempts to use the KILL command on a server
           are to be refused and this error returned directly
           to the client.

       484    ERR_RESTRICTED
              ":Your connection is restricted!"

         - Sent by the server to a user upon connection to indicate
           the restricted nature of the connection (user mode "+r").

       485    ERR_UNIQOPPRIVSNEEDED
              ":You're not the original channel operator"

         - Any MODE requiring "channel creator" privileges MUST
           return this error if the client making the attempt is not
           a chanop on the specified channel.

       491    ERR_NOOPERHOST
              ":No O-lines for your host"

         - If a client sends an OPER message and the server has
           not been configured to allow connections from the
           client's host as an operator, this error MUST be
           returned.

       501    ERR_UMODEUNKNOWNFLAG
              ":Unknown MODE flag"

         - Returned by the server to indicate that a MODE
           message was sent with a nickname parameter and that
           the a mode flag sent was not recognized.

       502    ERR_USERSDONTMATCH
              ":Cannot change mode for other users"

         - Error sent to any user trying to view or change the
           user mode for a user other than themselves.
```

5.3 Reserved numerics

These numerics are not described above since they fall into one of the following categories:

1. no longer in use;

2. reserved for future planned use;

3. in current use but are part of a non-generic 'feature' of the current IRC server.

231	RPL_SERVICEINFO	232	RPL_ENDOFSERVICES
233	RPL_SERVICE		
300	RPL_NONE	316	RPL_WHOISCHANOP
361	RPL_KILLDONE	362	RPL_CLOSING
363	RPL_CLOSEEND	373	RPL_INFOSTART
384	RPL_MYPORTIS		
213	RPL_STATSCLINE	214	RPL_STATSNLINE
215	RPL_STATSILINE	216	RPL_STATSKLINE
217	RPL_STATSQLINE	218	RPL_STATSYLINE
240	RPL_STATSVLINE	241	RPL_STATSLLINE
244	RPL_STATSHLINE	244	RPL_STATSSLINE
246	RPL_STATSPING	247	RPL_STATSBLINE
250	RPL_STATSDLINE		
492	ERR_NOSERVICEHOST		

6. Current implementations

The IRC software, version 2.10 is the only complete implementation of the IRC protocol (client and server). Because of the small amount of changes in the client protocol since the publication of RFC 1459 [IRC], implementations that follow it are likely to be compliant with this protocol or to require a small amount of changes to reach compliance.

7. Current problems

There are a number of recognized problems with the IRC Client Protocol, and more generally with the IRC Server Protocol. In order to preserve backward compatibility with old clients, this protocol has almost not evolved since the publication of RFC 1459 [IRC].

7.1 Nicknames

The idea of the nickname on IRC is very convenient for users to use when talking to each other outside of a channel, but there is only a finite nickname space and being what they are, it's not uncommon for several people to want to use the same nick. If a nickname is chosen by two people using this protocol, either one will not succeed or both will removed by use of a server KILL (See Section 3.7.1).

7.2 Limitation of wildcards

There is no way to escape the escape character "\" (%x5C). While this isn’t usually a problem, it makes it impossible to form a mask with a backslash character ("\") preceding a wildcard.

7.3 Security considerations

Security issues related to this protocol are discussed in the "IRC Server Protocol" [IRC-SERVER] as they are mostly an issue for the server side of the connection.

8. Current support and availability

Mailing lists for IRC related discussion:
  General discussion: ircd-users@irc.org
  Protocol development: ircd-dev@irc.org

Software implementations:
  ftp://ftp.irc.org/irc/server
  ftp://ftp.funet.fi/pub/unix/irc
  ftp://ftp.irc.org/irc/clients

Newsgroup: alt.irc

9. Acknowledgements

Parts of this document were copied from the RFC 1459 [IRC] which first formally documented the IRC Protocol. It has also benefited from many rounds of review and comments. In particular, the following people have made significant contributions to this document:

Matthew Green, Michael Neumayer, Volker Paulsen, Kurt Roeckx, Vesa Ruokonen, Magnus Tjernstrom, Stefan Zehl.

10. References

[KEYWORDS]   Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.

[ABNF]   Crocker, D. and P. Overell, "Augmented BNF for Syntax Specifications: ABNF", RFC 2234, November 1997.

[HNAME]   Braden, R., "Requirements for Internet Hosts -- Application and Support", STD 3, RFC 1123, October 1989.

[IRC]   Oikarinen, J. & D. Reed, "Internet Relay Chat Protocol", RFC 1459, May 1993.

[IRC-ARCH]   Kalt, C., "Internet Relay Chat: Architecture", RFC 2810, April 2000.

[IRC-CHAN]   Kalt, C., "Internet Relay Chat: Channel Management", RFC 2811, April 2000.

[IRC-SERVER] Kalt, C., "Internet Relay Chat: Server Protocol", RFC 2813, April 2000.

11. Author’s Address

Christophe Kalt
99 Teaneck Rd, Apt #117
Ridgefield Park, NJ 07660
USA

EMail: kalt@stealth.net

12. Full Copyright Statement

Acknowledgement

Funding for the RFC Editor function is currently provided by the Internet Society.