Supybook

REVISION HISTORY			
NUMBER	DATE	DESCRIPTION	NAME
0.0.4	2011-09-14		HH

Contents

0.0.4

Preface

Note
At the time of this writing, Supybot version is 0.83.3. If you are using a newer (or even older) version, keep that in mind.

Note
This document is very much a work in process. It covers nowhere near everything there is to Supybot. However, it hopefully allows a good start into learning Supybot.

What is this document?

This document is a handbook for Supybot, the IRC (Internet Relay Chat) bot written in Python.

Motivation behind this document

Some time ago I started needing an IRC bot for various purposes. The bot would have to be able to take care of auto-opping and similar "traditional" channel duties. This made me think of Oer, an old but very nice bot. However, I also wanted to have factoid functionality similar to blootbot/infobot, without having to run multiple bots. That was when I arrived at Supybot once again. This time I decided to give it a closer look.

My experience of the documentation was lacking though. I longed for something similar to Oer's User Manual that provides a quick references for basic administrator tasks and so forth. Alas, I could not find such document.

I figured I could as well wrap up such a handbook while learning to use the bot myself. Hopefully someone finds reading this document as useful as writing it was.

Conventions used in this document

`command <name> [value]`
A command that takes `name` as a required parameter and `value` as an optional parameter.

`command <nick...>`
A command that takes one or more parameters.

`command [--{foo,bar}] <value>`
A command that takes two optional options as parameters, eg. `--foo value` and/or `--bar value`.

Plugin
A plugin name

This document is Free

How to give feedback

If you spot an error, have suggestions or just want to tell me how much you love/hate the document, send e-mail to hoxu@users.sf.net. Prefix the Subject with `[supybook]`.

Chapter 1

Getting started

1.1 Initial configuration

Create a new directory (eg. `mkdir ~/supybot`) and run `supybot-wizard` in it. Follow the directions to get the bot initially configured.

Once you are ready with the wizard, I suggest you start the bot inside screen:

```
$ screen
$ supybot configFile
```

This way you can attach to the screen later to see the messages it writes to stdout, while still running the bot in the background. Alternative is to run it in the daemon mode, `supybot --daemon configFile`.

You can set up a crontab to start the bot automatically, for example:

```
$ crontab -e
> @reboot screen -d -m supybot /path/to/configFile
```

1.2 Identifying to the bot

After running supybot-wizard and starting the bot, connect to the same IRC network it is on, and /query it. You can identify with the `identify <name> <password>` command. You can check the bot's idea of who you are with the `whoami` command.

Note
All commands that contain a `password` must be sent to the bot in private.

1.3 Accessing the online help

Most commands on the bot have a short online help available. You can use `list` to list loaded plugins, and `list <plugin>` to list commands in those plugins. `help [plugin] <command>` can be used to access the command help. Use `more` to read long messages from the bot.

Help for configuration items can be accessed with the `config help <key>` command.

Chapter 2

Understanding supybot's peculiarities

Supybot has a couple of features that sets it apart from more traditional bots. More specifically, nested commands, plugin framework (even the basic functionality is implemented using plugins), and capabilities. This section covers some basic information about Supybot. Feel free to skip it, but come back later if you run into something you don't understand.

2.1 Nested commands

Supybot allows nested commands, in other words, you can pass the result of a command to another command as a parameter. What does this allow then, though, apart from the obvious bragging rights? For example, if you want to restore a configuration entry to the default value, you can type:

```
config reply.withNickPrefix [config default reply.withNickPrefix]
```

Obviously, that wasn't very neat. They could've included a `config reset <key>` command instead, for example. But I'm sure you can figure out **something** more useful to do with this, as a homework ;-)

2.2 Plugins

Pretty much everything in Supybot is a plugin. Most commands you use belong to a plugin. If two plugins provide the same command, then you need to prefix the command with the plugin name. For example, the `ignore list` command yields the following output:

output

```
Error: The command "ignore list" is available in the Admin and Channel plugins.
Please specify the plugin whose command you wish to call by using its name as a command ↩
    before "ignore list".
```

Thus, you need to either use `admin ignore list` or `channel ignore list` command. That is, unless you define which plugin is the default one using the `defaultplugin [--remove] <command> [plugin]` command provided in the Owner plugin.

The `list` command (Plugin) can be used to list loaded plugins. You can prevent some plugins from showing in the list with the command:

```
config plugins.<plugin>.public False
```

> **Warning**
> This does not really make the plugins private. They can still be listed by unregistered users (unless they have anticapability for `list`). This is a bug in my opinion.

list
: List currently loaded plugins, excluding the ones set non-public.

list --private
: List loaded plugins that are set to non-public.

list <plugin>
: List commands provided by the given plugin.

load [--deprecated] <plugin>
: Load a plugin. Supybot looks for plugins in directories listed in `config conf.supybot.directories.plugins`.

unload <plugin>
: Unloads a plugin. The Owner plugin cannot be unloaded.

Note
Plugins can store information in a database. By default, the database is channel-specific. This can be changed by modifying `config databases.plugins.channelSpecific` to be global. The config item can also be set on per-channel basis.

2.3 Configuration

Most of Supybot configuration is done online using the Config plugin. This excludes things such as granting the owner capability (must be done by modifying the files).

There are two types of configuration items: global and channel-specific. The latter are actually no different from the global ones, except that they can be set for channels as well. This means you can have a global default which is overridden for specific channels.

