CocoMUD client - Feature #13

Feature # 9 (Open): An easy yet powerful setting system for customization

Create the CocoMUD sharp script

09/21/2016 08:01 PM - Vincent Le Goff

Status:	Closed	**% Done:**	100%
Priority:	High		
Assignee:	Vincent Le Goff		
Category:	Customization		
Sprint/Milestone:	1		

Description

CocoMUD client provides configuration through its dialog box. It also needs to implement the second and third layer of customization through scripting and coding. To do so, a particular syntax must be defined.

Like most MUD clients, CocoMUD will use the hash (or sharp) symbol to indicate client configurations. This script can be found inside configuration files of all types, including the **.set** files, which will be read by the Client but not modified. For readability purposes, this scripting language (called CocoMUD sharp script) should allow simple actions, extendable through code. The definition of aliases and triggers will be used here.

Syntax of the sharp script

Lines should begin with a hash (or sharp) symbol:

```
#line
```

This will indicate to the client that the line shouldn't be sent to the MUD, but processed by the sharp script engine. The syntax of a line will follow this rule:

- Just after the hash symbol should be the name of the action to be performed (for instance, #alias, #macro, #trigger, ...).
- All arguments should be separated by a space when on the same line.
- Arguments containing space should be enclosed in braces ({}). This syntax also supports multi-line arguments.
- Expressions beginning with {+ allow raw Python code to be written in the script.
- Flags can be inserted in the list of arguments by preceding them with a minus (-) or plus (+) sign.

Examples

Macro

When the F1 key is pressed, go north:

```
#macro F1 north
```

Same macro, but using Ctrl + F1:

```
#macro Ctrl+F1 north
```

Or, more readable, with spaces (note that we have to enclose the shortcut in braces this time):

```
#macro {Ctrl + F1} north
```

The braces are necessary because the argument contains a space. If the braces are omitted in this example, #macro is given 4 arguments (instead of 2).

Alias

An input containing only 'l' will redirect to the 'look' command:

```
#alias l look
```

Splits the alias on several lines, we use braces:

```
#alias h {
    ooc greetings
    say hello all
}
```

Note that the indentation in that case isn't mandatory: it's there merely for readability.

This alias could also be written like this:

```
#alias h {ooc greetings;say hello all}
```

This time, we write everything on the same line and separate commands with a semi-colon (;). This syntax is less readable but can still be useful in some contexts.

To define an alias beginning with 'gr ', taking an argument and sending it both to the say and ooc command:

```
#alias {gr *} {
    ooc %*
    say %*
}
```

An alias that bends a bow with an arrow (specified as argument), wait for 3 seconds and then shoot it:

```
#alias {b *} {
    bend bow with %*
    #sleep 3
    shoot
}
```

Trigger

A simple trigger that waits for the messages sent on the **ooc** channel and plays a sound:

```
#trigger {[ooc]*} {#play ooc.wav}
```

Whenever a line beginning with [ooc] appears, the sound file "ooc.wav" is played.

Whenever the line 'You received X XP' is displayed, play a sound depending on the number of XP (Python):

```
#trigger {You received {xp} XP.} {+
    if xp.isdigit():
        xp = int(xp)
        if xp > 500:
            play("victory.wav")
        elif xp > 100:
            play("good.wav")
        else:
```

```
            play("some.wav")
}
```

This trigger may be a bit more complex to understand at first glance:

- We capture the message 'You received {xp} XP.'. {xp} is replaced by something (probably a number, but we don't know that for sure). The variable xp is created and passed to the script.
- The second argument of #trigger is surrounded by braces. Note, however, that the opening brace is followed by a + sign, indicating that the following lines are Python code, not sharp script.
  - The first line of our Python script checks whether xp contains only digits. It's better to check that it's a number before converting it (every argument passed to the script is a str).
  - We then convert it into an int. Note that the indentation is not only for readability this time, Python needs it.
  - We then have a condition triggered if the XP is greater than 500. If so, we call the play function. It has exactly the same behavior as #play in sharp script (it's the same function, actually) and will play the sound given in argument.
  - We play different sounds in other cases.

Including Python code in sharp script is always possible, although not always useful. It allows to easily extend the sharp script with features, while remaining pretty readable. It's also possible to add sharp script functions (that would become Python functions).

Flags in functions

We can also add flags, adding or removing specific features from functions that support them. Consider the #say function, which takes a mandatory argument: the message to be displayed on the client. This message will also be sent to the screen reader (and the Braille display), if supported.

```
#say {That sounds terrific!}
```

One could prefer not to display this message on the screen, but to send it to the screen reader regardless:

```
#say {That sounds terrific!} -screen
```

Or one could send it to the screen reader but not to the Braille display:

```
#say {That sounds terrific!} -braille -screen
```

Default values of flags depend on functions calling them. For the "#say" function described here, three flags could exist:

- The flag braille that would send to the Braille display, if supported (default to true).
- The flag speech that would send the message to the screen reader to be spoken (default to true).
- The flag screen that would display the message on the screen, as if it were received from the server (default to true).

Examples of syntax could be:

```
#say Hello -braille
#say Hello +screen -speech
```

Associated revisions

Revision 05cb04d9 - 12/14/2018 09:14 AM - vincent-lg

Merge pull request #13 from francipvb/i144

ANSI colors processing fixed

History

#1 - 09/21/2016 08:03 PM - Vincent Le Goff

- Status changed from Open to In Progress

#2 - 09/21/2016 08:05 PM - Vincent Le Goff

- Sprint/Milestone set to 1

#3 - 09/23/2016 05:14 AM - Vincent Le Goff

- % Done changed from 0 to 20

The basic hierarchy of the SharpScript has been developed. The #say function has been provided to test. One can test it in the client, although this feature will most certainly disappear after some time:

```
#say {That works so well!}
```

That will display the line "That works so well!" in the client's window, and will send it to the screen reader (and the Braille display), if supported.

#4 - 09/23/2016 07:56 PM - Vincent Le Goff

- % Done changed from 20 to 40

The syntax is now validated by Unittests. This process ensures that the modification to the SharpScript engine will not break compatibility (commit b57dc8cb9ec640a99b61f820955949965b4b211a).

#5 - 09/23/2016 10:17 PM - Vincent Le Goff

- % Done changed from 40 to 50

A basic trigger system has been added in SharpScript (commit f6bb9547d74d678f37f2e76351f24785a6f29609). It already supports SharpScript converting:

```
#trigger {* tells you:*} {#play tell.wav}
```

This syntax has been tested and works perfectly well. Management of Python code remains to be tested and approved.

#6 - 09/28/2016 09:25 PM - Vincent Le Goff

- Description updated

- % Done changed from 50 to 60

Add a flag system with - and + sign (commit 95c4bc24f47c3237350fd7b99b27328fd8f6f86f).

#7 - 09/29/2016 01:41 AM - Vincent Le Goff

- Description updated

- % Done changed from 60 to 80

Inserting and executing Python code has been tested (commit ed87374309e7ec9db62df4ab512feee9a319e7cc).

#8 - 09/29/2016 06:59 AM - Vincent Le Goff

- Status changed from In Progress to Closed

- % Done changed from 80 to 100

Semi-colons in arguments (plain text or SharpScript) are now handled properly (commit 457ab841a0904ddbc5a9afacbe4beeaa9fd8a8c3).

There's also a documentation written in the SharpScript section on the wiki.