Aliases in CocoMUD

Aliases are a special feature of MUD clients that can be used to shorten commands. An alias is itself a command, but instead of being sent to the server as is, it is first analyzed by the client, modified and extended if necessary. Some aliases don't leave the client at all.

This document will introduce the concept of aliases and explain how to create and manage them.

Adding an alias

You can create an alias using the interface or through the SharpScript syntax. The interface being easier at first glance, it will be explained in first.

Through the dialog box

In the menu bar, select Game -> Aliases. You will find yourself in a dialog listing the current aliases for this world. Note that aliases are usually created in a world, being shared by characters. This will change in future versions.

In this dialog, you can add, edit or remove aliases. By default, the list of alias will most likely be empty, so you can select **Add** to add one.

You will then have to choose an alias name. This is the command you want to create, basically. In this example, we're going to create a "tts" alias that enables/disables the text-to-speech in the client.

Enter "tts" in this field. When you press Tab, you will be presented with a list of actions to be performed when you type this command. For instance, you may want the "rs" alias to send two commands at the MUD server: "reload" and "shoot". In this case, you would select the action named "send one or more commands to the server".

In our example, we'll select "Enable/Disable TTS", since that's what we want to do when we type "tts" in the client. Tab again and click the **Add action** button.

You will be prompted with a dialog asking you to review parameters to this action. For this action, there is none, so you can press "OK" to add the action.

The new action has been added. The cursor will be put in the list of current actions linked to this alias. You can link an alias with several actions (even with the same action several times, to play several sounds, for instance, although it might not be very useful).

The list of current actions may be a bit scary. It contains summarized information. If you have selected the "Enable/Disable TTS", it will create the action #tts. That's a shortcut, a notation used by the SharpScript engine, and you don't need to worry much about it, unless you want to play with the scripting system in CocoMUD.

To summarize the dialog when you add or edit an alias, you will find:

- The alias' name (a text edit).
- A list of actions that are currently linked with this alias. If the alias isn't linked with any action, the list won't appear.
- A button to edit the selected line of action. If the alias isn't linked with any action, the button won't appear.
- A button to remove this line of action from the alias. If the alias isn't linked with any action, the button won't appear.
- A list of actions that can be linked to this alias.
- The button to add this action.

You should get accustomed to this dialog, as it will be present for most configuration involving SharpScript. It is, in fact, the SharpScript editor that do not ask you to edit the configuration file by hand. You will find more examples in the rest of the documentation.

To save the alias, don't forget to select "OK" several times, until the dialog box is closed.

If you then type "tts" in the client, you should see (and hear):

```
TTS off.
```

Type it again to switch it on. The text-to-speech will be disabled or enabled using this alias.

SharpScript syntax

You can also add an alias by editing your configuration file and adding it there. This solution might be preferred by some.

Open your configuration file for this world. You will find it in the "worlds" directory. Select the directory containing your world, then the "config.set" file. You can open it with a small editor like notepad.

To add an alias, use the #alias action with two parameters:

- The alias' name.
- The alias' actions to be performed.

If you want to create a "tts" alias that will enable or disable the text-to-speech, add in your file the following line:

```
#alias tts #tts
```

If you have several actions, don't hesitate to describe the alias on several lines:

```
#alias go {
    #play sounds/go.wav
    #say {Here we go!}
}
```

For an explanation of the SharpScript syntax, refer to the section describing SharpScript.

Editing an alias

At any time, whether you have created the alias in the dialog box or in the configuration file, you can edit it. If you have created it in the configuration file, you can modify it through the interface, and vice versa.

Remember that the configuration is loaded when you select the world, which will most likely happen when you open the software. If you modify the configuration file, restart the software to take it into account.

Removing an alias

You can remove an alias through the dialog box or by editing the configuration file. If you do it through the dialog box, remember the changes will be lost if you close the dialog but do not use the "OK" button.

Aliases with variables

CocoMUD provides a system of variables. They can be used anywhere in your script or in any feature using the SharpScript engine (aliases, macros, triggers...).

