Client-Server Communication Protocol (RMCP)

Commands are always sent from client. Upon being received by the server, server does an immediate response.

In this document, '[' and ']' surround areas where variable content will be inserted.

& symbols denote a single ASCII character with a special code. (eg: &EOT)

Response Syntax

`<code> <message>:<args>` (Optional)

Response Codes

Code	Message	Special Syntax	Purpose
100	Done	-	Denotes the end of a connection, whether the command was successful or unsuccessful.
101	Success	`101 success:[message]`	A command completed successfully with a success message to show.
102	Ok, text	-	Command received and ascii data will be sent (until an ascii &EOT character is received).
103	Ok, bytes	`103 ok, bytes:[message length in bytes]`	Command received and byte message of specified length will be sent.
420	Hello	`420 hello:<port>`	New session created on port <port>
300	Catastrophic failure	-	Something happened, and the command failed to execute.
301	Failure	`301 failure:[message]`	Something went wrong in executing a command, the defined message is to be displayed.
302	Invalid command	–	An unrecognised command was sent.

Request/Response Parsing

- Spaces separate the parts of a request, including separating the command name from parameters and parameters from each other
- Surrounded by Quotation marks ("):
  - mean to parse new line characters as part of the parsed string
  - mean to include space characters as part of the string being sent

Commands

- **List scripts**
  The client asking for a list of all scripts that can be executed by the user on the client.
  Syntax: `listScripts`
  Response (text):
  ```
  [script short name 1] &NEWLINE
  [script short name 2] &NEWLINE
  …
  ```

- **Run script**
  The client asks for a particular script to be run.
  Syntax: `run [script short name]`
  Response (text): Copy of stream of output from script. Sends termination characters (as above) when the script stops.

- **Kill script**
  Usually caused by the user returning to the script list. Destroys the currently running script.
  Syntax: `kill`
  Response: None. Sends only basic message acknowledgement messages.

- **Get Script Icon**
  Plugins have associated icons. If the client does not have an icon cached already, it will need to fetch this icon from the server.
  Syntax: `getIcon [script short name]`
  Response (bytes): Sends the byte stream of the image (so essentially a file transfer).

Special Case for creating a session

When a client first connects to the server it needs to create a new ‘session’ through which to give commands to the server. As each command is sent over a new TCP connection the server needs to inform the client as to which port to use to send commands over. This interaction is handled by the ‘hello’ command. When a client first connects to the server it must send the crea

Example Communications (to be used as part of testing)

Example 1 - client gets script listing

```
C: listScriptsRequest
S: 102 ok, text
S: RebootSystem
S: ClearCache
S: SkipSong
S: &EOT
S: 100 done
```

Example 2 - client runs script

```
C: run SkipSong
S: 102 ok, text
S: Your current song is now, Friday, by Rebecca Black
S: &EOT
S: 100 done
```

Example 3 - client runs script but returns to the script screen before fully responding

```
C (connection 1): run lsAll
S (connection 1): 102 ok, text
S (connection 1): cats
S (connection 1): dogs
S (connection 1): monkeys
C (connection 2): killScript
S (connection 1): &EOT
S (connection 1): 100 done
S (connection 2): 100 done
```

Example 4 - the client runs an invalid script

```
C: run RmRfAll
S: 301 failure:Invalid script!
S: 100 done
```

Example 5 - the client sends an invalid command

```
C: runnnnzzzz
S: 302 invalid command
S: 100 done
```

Example 6 - the client gets a script icon

```
C: getIcon MuchReboot
S: 103 ok, bytes [length]
S: [http://goo.gl/qUimhf in bytes]
S: 100 done
```