Configuration items are hierarchical. The "root" item is `supybot`, which can be omitted. Configs for plugins live under their own key, `supybot.plugins`, or shortly, `plugins`.

config <name> [value]
: Get the current value of `name`, or set it to `value` if provided.

config channel [channel] <name> [value]
: Ditto, but for channel configs.

config default <name>
: Get the default value of `name`. This does not change the value. No, there is no command to reset an item to default value. You need to use `config <name> [config default <name>]` for that.

config export <filename>
: Export non-confidential parts of configuration to a file for debugging purposes.

config help <name>
: Show help for a configuration item.

config list <group>
: List configuration items in `group`. Subgroups are prefixed with @, channel-specific items with #.

config reload
: Reloads configuration. Mostly useful if you’ve had to modify the files by hand.

config search <word>
: Show configuration items matching `word`.

owner flush
: Save configuration changes to disk. This should be done automatically as well, if the `flush` configuration item is True (the default).

Note
Some of the sections in this document have a list of related configuration items. They are listed similarly to the table below. In this example the config value could be modified with the `config plugin.example.foo <True|False>` command. If the key is prefixed with `#`, the config value can be set both globally and on channel-specific basis.

Table 2.1: plugin.example

Key	Default	Description
foo	True	Example key.
#bar	False	Example key that can be set on channels as well.

2.4 Capabilities

Many traditional IRC bots manage permissions of users using flags. Some network services in fact, do, too. Each user can have various flags on each channel. The flags can mean auto-op, op, use of !ban command and so forth. On side of those, there are often global user flags that entitle the user to full access, global auto-op, etc. Some bots also support channel flags that determine what bot functionality is available on the channel.

Supybot does not have any flags. Instead, the permissions are managed using *capabilities*. There are two kinds of capabilities: `user capabilities` and `channel capabilities`.

User capabilities are checked first when a user tries to run a command. If the user has an *anticapability* for the command (eg. `-command`, `-Plugin.command`) or the Plugin (`-Plugin`), it won't be run.

Next, if the command was run on the channel, the *channel capabilities* are checked. The logic is same as above, but the checked capabilities are prefixed with `#channel,`, for example `#channel,-Plugin.command`.

There are some special capabilities recognized by Supybot:

owner
: For bot owners: the people who have "physical" access to the bot and its files. This cannot be granted online; the *conf/users.conf* file must be edited by hand followed by `reload`. Owners are exempt from `#channel,op` capability checks and channel anticapabilities.

admin
: For bot administrators. Users with this capability can manage global bot properties, make the bot join new channels and so forth. However, they can't do channel administration which is reserved for ops.

#channel,op
: Channel ops can execute channel-related commands.

trusted
: Allow user to run commands that can potentially crash the bot, or cause denial of service on the system it's running on.

Commands used to manipulate capabilities are covered in the Manipulating channel capabilities and Manipulating user capabilities sections.

Chapter 3

Administrative tasks

3.1 Networks

Related plugins: Network, Services.

3.1.1 Adding a network

network connect [--ssl] <network> [<host[:port]>] [password]
Connect to `network`. `host` must be provided if the network is new or has no servers defined.

3.1.2 Reconnecting

reconnect [network] [message]
Disconnects and connects `network`, or current if not specified. `message`, if given, is shown as the quit message, otherwise `config plugins.Owner.quitMsg` is used, or your nickname.

3.1.3 Disconnecting

network disconnect [network] [message]
Disconnect `network`, or current if not specified. `message` as above.

3.1.4 Listing networks

networks
List of networks & servers currently connected to.

output

```
freenode: wolfe.freenode.net and ircnet: irc.elisa.fi
```

config list networks
List all networks.

3.1.5 Adding more servers

Once you've added a network with the initial server, you can add more servers:

`config networks.<network>.servers [config networks.<network>.servers] server:6667`

3.1.6 Listing network servers

`config networks.<network>.servers`

output

```
chat.freenode.net:6667
```

3.1.7 Services: NickServ

> **Warning**
> My pratical experience with Services plugin on Freenode is... not-so-good. It definitely does not work all the time as expected.

You can make Supybot identify itself to the network NickServ after it has connected.

`config plugins.Services.noJoinsUntilIdentified True`
: Settings this is useful on Freenode and other networks who change the user mask after identifying to NickServ. In my experience this seems to be a tad buggy, so I don't recommend enabling it unless really needed.

`config plugins.Services.NickServ NickServ`
: Tell the bot what name NickServ can be found under.

`services password <nick> [password]`
: Can be used to set or remove NickServ password.

```
NOTE: Password removal did not work for me on Supybot 0.83.3
```

`services identify`
: Identifies the bot to NickServ with the current nick. You don't need to give this command after the bot has been set up; it will identify when connecting to the network automatically.

3.1.8 Services: ChanServ

You can make Supybot request op after joining a channel on a network with ChanServ.

`config plugins.Services.ChanServ ChanServ`
: Tell the bot what name ChanServ can be found under.

`config plugins.Services.ChanServ.op <on|off>`
: Set the default for all channels. This will be used unless a channel-specific config overrides it.

`config channel [channel] plugins.services.ChanServ.op <on|off>`
: Set to request op on the given `channel`.

Voice and half-op (on networks supporting it) can be used similarly.

3.2 Channels

Related plugins: Channel.

Note
Commands in this section (such as adding/removing channels) work in the current network, eg. the one you are messaging the bot in. The commands also accept a `[channel]` parameter which is needed only when the command is written in private.

Note
If you want to /msg the bot in one network, while having the command apply in another network, you can use the `network command <network> <command> [params]` command. The reply comes in the other network if you are there as well.