Variables can be more than useful for aliases to create shortcut commands with unknown parameters. For instance:

> I want to create an alias that begins by =. Everything after the = sign should be sent to the server using the say command. If I type =hello ! in the client, then it should send say hello ! to the MUD.

For this case study, we will use variables. The alias begins by an equal sign (=), but we don't know what's after that. How to handle that context? We will use the * sign in our alias name, which means "about everything, as long as necessary".

So if we enter =* as an alias name, CocoMUD will understand it as meaning "everything starting with an equal sign".

But we don't only need for the alias to be activated. We need for it to send a command with what has been entered after the equal

sign. Here is the process step by step, explained below:

- In the menu bar, select **game** then **aliases**.
- Click the **add** button to add a new alias.
- In the alias field, enter =* (an equal sign and the asterisk sign).
- Tab to move to the list of actions that can be linked to this alias.
- Select "send one ore more commands to the server", tab and click the **add action** button.
- You'll be asked to configure this action. Tab until you hear "Commands to be sent to the server". In this field, type say $1 (the $1 will be explained below).
- Select **OK**. The action has been linked, and you should see it in this field:

  ```
  #send {say $1}
  ```

  What it means will be detailed below.
- Tab until you find the **OK** button and click on it. You will find yourself in the list of aliases, press **OK** again.

If you type in the client:

```
=hello !
```

The alias will send "say hello !" to the server.

What happens is pretty simple:

- You enter "=something".
- CocoMUD identifies a matching alias.
- The "something" part (after the equal sign) will be captured and put in a variable, called $1.
- When you send "say $1" to the server, CocoMUD replaces it $1 with the "something" part.

The variable is called $1 because you can create a lot of variables. If you enter the alias:

```
-*+*
```

You will be able to send:

```
remove $1
wear $2
```

And then type:

```
-cloak+armor
```

Variables in SharpScript are described more completely in the section describing SharpScript.

CocoMUD basics

CocoMUD is a MUD client specifically designed to enhance accessibility with screen readers. It supports a native TTS (Text-to-Speech) for most common screen readers on Windows, as well as Braille.

This document aims at explaining the basic features of CocoMUD client in a practical way, to tell you what you can do with it and how to do it. More complex features will have a document of their own.

When opening CocoMUD

When opening CocoMUD, it should display a list of configured servers (or worlds, in MUD client terminology). It's a list, you can select another world by pressing the arrow keys or move more quickly by typing the first letters of a server. It's possible that the server you would like to connect to doesn't appear in the list: you may need to add it before going on.

When you have selected the world to which to connect, press RETURN (or click on the **connect** button).

The client's main window

CocoMUD client's main window is as simple as it can be. There's only one text field, through which you can navigate using the arrow keys, select and copy text, move at the beginning of the end of the window, and so on.

If you start typing in this text field, the cursor is moved to the last line. Consider this text field like a read-only area, except for the last line. You can type your command as usual and press RETURN to send it. You can then arrow up to see the result of your command (and up again to see previous messages). This should be quite straightforward after just a minute or so.

When you press RETURN on a world, CocoMUD tries to connect to the server. If everything goes well, the client connects and the welcome message of the server is displayed (you can read it with the arrow keys). The welcome message should also be sent to the screen reader (and, if you have a Braille display, you should see it there as well). Most MUD servers will ask for your username and password.

The command history

Whenever you press RETURN to send a command, it is added into the command history. You can browse through the command history to see what has been send, send a command again or even modify a command you have entered. There are two ways to use the command history:

- Using the CTRL + Up and Down arrow keys, you can browse through the command history. If you press CTRL + up arrow key, you will be prompted with the command you entered previously. You can press RETURN or modify this command at will. Press CTRL + down arrow key to go down into your command history.
- Using the lock mode. To enter into lock mode, press Escape. You can now navigate through the history using the arrow keys. You have to leave the lock mode to see the result of your commands, by pressing Escape again.

Tab-completion

CocoMUD client supports tab-completion. If you start typing a word, you can press tab and the client will try to finish it, based on what you have received during this session. For instance, if you type "st" in the client, then press tab, the client will try to complete the word with what you have received. Perhaps it will suggest "street" or "status". Words that are more frequent will be suggested first.

