# Scripting in CocoMUD client

CocoMUD client offers a simple yet easily-extendable scripting language. This language can be used to describe macros, aliases or triggers, or more complex features in the MUD.

This document describes the syntax of SharpScript, gives examples and provides frequent questions at the bottom of each section. If you want the answer to one of these questions, just click on the question, the answer will appear on the next line.

## The SharpScript logic

The logic of the SharpScript can be summarized in two main ideas:

- Have a very easy-to-write language to perform simple functions ;
- Allow to include Python code in this language to perform more complex tasks.

As you will see, the syntax of the SharpScript is pretty light, but it already allows interesting features. Should you need more, Python is here, and it's not exactly a limited alternative.

## Basic syntax

The SharpScript finds its name in its syntax. As most MUD clients, a SharpScript command begins with a sharp symbol (#). It can already feel like a constraint of some type, but maintaining the difference between commands to the server and to the clients is important. Unlike most MUD clients, CocoMUD client tries to not force symbol in user input. If you want to send a message beginning with the sharp symbol, you can do so, unless you configure your client to interpret SharpScript as input.

## Commands

SharpScript features are kept in commands (or functions). Both terms refer to the same thing in this documentation. These commands can ask to create a macro, an alias, a trigger, send some message to the server, display some message to the client, prompt the client with a question, store some information about a character and so on.

Here's a single example:

```
#say Hello!
```

If you try this piece of SharpScript in an input that accepts SharpScript syntax, the client should display "Hello!" at the bottom of your screen. The text should also be sent to the screen reader, spoken by it and displayed on a Braille display, if supported.

The #say command that we have used is very simple: It expects one argument, one information, which is the message to be displayed.

Frequent questions:

How to send a command with a sharp symbol (#) in it?How to send a command with a sharp symbol (#) in it?

By default, CocoMUD client doesn't interfere with your playing. When you are in the input field on the client, you cannot enter SharpScript unless you enable that setting. So you can type about every symbol you want, none of them will be interpreted by the client.

However, at times, you really want to send sharp signs to the client while having SharpScript interpret part of your commands. To do so, you must precede the sharp sign (#) with another one. This syntax is only necessary at the beginning of a command or an argument:

```
#say {{##I'm saying something with a #.}
```

This will display: #I'm saying something with a #.

Notice that only the first sharp symbol had to be kept twice (at the beginning of the argument). The other (at the end) didn't need to be escaped.

```
##forward
```

This will send #forward to the client.

## Arguments with spaces

If you try to display a message with spaces, it will not work:

```
#say This character isn't feeling so well.
```

Some commands take more than one argument, and to separate them, they use the space (we will see examples a little below). Therefore, if you want to have spaces in your argument, you should surround it by braces ({}) :

```
#say {This character isn't feeling so well.}
```

Surrounding arguments by braces is only necessary if this argument contains spaces.  Consider the following example, to create a macro:

```
#macro F1 north
```

This time, the #macro command expects two arguments:

- The shortcut to which this macro should react.
- The action to be performed.

Here, when we press F1, the client will send "north" to the server.

Remember to enclose the arguments containing spaces, however:

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

This time, the shortcut is Ctrl + F1. Because there are spaces in this argument, we enclose it in braces.

```
#macro F8 {say Greetings!}
```

When we press F8, the client will send "say greetings!" to the server.

```
#macro {Ctrl + Shift + K} {look into backpack}
```

Since both arguments contain spaces, we enclose them both.

Notice that if you have a doubt, use braces.  It will work regardless:

```
#macro {F1} {north}
```

## Multi-line scripts

By default, SharpScript expects every command to be on a different line.  This is not always a good thing for readability's sake, and sometimes it can get really complicated.

Let's say we want to create the alias as follows:  When we enter "victory", the client plays a sound and sends a few commands to the server:

```
#alias victory {
    #play victory.wav
    say I've done it!
    north
    #wait 3
    sheathe sword
}
```

The second argument is split on several lines, because it's much more readable.  Notice here that the argument contains SharpScript commands (beginning with a sharp symbol) and commands to be sent to the server.  The lines not beginning with a sharp symbol (#) are sent as it to the server.  This is the case for the line 3 (say I've done it!) for instance.