Warning
The `network command` command does not seem to check if the user having your nick in the other network is recognized (eg. identified or recognized by hostmasks), so the reply may end up to someone else.

3.2.1 Adding a new channel

`join <channel> [key]`
Joins the `channel` using `key` if provided. Channels are automatically remembered and joined when the bot connects to the network next time.

Note
`config plugins.Channel.alwaysRejoin` determines whether the bot will (attempt to) rejoin the channel when kicked out (the default).

3.2.2 Listing channels

`channels`

Note
This only works in private, to prevent knowledge of top secret channels from falling into wrong hands.

You can list channels in another network with `config networks.<network>.channels`. Or with `network command <network> channels`.

3.2.3 Removing a channel

`part [channel] [reason]`
Makes the bot leave `channel`, showing `reason` as part message if given. Note that all channel data is retained, but the bot does not join the channel anymore the next time it connects to the network.

3.2.4 Modifying channel config

`config channel [channel] <name> <value>`
Sets config item `name` on `channel` to `value`, overriding the global value.

3.2.5 Setting the key

`channel key [channel] [key]`
Sets key on `channel` to `key`, or removes it if `key` is not given.

3.2.6 Setting the limit

`channel limit [channel] [limit]`
Sets limit on `channel` to `limit`, or removes it if `limit` is not given (or is zero).

3.2.7 Channel commands

Channel ops can use the following commands to control the channel via the bot, assuming it is opped.

`op [channel] [nick...]`
Ops the given nicks (or you if none) on the channel.

`deop [channel] [nick...]`
Ditto, but deops.

`voice [channel] [nick...]`
Voices the given nicks (or you if none) on the channel.

`devoice [channel] [nick...]`
Ditto, but devoices.

`kban [channel] [--{exact,nick,user,host}] <nick> [seconds] [reason]`
Bans and kicks the given nick from the channel. If `seconds` is specified and is not 0, the ban will expire after that time.

`mode [channel] <mode> [params]`
Set channel mode. This can be used to change any channel modes, making the commands below redundant aliases.

`moderate [channel]`
Set +m. This is not enforced by the bot, so any channel op can remove it.

`unmoderate [channel]`
Set -m.

`topic lock [channel]`
Set +t. Not enforced, so any channel op can remove it.

`topic unlock [channel]`
Set -t.

`alert [channel] <text>`
Sends `text` to all users on the channel with op capability.

`cycle [channel]`
Make the bot part and join the channel. Mostly useful to test whether auto-ops from other bots/users work for the bot.

3.2.8 Maintaining the ban list

ban add [channel] <nick|hostmask> [expires]
Add ban for given nick or hostmask on the channel. If nick is given, the full hostmask is banned. `expires` when gives, expires the ban after so many seconds.

ban list [channel]
List bans with their expire times.

ban remove [channel] <hostmask>
Removes the ban on given `hostmask`.

config plugins.Channel.banmask <string>
Hostmask style for `ban add`: `exact`, `nick`, `user`, `host`. This means which parts of the nick's current hostmask are used (rest are wildcarded). Default is `host user` (eg. `*!user@host`).

Note
Users matching the ban list are not automatically kicked off the channel. See the `kban` command in the previous section to kick and ban a user.

3.2.9 Maintaining the ignore list

channel ignore add [channel] <nick|hostmask> [expires]
Ignores `hostmask` (or the full hostmask `nick` currently has) on `channel`. If `expires` is given, the ignore expires after that many seconds.

channel ignore list [channel]
Show ignored hostmasks on `channel`. This does not show the expire times, though (argh!).

channel ignore remove [channel] <hostmask>
Remove `hostmask` from list of ignored hostmasks on `channel`.

Note
There is also a global ignore list, available for admins.

3.2.10 Listing channel nicks

`channel nicks [channel]`

3.2.11 Topic operations

Related plugins: Topic.

Supybot allows elaborate manipulation of the channel topic. Topic consists of items, separated by a configurable character (default is `||`). Items are one-indexed.

People who have used the Oer bot may notice that the topic manipulation commands are somewhat similar.

topic add [channel] <text>
Add a new topic item at the end.

topic change [channel] <number> <regexp>
Do a regular expression substitution on the topic item `number`. For example: `topic change 2 s/foo/bar/`.

topic default [channel]
Restore topic on channel to the default set with `config plugins.Topic.default`.

topic fit [channel] <text>
Adds a new topic item at the end like `topic add`, but if there isn't enough space, first removes some items from the beginning. `topic add` complains if there isn't enough space.

topic get [channel] <number>
Return topic item `number`.

topic insert [channel] <text>
Add a new topic item in the beginning.

topic list [channel]
Return list of topic items.

topic lock [channel]
Set channel mode `+t`, preventing non-opped users from changing the topic.

topic redo [channel]
Undo the last undo.

topic remove [channel] <number>
Remove item `number` from the topic.

topic reorder [channel] <number>...
Reorder topic items in the given order. You must give as many numbers as there are items. For example, to move third item to first position, use `topic reorder 3 1 2`.

topic replace [channel] <number> <text>
Replace topic item `number` with `text`.

topic restore [channel]
Revert any changes made to the topic by users, and set it back to whatever the bot last set it to.

topic separator [channel] <separator>
Change the topic separator to `separator`.

topic set [channel] [number] <text>
Replace either the whole topic or, if `number` is given, one item. In the latter case, this is the same as `topic replace`.

topic shuffle [channel]
Reorder the topic items randomly.

topic swap [channel] <number1> <number2>
Swap topic items at the given positions.

topic [channel]
Show (the whole) topic for the channel.

topic undo [channel]
Revert the last change a topic command made to the topic. Note that if users directly edited the topic, those changes will be lost. Can be used multiple times.

topic unlock [channel]
Set channel mode `-t`, allowing all users to change the topic.