When you press tab, the word will be written under the cursor. You can continue typing or press RETURN to send the full command. If you press tab again, the client will look for another possibility beginning with the same letters you had initially entered. You can press tab as many times as you want, the client will write suggestions from most frequent to less frequent. If the client can't find anything, it will just enter your initial letters and you'll have to type the rest of the word.

Command stacking

Command stacking is a feature that allows users to send multiple commands at once. For instance:

```
say sounds;say good
```

This syntax is equivalent to writing:

```
say sounds

say good
```

Sending multiple commands at once can be useful in some situations. By default, the character for command stacking is the semicolon (;), but it can be changed in the preferences, input tab. If you wish to remove command stacking, simply remove the character in the setting.

You can double the command stacking character to actually send one. If you have set the semicolon (;) as a delimiter for command stacking (the default), you can type:

```
say one command;say another command with a wink emoticon ;;)
```

This command is equivalent to writing:

```
say one command

say another command with a wink emoticon ;)
```

Once the delimiter is present more than one time, it is not used as delimiter in the command stacking. Therefore:

Command	Equivalent
say ;;)	say ;)
say ;;;)	say ;;)
say ;;;;)	say ;;;)

The menu bar

Some options are accessible through the menu bar. In File -> Preferences, you will see some settings that you can modify to have your experience with CocoMUD more comfortable.

The preferences

When you select File -> Preferences in the menu bar, or press *Alt* + *Enter*, you should see a dialog with several tabs. The first tab (selected by default) is **general**. It only contains the language selection for the time being. CocoMUD should be in the language of your system, if it is translated in this language. Otherwise, English should be selected.

In the **display** tab, you can select a different encoding. By default, CocoMUD is set to use a Latin encoding, but you can change it. If you connect to only-English MUDs, you may not need this setting.

In the **input** tab, you can change the delimiter for command stacking.

In the **accessibility** tab are several options that affect accessibility. Here they are, in more details:

- Enable TTS (Text-to-Speech): you can here disable the TTS (Text-to-Speech). The TTS is what sends the content of the output field to your screen reader. Sometimes, it's really not necessary, and you can turn it off.
- Enable TTS on a different window: by default, CocoMUD client enables its TTS even if you're not currently in the CocoMUD application. It means that, you can be reading a web page, or a book, or sending an email, and suddenly your screen reader

starts on the messages you have received on the MUD. As this can be useful, this can sometimes be annoying, so you can turn this option off to let TTS only speak when you are in the CocoMUD client.

Adding a world

If you want to add a new world, in the world selection (when the client opens), select the **add** button. You will be asked three information:

- The world's name: what name should the world be given? Most likely, you will use the MUD's name, like Alter Aeon or T2T.
- The server's hostname.
- The server's port number.

The name, hostname and port number can be changed if needed. Notice, however, that the name is used to determine the location. This will be a directory created for the world in which settings are stored. If you change the name, the location won't be changed.

Once you have created a new world, it will appear in the world list and you will be able to connect to it.

What's new in recent builds

This page describes the new features added and changes made to each build. You can browse through each build using headings.