For readability, the second argument is indented a little on the right:  Each command in this second argument stands 4 space on the right.  This is not mandatory, it just makes things easier to understand.  Since Python relies on indentation however, it might be a

good thing to get used to it, regardless of its being necessary or not.

Frequent questions:

Can I put several instructions on a single line?Can I put several instructions on a single line?

You can, although it might not be very readable. The syntax to do so is to use semi-colons to separate commands on a single line. The previous example could be written on a single line like this:

```
#alias victory {#play victory.wav;say I've done it!;north;#wait 3;sheathe sword}
```

As you can see, it's not as readable, but this syntax may sometimes be useful.

If you want o write a semi-colon in your SharpScript command, just put two semi-colons instead of one:

```
#say {I would like to display something;; but I'm not sure what.}
```

## Flags

Some commands support flags: Flags are here to influence the behavior of a function in some way. The best example available at this time is the #say command we have seen. By default, it displays the provided text, sends it to the screen reader to be spoken, and to the Braille display to be displayed. There are three flags that control that:

- "screen": Should the text be displayed on the screen (as if it were coming from the server)? If you don't change it, it's on by default.
- "speech": Should the text be sent to the screen reader to be spoken aloud? Once again, if not changed, it's on.
- "braille": Should the text be sent to the Braille display? Again, this flag is on by default.

You can change flags given to a command at the end of the SharpScript line (or instruction). To set a flag on, write its name after a plus sign (+). If you want to set this flag off, write its name after a minus sign (-).

```
#say {I don't want it to be displayed.} -screen
#say {And that shouldn't be spoken nor displayed in Braille.} -speech -braille
#say {This may be displayed on screen and on the Braille display.} +screen -speech +braille
```

Notice that the flags "screen" and "braille" are not necessary in the last example: Both are on by default. This example is here to illustrate the syntax.

## Embedding Python into SharpScript

Sometimes, what we want to do is a bit too complex in SharpScript. It's possible to extend its syntax and bring new commands into it, but it's better to keep it simple and to learn to do more complex things with Python, which is a highly-readable language without few limitations. It's still a good thing to keep your script readable, not only for you (although it might be handy, should you modify it), but to potential users.

To add Python code, use the syntax for long arguments (with braces), but after the left brace, add a plus sign (+). This tells the client that what follows between the braces isn't SharpScript, but Python code.

If we want to write a script that plays different sounds depending on the XP we receive, we might do it that way:

```
#trigger {You received {xp} XP.} {+
    # The 'xp' variable contains the received XP
    # It might be a number, but we have to convert it
    if xp.isdigit():
        if xp > 200:
            play("victory.wav")
        elif xp > 100:
            play("notbad.wav")
        elif xp > 10:
            play("notalot.wav")
}
```

This trigger will wait for the line "You received *** XP." and will put whatever XP in the 'xp' variable, before passing it to the Python script. The Python script will convert the XP (if it's a number) and will play a different sound:

- If the received XP is over 200, it will play "victory.wav".
- If it's between 100 and 200, it will play "notbad.wav".
- If it's between 10 and 100, it will play "notalot.wav".

Notice that nothing happens if you receive less than 10 XP in this example.

It's very useful to embed Python code into SharpScript that way.  It makes for clear and readable scripts that are almost limitless.  Keep the indentation in this example, as it will be used by Python to determine blocks.

Frequent questions:

Which functions are available in embedded Python?Which functions are available in embedded Python?

All SharpScript commands are available as functions.  That's why you can use the #play or #say command.  Inside of Python, the commands are not preceded by a sharp sign and are just respect the function syntax:

```
say("Could you display that?")
say("After all, just speak that.", screen=False)
play("sound/file.ogg")
```

What variables are available in embedded Python?What variables are available in embedded Python?

Python scripts share their variable across the entire game setting.  This can sometimes be confusing, but it also prevents from bad headaches if you remember that no variable defined in a script will magically disappear unless you close the program.  Therefore, if you have a script like this:

```
#alias todo {+
    health = 38
}
```

The variable 'health' will be available in all other Python scripts.

In some cases, other variables are defined by the client.  For instance, the #trigger command creates variables depending on the trigger.  For more information, read the section about triggers.