Table 3.1: plugins.Topic

Key	Default	Description
#default		Default channel topic
#format	$topic ($nick)	Format for the items

Table 3.1: (continued)

Key	Default	Description
#recognizeTopiclen	True	Whether to recognize max topic length given by the server and refuse to set longer topics.
#separator	||	Used to concatenate items
#undo.max	10	How long undo history to keep

Aliases for partial oer compatibility:

```
alias add ta topic add $1
alias add td topic delete $1
alias add te topic replace $1 $2
alias add ts topic swap $1 $2
```

3.2.12 Logging

Logging of channels is provided by the ChannelLogger plugin. Various channel-specific configuration items are provided, see `config list plugins.ChannelLogger`.

By default logs will go into logs/ChannelLogger/<network>/<channel>/<channel>.log and will be rotated when the default "%d-%a-%Y" (eg. 06-Sat-2008) timestamp rotates. See the python documentation for strftime for the formatting characters.

> **Note**
> The logs of ChannelLogger cannot be searched online. However, other plugins provide some searching functionality. See Searching the history.

3.2.13 Auto-ops and voices

Auto-opping is provided by the AutoMode plugin.

`load AutoMode`
Loads the AutoMode plugin, which works out-of-box. By default it's enabled on all channels and voices/halfops/ops users with respective capabilities. You can enable auto-opping only on some channels by tweaking the config keys below.

Table 3.2: plugins.AutoMode

Key	Default	Description
#enable	True	Whether the plugin is enabled.
#fallthrough	False	If enabled and `op` is False, halfops/voices instead if they are True.
#halfop	True	Halfop users with the `halfop` capability.
#op	True	Ditto, but op.
#voice	True	Ditto, but voice.
#ban	True	Whether to ban people who join the channel and are on the banlist.
#ban.period	86400	How many seconds bans will last.

3.2.14 Manipulating channel capabilities

capability list [channel]
List capabilities on `channel`.

capability set [channel] <capability..>
Adds the given `capability` to `channel`.

capability unset [channel] <capability..>
Removes the given `capability` from `channel`.

capability setdefault [channel] <True|False>
Whether to allow users on `channel` by default to access non-maintenance related commands. Default is True.
Note that this concerns unregistered users as well. So if you want to disallow use of commands by unregistered users, set default user capabilities to allow them, and set this to False.

See also Capabilities.

3.3 Users

Related plugins: Users.

Supybot users are global: they are visible across networks. This means the same username/password and hostmasks will work in all networks the bot is on.

Users are recognized either by matching hostmasks, or after manually identifying to the bot. In secure mode, the user must both match a hostmask and identify to the bot (`uset set secure [password] <True|False>`).

Note
If users knowing each others' nicks is an issue, they could come up with different aliases when registering to the bot. The bot username does not have to match the nick of the user.

3.3.1 Adding a new user

`user register <name> <password>`

3.3.2 Manipulating hostmasks

`user hostmask add` - add your current hostmask. Obviously this makes only sense after identify.

`user hostmask add [name] [hostmask] [password]` - add hostmask for another user. If not owner, password must be given.

`user hostmask remove <name> <hostmask> [password]`

Note
There is no way to add network-specific hostmasks.

3.3.3 Listing users

`user list [glob]` - list registered users. Note that the list of users is global across networks.

3.3.4 Deleting users

`user unregister <name> [password]`

3.3.5 Changing password

`user set password <user> <old password> <new password>`.

3.3.6 Renaming a user

`user changename <name> <new name> [password]`

Users can change their name themselves.

3.3.7 Manipulating user capabilities

capabilities [user]
List capabilities of the `user`, or the calling user.

admin capability add <user|hostmask> <capability>
Add `capability` to `user` or a user that matches the `hostmask`.

admin capability remove <user|hostmask> <capability>
Ditto, but remove the capability.

channel capability add [channel] <nick|user> <capability..>
Add capability `capability` on `channel` to `nick`/`user`.

channel capability remove [channel] <nick|user> <capability..>
Remove capability `capability` on `channel` from `nick`/`user`.

defaultcapability <add|remove> <capability>
Add or remove `capability` from list of capabilities given to new users.

config capabilities
List default capabilities given to new users.

output

```
-owner -admin -trusted
```

See also Capabilities.

3.4 General bot maintenance

Related plugins: Admin, Config.

3.4.1 Setting nickname & alternative nick

admin nick <newnick>
Change nick to `newnick`.

config nick
Default nick.

config nick.alternates
Space-separated list of alternate nicks, %s refers to nick.

Note
It is not possible to have a different nick in different networks.

3.4.2 Setting ident

config ident <newident>
: Sets the bot's ident (nick!ident@host).

3.4.3 Setting ircname

config user [ircname]
: Sets the bot's ircname/realname to ircname. If left empty, defaults to *Supybot 0.83.3* for example.

3.4.4 Setting command prefix / controlling when the bot replies

Like most other bots, the bot can be addressed by its nickname, or a command prefix character (any or many of ~!@#$%^&*()_-+=[{}
For special (braindead?) purposes the bot can also be made assume that all lines are addressed to it.

config reply.whenAddressedBy.chars
: List of characters the bot will recognize as addressing, besides the nick of the bot.

config channel [channel] reply.whenAddressedBy.chars
: Ditto, but for a specific channel.

config reply.whenNotAddressed
: Assume everyone wants to talk to the bot, eg. treat all messages as if addressed to the bot. This does not imply reply.WhenNotCommand False which you should set as well.

config reply.whenNotCommand
: Whether to reply when addressed with an invalid command.

