1 IRC

1.1 IRC API

`make-irc` [*#:nick="bot"*] [*#:realname="bot"*] [*#:hostname="localhost"*] [*#:server="localhost"*] [*#:port=6667*] [*#:password="#f"*] [Procedure]

Create a new `irc-object`.

`*nick*` *"bot"* [Constant]
`*realname*` *"mr bot"* [Constant]
`*server*` *"localhost"* [Constant]
`*port*` *6667* [Constant]
`*hostname*` *"localhost"* [Constant]
`*quitmsg*` *"Not enough parenthesis"* [Constant]

Default values.

`irc-object?` *obj* [Procedure]

Return `#t` if *obj* is an `irc-object`, else return `#f`.

`nick` *obj* [Procedure]
`realname` *obj* [Procedure]
`hostname` *obj* [Procedure]
`password` *obj* [Procedure]
`port` *obj* [Procedure]
`server` *obj* [Procedure]

Access the corresponding field of `irc-object` *obj*.

`set-port!` *obj val* [Procedure]
`set-server!` *obj val* [Procedure]
`set-nick!` *obj val* [Procedure]
`set-realname!` *obj val* [Procedure]
`set-hostname!` *obj val* [Procedure]
`set-password!` *obj val* [Procedure]

Set the field of `irc-object` *obj*. *set-port!* expects a valid port number, all others expect a string. Using the empty string will reset the values to their defaults (see make-irc). The return value is not specified.

Note that it is not possible to set a value when connected. Trying to do so will result in an error.

`do-nick` *obj nick* [Procedure]

Not yet implemented.

`do-connect` *obj* [Procedure]

Connect `irc-object` *obj* to server, and try to register (PASS USER NICK sequence). Failure to connect results in an error.

Note that there is no nick collision detection yet, so make sure you use a 'free' nick.

`do-quit` *obj* [*#:quit-msg=*quit**] [Procedure]

Send the `QUIT` command, using *quit-msg* as quit message.

`do-close` *obj* [Procedure]

Close the connection without sending the `QUIT` command.

`do-command` *obj* [*#:command*] [*#:middle*] [*#:trailing*] [Procedure]

`do-privmsg` *obj receiver msg* [Procedure]

Send message *msg* to user or channel *receiver*.

`do-listen` *obj* [Procedure]

Returns a `irc-message-object` if there is data available, `#f` otherwise.

`do-wait` *obj* [Procedure]

Similar to *do-listen* but keeps waiting till data is available.

`do-join` *obj chan* [Procedure]

Send the `JOIN` command.

Currently there is no error checking implemented, so a rejected join still shows in the channel list.

`do-runloop` *obj* [Procedure]

```
(let ([sock (_socket obj)])
  (while (not (port-closed? sock))
    (handle-message obj (do-wait obj))))
```

`do-part` *obj chan* [Procedure]

Leave channel *chan*.

`add-message-hook` *obj proc* [*#:tag*] [*#:append=#f*] [Procedure]

`add-simple-message-hook!` *obj proc* [*#:sender*] [*#:receiver*] [*#:command*] [*#:middle*] [*#:trailing*] [*#:tag*] [*#:append*] [Procedure]

`exists-message-hook?` *obj tag* [Procedure]

Returns `#t` if a hook with tag *tag* exists, `#f` otherwise.

`remove-message-hook!` *obj tag* [Procedure]

Remove the procedure with tag *tag* from the message-hook of `irc-object` *obj*.

`run-message-hook` *obj* [*args*] [Procedure]

Apply all procedures from the message-hook of `irc-object` *obj* to the arguments arg .The order of the procedure application is first to last. The return value of this procedure is not specified.

`reset-message-hook!` *obj* [Procedure]

Remove all procedures from the message-hook of `irc-object` *obj*.

`channels->list` *obj* [Procedure]

Return the channels joined by `irc-object` *obj* as list.

`in-channel?` *obj chan* [Procedure]

Returns `#t` if channel *chan* is joined, `#f` otherwise.

2 Message handling

Messages are parsed according to the 'pseudo' BNF in RFC 1459 (http://www.ietf.org/rfc/rfc1459.txt).

```
<message> ::=
    [':' <prefix> <SPACE> ] <command> <params> <crlf>
<prefix> ::=
    <servername> | <nick> [ '!' <user> ] [ '@' <host> ]
<command> ::=
    <letter> { <letter> } | <number> <number> <number>
<SPACE> ::=
    ' ' { ' ' }
<params> ::=
    <SPACE> [ ':' <trailing> | <middle> <params> ]
<middle> ::=
    <Any *non-empty* sequence of octets not including SPACE or NUL or CR or LF, the first o
<trailing> ::=
    <Any, possibly *empty*, sequence of octets not including NUL or CR or LF>
<crlf> ::=
    CR LF
```

2.1 Message API

`(irc` *message*`)` [Module]

`parse-message-string` *msg* [Procedure]

Parse the message string *msg* into an `message-object`.

`make-message` [*#:command*] [*#:middle*] [*#:trailing*] [Procedure]

Create a new `message-object`.

`message?` *obj* [Procedure]

Return `#t` if *obj* is an message-object, else `#f`.

`prefix` *msg* [Procedure]

Return the prefix part of `message-object` *msg*.

Either '(user nick host), server or #f.

`prefix-type` *msg* [Procedure]

Returns 'USER if the message was send by a user, 'SERVER if the message was send by a server and #f otherwise.

`command` *msg* [Procedure]

Returns either a number or symbol depending on the type of command.

`middle` *msg* [Procedure]

Returns the empty string, a string or a list of string.

`trailing` *msg* [Procedure]

Returns a string or #f.

`time` *msg* [Procedure]

Returns a timestamp as created by (current-time).

2.1.1 Message handling helpers

`parse-source` *msg* [Procedure]

Find the source or `message-object` *msg*. If the source is found the return value is a string, otherwise it is `#f`.

`parse-target` *msg* [Procedure]

Find out who to send a reply to.

Note that this only works for PRIVMSG and PING commands.

`is-channel?` *str* [Procedure]

Test if `string` *str* is a valid channel.

`message->string` *msg* [Procedure]

Transform `message-object` *msg* into a sendable string (i.e. command middle :trailing).