

Choobogo - Choob's implementation of PogoBOT

Faux

January 12, 2006

1 Introduction

Here lies the specification of Choob's implementation (and extension) of the PogoBOT protocol, version 2. The original protocol is not well specified, which made an implementation kind of impossible, so it's just what makes sense.

The original protocol is available at <http://retout.uwcs.co.uk/pogobot2/>, credit to Tim.

2 Notation

A brief note on notation: Where necessary, I'll use C-style escape codes (for instance, `\n` for 'new line'), and Perl-compatible regular expressions, appropriately surrounded with forward-slashes and double-backslashed (ie. `/\s+/` for 'at least one character of whitespace').

3 The Protocol

3.1 The Concept

The basic functionality that the 'core' of the bot, ie. what Choobogo needs to implement, is listed here:

- The 'core' of the bot (read: the bot) listens for connections.

3.2.3 LSREG

This command lists all of the commands that are currently registered with the ‘core’. The command takes no arguments, so the syntax is `/^LSREG$/`.

The reply is not well defined, Choobogo replies with `OK LSREG` followed by a series of space-separated commands.

For instance, if the commands ‘foo’, ‘bar’ and ‘eee’ are defined, the reply will match: `/^OK LSREG foo bar eee$/`.

3.2.4 Any other command...

Any command that doesn’t match as a mentioned command will attempt to be treated as a ‘context’ (see Section 4.2.1), and, failing this, will generate an error (see Section 4.1.2).

3.2.5 EXEC

EXEC is not implemented, and will be treated as an invalid command.

4 Core Messages

4.1 Status Replies

4.1.1 OK

This message tells you that something worked. Normally it doesn’t tell you what. God help you if you’re trying to process input quicker than the time it takes for the ‘core’ to respond.

Normally the form of this is: `/^OK.*$/`.

Some commands, however, send different flavours of the OK reply. In general, this should hopefully be `/^OK {commandname} .*/`, for instance `OK LSREG foo`. See Section 3.2 for information on specific commands.

4.1.2 ERR

This message tells you that something went wrong.

Normally the form of this is: `/^ERR.*$/`.

4.2 Commands

User invocation of commands will result in a message being sent from the core.

This has the general form of `/^([^\s]+)\t([^\s]+)(?:\t(.*))?$/`, ie. two or three tab-separated strings, of which the first (or only) two may not contain whitespace.

The arguments:

1. The ‘context’.
2. The command that was invoked by the user.
3. Optionally, any arguments the user passed.

4.2.1 Context

This is the reference to the message that the ‘core’ has sent to you. It forms the core of the reply (see Section 4.3). It has no defined format, Choobogo currently uses random floating-point numbers.

4.2.2 Command

This is the name of the command you invoked, and is provided so the plugin can work out what it is supposed to be doing.

4.2.3 Arguments

Optionally, the arguments that the user passed to the command, for the plugin/command to process.

4.3 Replying

Once you have received a message, you may want to reply to it. You can do this by simply messaging the ‘core’ with the context you received, and your reply. This should match: `~/([\s]+)\s+(.+)$/`.

This will send a reply to the user you received the message from, in the correct context. If they pmed the bot, they’ll receive the reply in pm, if they did it in a channel, they’ll receive the reply in the channel, with their nick prefixed.

You may only reply once to each context.

An example reply, assuming you recently received (from the ‘core’):

```
$co44286372nt$ calc 5*3+11
```

might be:

```
$co44286372nt$ The answer isn't 17!1111
```

5 Example Plugin

To try and make the specification less confusing, here’s some sample code that implements a basic plugin.

5.1 Sample Code

A sample plugin, written in a completely made up language:

```
print "REG isone"  
print "REG diediedie"  
  
loop while ( if stdin isn't end-of-file , read a line of it into $a )  
{  
  $context = bit before first tab of $a  
  $command = bit after first tab in $a  
  $args = rest of $a, ignoring tabs.  
  
  if ($command is "isone")  
  {
```

```

    if ($args is "1")
    {
        print line ($context followed by " It 's one!!!!!!!!")
    }
    else
    {
        print line ($context followed by " Not one :(")
    }
}

if ($command is "diediedie")
{
    exit
}
}

exit

```

5.2 Using the Sample Plugin

If a plugin is unable or too lazy to talk to the ‘core’ directly, it can be wrapped in netcat.

Netcat is available for most OSes, including Windows and Linux.

Usage: `nc -e {command to execute} {server to connect to} {port to connect on}`.

So, if you have a binary plugin called, say, ‘foo’, and are trying to connect to a local instance of Choobogo, which is running on port 3090 (the default), you’d use the line:

```
nc -e foo localhost 3090
```

6 Conclusion

The protocol is fine, unless something goes wrong, then you’re screwed. Enjoy.