See also: Configuration: reply.

3.4.5 Keeping the primary nick

Like other bots, Supybot can be configured to try and keep the primary nick using the NickCapture plugin. This is primarily useful in networks with no NickServ support.

load NickCapture
: Loads the plugin which works without further configuration.

config plugins.NickCapture.ison
: Whether the plugin is actively checking for the primary nick. This setting makes no sense, as you might as well unload NickCapture if you think about setting this to false.

config plugins.NickCapture.ison.period
: How many seconds to wait between nick availibity polls. The default is 600 (10 min). The smaller you set this, the higher the chance of the bot recovering the nick when it becomes available. On the other hand, you will also generate more traffic so you might want to avoid that.

3.5 Owner commands

owner announce <text> - send text to all channels the bot is on.

owner ircquote <raw> - send raw as-is to the server. You need to know your way around RFC1459 pretty well to use this.

Chapter 4

User commands

4.1 Searching the history

url last [channel] [--{from,with,without,near,proto} value] [--nolimit]
Find last URL (or all with `--nolimit`) matching given criteria. From matches nick, with(out) part of the URL, near rest of the line where the URL was, and proto matches the protocol (https, ftp, etc). In case of multiple URLs, the newest is listed first. Multiple criterias can be given.

```
NOTE: This command lists only the URLs, not nick or what message the URLs were part of
```

last [--from,in,on,with,without,regexp} value] [--nolimit]
Find messages matching given criteria. From matches nick, in matches channel, on matches network, with(out) matches part of the message, regexp matches messages that are included by the regular expression. Also see `config protocols.irc.`

4.2 Useful plugins

4.2.1 Alias

Alias is one of the most powerful plugins available in Supybot. It allows you to set up *shortcuts* to other commands. When put together with command nesting, and the ability to provide arbitrary number of required and optional arguments to each alias, this becomes a very versatile feature. The main commands in this plugin are `add` and `remove` to add and remove aliases:

add <name> <alias>
Defines an alias `name` that executes `alias`. The `alias` should be in the standard "command argument [nestedcommand argument]" arguments to the alias; they'll be filled with the first, second, etc. arguments. $1, $2, etc. can be used for required arguments. @1, @2, etc. can be used for optional arguments. $* simply means "all remaining arguments," and cannot be combined with optional arguments.

remove <name>
Removes the alias named `name`.

As a simple example, let's say you want to set up a quick check to see if some website is down, using the downforeveryoneorjustme.com service. See the following example session for how you'd accomlish that:

```
<user> alias add isitdown web title http://downforeveryoneorjustme.com/$1
<supybot> The operation succeeded.
<user> isitdown google.com
<supybot> It's just you.
```

What this has accomplished is that you don't have to give the full command, which would be `web title http://downforeveryo`
instead you now have a quick shortcut to check the status of a site. Aliases get their own help messages, just like all the normal commands, which tell us what got stored in the alias. For example, for the alias above:

```
<user> help isitdown
<supybot> (isitdown <an alias, 1 argument>) -- Alias for "web title http:// ↩
    downforeveryoneorjustme.com/$1".
```

We have introduced the variable substitution feature of aliases here. Note the `$1` in the alias, which means "put the first argument provided to this command right here". You can also have optional arguments, which are represented as `@1`, `@2`, etc. There is also `$*` which acts as a placeholder for "all the remaining arguments", but it cannot be mixed with optional arguments. For example, we can try the following:

```
<user> alias add saymoo echo moo $1 @1
<supybot> The operation succeeded.
<user> help saymoo
<supybot> (saymoo <an alias, at least 1 argument>) -- Alias for "echo moo $1 @1".
<user> saymoo bla
<supybot> moo bla
<user> saymoo bla stuff
<supybot> moo bla stuff
```

There are some important details to understand when you use nested commands and quotes in your aliases. For example, you might naively try the following:

```
<user> alias add test echo [uptime]
<supybot> The operation succeeded.
<user> help test
<supybot> (test <an alias, 0 arguments>) -- Alias for "echo I have been running for 1 day,  ↩
    14 hours, 46 minutes, and 53 seconds.".
```

Notice what happened - the uptime command was executed, then stored into the alias, rather than the actual command nesting getting stored. Every time you call the test command, the output will always be exactly the same. This is probably not what you had in mind. To avoid immediate nested command evaluation, you must put quotes around the argument to alias (or around the argument to echo). Like so:

```
<user> alias add test "echo [uptime]"
<supybot> The operation succeeded.
<user> help test
<supybot> (test <an alias, 0 arguments>) -- Alias for "echo [uptime]".
<user> test
<supybot> I have been running for 1 day, 14 hours, 50 minutes, and 19 seconds.
```

There, much better!

Another issue we'll want to address is, what if you want literal quotes in your alias output? For example, say you want an alias that would give the following output:

```
<user> test
<supybot> bla "moo" bla
```

We are going to use the `echo` command from the *Utilities* plugin, which lets the user make the bot say any string. Our first naive try may be the following:

```
<user> alias add test echo bla "moo" bla
<supybot> The operation succeeded.
<user> help test
<supybot> (test <an alias, 0 arguments>) -- Alias for "echo bla moo bla".
<user> test
<supybot> bla moo bla
```

Doesn't work. The quotes are simply taken as quoting an argument, and don't make it into the alias. Let's try escaping the quotes and see what happens.

```
<user> alias add test echo bla \"moo\" bla
<supybot> The operation succeeded.
<user> help test
<supybot> (test <an alias, 0 arguments>) -- Alias for "echo bla \"moo\" bla".
<user> test
<supybot> bla \"moo\" bla
```

Closer, but still no cigar - this is because of the way supybot argument parsing works - unquoted arguments get taken as string literals, while quoted arguments get string parsing done on them. Compare the two outputs below:

```
<user> echo bla\"bla
<supybot> bla\"bla
<user> echo "bla\"bla"
<supybot> bla"bla
```

Raw unquoted string is taken as a literal, while the quoted string interprets the `\"` sequence as an escaped quote character. This is what we want. Our goal is, then, to get the string `echo "bla \"moo\" bla"` as the content of our alias.

Our final, successful, try follows:

```
<user> alias add test "echo \"bla \\\"moo\\\" bla\""
<supybot> The operation succeeded.
<user> test
<supybot> bla "moo" bla
<user> help test
<supybot> (test <an alias, 0 arguments>) -- Alias for "echo "bla \"moo\" bla"".
```

Note what is happening here. First, we quote the entire content of the alias as an argument, to get string parsing going. We escape the quote character before bla, to get a literal quote character into the alias string. We then triple-escape the second quote character. `\\` gets a literal backslash into our alias string, while the following `\"` gets a literal quote. We do the same thing for the closing quotation around `moo`. Finally, we stick in another literal quote to finish quoting the argument to echo, and at the end, close the exterior quotation pair.

Quite a bit of detail here - but necessary to know if you're going to be using aliases to perform complex tasks.

4.2.2 Anonymous

Allows you to provide users a way to chat on a channel anonymously (eg. only the bot owner(s) know via logs who are talking).

do <channel> <action>
Sends `action` to `channel`. This is the same as a normal IRC client `/me does something` command.

say <channel> <text>
Say `text` on `channel`.

Table 4.1: plugins.Anonymous

Key	Default	Description
#requirePresenceInChannel	True	Whether the user must be in the channel the message is targeted to.
allowPrivateTarget	False	Whether to allow a nick as a target for say. NOTE: This has not been implemented in 0.83.3 although it exists!
requireCapability		If set, capability to check for.
requireRegistration	True	Whether registration is required to use this plugin.

4.2.3 Dict

Dict provides dictionary functionality using dict.org. You can also use a local dictd server.

dict [dictionary] <word>
Show dictionary entry for `word`, from `dictionary` if provided. If `plugins.Dict.default` is set, use the specified dictionary instead of all.

dictionaries
List dictionaries available on the used server.

dict random
Show a random dictionary from available dictionaries.

config plugins.Dict.server [server]
The dictd server to be used, default is dict.org.

config plugins.Dict.default [dictionary]
Channel-specific default dictionary for `dict` command. `*` means to use all dictionaries. `wn` is a good default if english words are mostly looked up.

4.2.4 Later

This is a nick-based replacement for NoteServ and the likes. Simply put, you give the bot a note to deliver to a nick (or wildcard) the next time it sees a matching nick. In other words, this can be used to deliver messages to people who are not registered to the bot. Naturally that is not a very safe method of communicating.

notes [nick]
List nicks that have notes queued, or the notes queued for `nick` if given.

later tell <nick> <text>
Queues `text` to be sent to first matching `nick` when seen. Nick can contain wildcards, eg. `foo*`.

config plugins.Later.maximum
How many messages can be queued per nick at maximum, default is 0 = no limit.

config plugins.Later.private
Whether to send notes in private or on the channel where the recipient is seen.

4.2.5 MoobotFactoids

MoobotFactoids implements nifty factoids.

```
<user> !supybook is <reply> Please read http://supybook.fealdia.org/ before asking.
<bot> OK
<user> !supybook
<bot> Please read http://supybook.fealdia.org/ before asking.
```

Setting up

```
config databases [config databases] sqlite
load moobotfactoids
```

Note
If you get any of the following errors, you need to install sqlite, python-sqlite and python-pysqlite.

```
NoSuitableDatabase: No suitable databases were found.  Suitable databases include sqlite.
Error: You need to have PySQLite installed to use this plugin.  Download it at <http:// ↩
    pysqlite.org/>
```

4.2.6 Seen

The Seen plugin keeps track of last channel/nick/user activity. Most typical use is asking the bot when a given user was last seen chatting on a channel.

`seen any [channel] [--user <user>] [nick]`
: Lists any activity given `nick` or `user` was doing on the channel. If no `nick` or `user` is given, returns the last activity on the channel, regardless of who it was from.

`seen last [channel]`
: Last line said on the channel.

`seen [channel] <nick>`
: Last time a `nick` was seen on a channel and what it said.

`seen user [channel] <user>`: Ditto, except use a user name, disregarding what nick the said user had.

4.2.7 Web

The Web plugin contains some useful WWW-related functionality, such as fetching titles for URLs users paste on the channel.

`doctype <url>`
: Show the doctype line of `url`, if any.

`fetch <url>`
: Show the contents of `url`. Amount of data shown is determined by the configuration variable `plugins.Web.fetch.maximu`

`headers <url>`
: Show web server headers for the `url`.

`netcraft <hostname|ip>`
: Ask netcraft what OS and web server it thinks the server is running.

`size <url>`
: Show size of `url`, based on the `Content-Length` header sent by the web server.

`title <url>`
: Show title for `url`. This can be done automatically for all URLs pasted on a channel; see the configuration variable `plugins.Web.titleSnarfer`.

`urlquote <text>`
: Return `text` quoted into a URL. Eg. `urlquote foo bar` → `foo%20bar`.

`urlunquote <text>`
: Likewise, but reverse.

Table 4.2: plugins.Web

Key	Default	Description
#nonSnarfingRegexp		if set, URLs matching the pattern are not snarfed.
#titleSnarfer	false	Whether to fetch and show the title for pasted URLs.
fetch.maximum	0	Maximum bytes to fetch with the `fetch` command. If zero, `fetch` is disabled.

4.3 Entertainment

4.3.1 ChannelStats

ChannelStats provides channel/registered user statistics (statistics as in *large numbers*).

`channelstats [channel]`: Show statistics for `channel`: messages, characters, words, smileys, frowns, actions, joins, parts, quits, kicks, mode changes, and topic changes.

`stats [channel] [user]`: Show statistics for `user` on `channel`: messages, characters, words, smileys, frowns, actions, joins, parts, quits, kicks given/received, topic changes, and mode changes.

Warning
This plugin may turn otherwise normal users into spammers. But it can also provide an incentive to register on the bot. :-)

Table 4.3: plugins.ChannelStats

<table>
<tr><th>Key</th><th>Default</th><th>Description</th></tr>
<tr><td>#frowns</td><td><code>:| :-/ :-\ :\ :/ :(:-(:'(</code></td><td>Space-separated list of frowns.</td></tr>
<tr><td>#selfStats</td><td><code>True</code></td><td>Whether to include the bot in the statistics.</td></tr>
<tr><td>#smileys</td><td><code>:) ;) ;] :-) :-D :D :P :p (= =)</code></td><td>Space-separated list of smileys.</td></tr>
</table>

4.3.2 Games

coin
Heads or tails?

dice <dices>d<sides>
Roll `dices` dices, each having `sides` sides. Lists result for each dice separately. The sum will be between `dices` and `dices` x `sides`.

eightball [question]
Answers a question. But don't expect the bot to pass

monologue [channel]
Check how long your monologue on the `channel` is, in case you are lose count. This is probably my favourite useless command.

roulette ["spin"]
Russian roulette. If `spin` is given, spins the chambers. This isn't really necessary since it will be done automatically at the end of the round. Provides a different experience if the bot is opped.

4.3.3 Nickometer

`nickometer [nick]`: Give an objective evaluation on the lameness of `nick`, or your nick if not provided. This is similar to the same command in blootbot.

output

```
The "lame nick-o-meter" reading for "1eEteStWaReZL0rD[69X~" is 99.98%.
```

4.3.4 Quote

quote add [channel] <text>
: Add `text` as a quote for `channel`.

quote change [channel] <id> <regexp>
: Change quote `id` on `channel` using `regexp`. For example, `s/foo/bar/g` changes all instances of `foo` to `bar`.

quote get [channel] <id>
: Show quote # `id` for `channel`.

quote random [channel]
: Show random quote from `channel`.

quote remove [channel] <id>
: Remove quote # `id` from `channel`.

quote search [channel] [--{regexp,by} value] [glob]
: Find quotes from `channel` matching regexp if provided, added by nick `by` if provided, and matching `glob` if provided. `glob` may contain wildcards (eg. `foo*bar`), while `regexp` is a regular expression (eg. `/la.*laa/`). Note that you can really provide both regexp and the glob - both must match in that case.

quote stats [channel]
: Show how many quotes there are.

4.3.5 QuoteGrabs

This plugin allows users to *grab* the most recent line of another nick as a quote.

quotegrabs get [channel] <id>
: Show quote # `id` .

quotegrabs grab [channel] <nick>
: Save last line from `nick` as a quote.

quotegrabs list [channel] <nick>
: List quotes for `nick`, newest first. This does not display full quotes, but part of each, along with the id.

quotegrabs quote [channel] <nick>
: Show last quote of `nick`.

quotegrabs random [channel] [nick]
: Random quote from any nick, or `nick` if provided.

quotegrabs search [channel] <text>
: Show quotes containing `text`. Unfortunately, this does not list the nick or allow searching only given nick's quotes. Id and quote content will be shown.

Table 4.4: plugins.QuoteGrabs

Key	Default	Description
#randomGrabber	False	Whether to grab random lines as quotes.
#randomGrabber.averageTimeBetweenGrabs	864000	Average number of seconds to wait between random grabs. When half of this has passed a random grab may occur.
randomGrabber.minimumCharacters	8	Minimum characters needed to consider a line eligible for random grab.
randomGrabber.minimumWords	3	Ditto, but for words.

4.4 Third-party plugins

4.4.1 MessageParser

```
<user> your question is covered in the ,,supybook
<supybot> Follow this link to the supybook, a supybot handbook: http://supybook.fealdia.org ↩
    /

<user> I'd like a ,,(factoids search *) please
<supybot> 'conditional', 'ggc', 'ggr', 'git', 'gitrepo', 'gribblegitcontent', ' ↩
    gribblegitrepo'
```

MessageParser allows adding regular expression triggers that can match any line. Multiple triggers can match a single line, and the same trigger can match several times. This plugin is a very useful addition for support channels.

Visit the MessageParser wiki for more information.

Chapter 5

Caveats

This is a list of issues I have not yet figured out how to do, or there simply isn't a way.

- How to enforce channel modes (eg. force +ns-t for example)
- How to delete config items
- How to delete channels / networks
- Ban add does not seem to work on Freenode
- No global ban list
- No way to delete a network
- Incomplete multi-network support
  - Capabilities are not network/channel -specific, but channel-specific. If channel by same name exists in two networks, the users have same capabilities on both
  - No way to add network-specific hostmasks
  - Not possible for the bot to have different nick in different networks
- There is no command to reboot the bot; it must be done from the shell
- Capabilities/anticapabilities for commands with spaces in them are not supported

Chapter 6

Tips

6.1 How to emulate blootbot CMDs using MoobotFactoids

Thanks to Tobias "beardy" Rosenqvist for the tip!

We try to make a command factoid, like we are used to with blootbot:

```
<user> cmdtest is <action> gives $1 "(an apple|a pear)"
<bot> Ok.
```

Then we test it:

```
<user> cmdtest someone
```

But it doesn't work. However, the factoid works:

```
<user> cmdtest
  * bot gives $1 an apple
```

The Moobotfactoids plugin in supybot doesn't handle arguments, (yet) unfortunately, so you need to do it another way. Use the Moobotfactoids to do the random part(s):

```
<user> fruits is <reply> "(an apple|an orange|a banana|a pear)"
<bot> Ok.
```

Then use an alias to do the command, with arguments (the "action" command is found in the Reply plugin), here you also see a use of a nested command:

```
<user> alias add givefruit action gives "[fruits]" to $1
<bot> Ok.
<user> givefruit someone
 * bot gives a pear to someone
```

6.2 Tidier bot replies

I don't personally like the default Supybot replies: I hate the nick prefix and the way too verbose *The operation succeeded.*.

Fortunately there are plenty of settings to configure how Supybot replies. These can be listed with `config reply` and `config replies`.

The `reply` items determine how Supybot acts, and `replies` contains some messages it uses. I prefer:

`config reply.withNickPrefix False`

`config replies.success OK`

`config reply.error.inPrivate True`.

6.3 More on nested commands

Thanks to Tobias "beardy" Rosenqvist for the tip!

For those familliar with unix shells, (bash in particular), nested commands can be compared to doing command substitution, as in `$(command)`.

Nested commands are by default enclosed by square brackets (`[]`). The `commands.nested.brackets` configuration variable can be used to set these to `<>`, `{}`, or `()`.

```
<user> echo The title of the Supybot website is: [web title http://www.supybot.com/]
<bot> The title of the Supybot website is: Welcome to Supybot.com! Supybot Website
```

Another way nested commands can work, is like a pipe, if the configuration variable `commands.nested.pipeSyntax` is set to `True`.

Same example as above, but using the pipeSyntax:

```
<user> web title http://www.supybot.com/ | echo The title of the Supybot website is:
<bot> The title of the Supybot website is: Welcome to Supybot.com! Supybot Website
```

Chapter 7

Reference

7.1 Configuration

7.1.1 reply

#inPrivate
Whether to reply in private to commands given on channel.

#requireChannelCommandsToBeSentInChannel
-

#showSimpleSyntax
Whether to be extra helpful when a user fails syntax of a command.

#whenNotAddressed
Assume everyone wants to talk to the bot, eg. treat all messages as if addressed to the bot. This does not imply `reply.WhenNotCommand False` which you should set as well.

#whenNotCommand
Whether to reply when addressed with an invalid command.

#withNickPrefix
Whether to prefix the reply with the nick of the user who gave the command.

#withNotice
Whether to use notices instead of regular messages.

#mores.instant
How many messages to send initially before prompting for `more`. Default is 1.

#mores.length
How long messages can be. Default is 0, which uses rocket science to determine the maximum number of characters that can be fit into a message without it collapsing into a black hole.

#mores.maximum
Maximum number of messages to queue, default is 50.

#error.inPrivate
Whether to send errors in private instead of replying on channel.

#error.noCapability
If True, don't tell users why they can not run a command because of missing capabilities.

#error.withNotice
Whether to send errors as notices instead of regular messages.

error.detailed
Whether to show an exception or a generic error when something breaks. Mostly useful for developers.

#format.time
Format string for timestamps (`%I:%M %p, %B %d, %Y`, eg. *08:41 PM, September 11, 2008*).

#whenAddressedBy.chars
List of characters the bot will recognize as addressing (when a command is prefixed with one of them), besides the nick of the bot.

#whenAddressedBy.nicks
List of extra nicks to consider as addressing the bot, despite the current nick.

#whenAddressedBy.strings
Like the chars key, except a space-separated list of strings to accept as command prefix. This allows multicharacter command prefixes.

#whenAddressedBy.nick.atEnd
Whether to consider messages that end in the bot's nick to be addressed to the bot.

maximumLength
Maximum length of a reply message from the bot. This does not mean the length of a single message (prompting for `more`), but the whole message.

oneToOne
Whether to send replies consisting of multiple messages in a single message.

withNoticeWhenPrivate
Whether to use notices instead of private messages.

7.2 Directory tree

This section contains list of the directories and files supybot uses, and what they are for.

Note
If you modify the configuration files by hand when the bot is running, you need to run `config reload`.

`backup/`
For backups of config files.

`conf/`
Configuration files: {channels,ignores,userdata,users}.conf

`data/`
Plugin databases.

`data/#channel/`
Channel-specific plugin databases.

`data/tmp/`
Temporary plugin data files (eg. database journals).

`logs/`
Logs.

`logs/ChannelLogger/`
: ChannelLogger logs. The structure depends on the config variables.

`logs/messages.log`
: The main logfile.

`plugins/`
: Local plugin directory. This is by default listed in `config directories.plugins`.

`<bot>.conf`
: The main configuration file of the bot.

`tmp/`
: Temporary files.