Build 49

- CocoMUD is now available in Spanish! Many thanks to the contributor who helped in translating CocoMUD in this language and offered additional help during this build.
- CocoMUD now runs on Python 3, instead of Python 2 (#128). Some minor errors with compatibility may still exist, please report if you find any.
- Macros and other scripting configuration shouldn't run in the wrong window, resulting in conflicting game play when running the same world in different tabs (#136).
- CocoMUD now supports playing audio files in .mp3, .wav and .ogg formats. The old dependency on Pygame was replaced with a much lighter and more efficient library (#134).
- The console window is now hidden whenever updating (#135). This is still a work in progress in designing a better updating system.
- The Python console was removed from the **File** menu. A new menu named **Tools** was added to support more advanced tools.
- A SharpScript console was added in the **Tools** menu. If you have been using CocoMUD for a long time, you may remember one could enter SharpScript commands (like #say) directly in the client. This was removed as it created issues. Now you can enter your custom configuration in this SharpScript console (#138).
- CocoMUD now takes an optional command-line argument to run a different configuration directory. That is, you can run CocoMUD's version but give it a different directory than the current directory. This feature is not likely to be used by many but it now exists (#139). For instance: cocomud.exe --config-dir=D:\CocoMUD
- The notification counter used to increase even with messages that were interpreted by mute triggers. This is now fixed (#127).
- CocoMUD used to crash on opening if no Internet connection was available. This is now fixed (#132).
- A regression problem was fixed, which prevented to load and run characters (#141). Also note that your world and character configuration (mainly your config.set files for the time being) will be converted to the utf-8 encoding from now on, since this encoding is much more appropriate to the international spreading of CocoMUD.
- Errors in the Python console weren't displayed correctly. This is now fixed (#142).
- Colors in game weren't always displayed. This is now fixed (#144).
- Note for developers: CocoMUD now runs and build with pipenv, which makes it much easier to develop and support all CocoMUD's dependencies. Some additional documentation will shortly be posted.

Build 48

- A new option in the menu bar (**Game** -> **Channels**) allows to edit channels. You will find the documentation on channels here (#120).
- A new sharp script function #randplay has been added, to play a random sound in a list. This function is usable but not yet present in the graphical user interface.
- Some work has been done on general configuration. This will be available on the next update.
- One can now clear the output window through the menu **Game** -> * Clear the output window* (#64).
- Several fixes on importing worlds from the website (#123).
- Fixes the error in changing preferences (#125).

Build 47

- You can now easily export a world in a ZIP archive, choosing what you want to export (**File** -> **Export this world**). You can

share this file and others can import it through **File** -> **Import a world** (#86).

- CocoMUD better handles the clipboard and pasting, particularly under Linux (#106). It also provides a checkbox to disable auto send when pasting text (#91).
- CocoMUD now supports and displays both **wav** and **OGG** files for playing sounds (#112).
- Rich-text was disabled by default to offer a better user experience with fewer bugs (#114).
- Several bug fixes and improvements under Linux (#105, #106).
- CocoMUD now opens and correctly closes preferences when the user language isn't supported (#108).
- Importing a world from the website doesn't generate encoding errors anymore (#117).
- CocoMUD now displays text using a more appropriate font, thank you again, sighted contributors.
- Note: for compatibility reasons, file encoding was updated. CocoMUD should support older versions of configuration files and convert them at startup, but there might be issues with some heavily modified configuration with special characters.

Build 46

- Command stacking is now working again and systematically checked (#98).
- Command stacking now supports special characters (#99).
- CocoMUD better handles variables if their value contains special characters (#93).

Build 45

- CocoMUD no longer makes the screen reader freeze (#96).
- CocoMUD now supports the SSL protocol, to encrypt the telnet connection (#94).

  SSL is not supported by all games, and is usually set on a different port than the plain telnet protocol.

- Add a new menu item to disable trigger sounds (#89).

Build 44

- Add a SharpScript action to repeat the last command (#85):

  This action, called #repeat, can be used to repeat the last entered command or send a command multiple times. It can easily be connected to a macro.

- When pressing a shortcut to a macro, go to the end of the window (#80).
- Fix several conflicts when opening the same world in different tabs (#90).

Build 43

- Add a documentation file to download and install CocoMUD.
- Add shortcut keys to add, edit or remove in all dialog boxes (#87).

Build 42

- Add the triggers with substitution (#79):

  Triggers can now handle substitution, that is replace one or more lines in the client.

- Update proper colors with the rich-text setting.

- Fix a bug with the cursor moving a bit randomly when several tabs were opened (#53).

Build 41

- Add default characters (#77):

  In the character dialog, one can now specify that a character should be loaded by default. When selecting a world in the connection list, the default character of this world (if any) is automatically selected. This saves time if you often login to the same character of a world, and doesn't prevent login on others.

- When creating a new character with special characters (like accents), CocoMUD doesn't crash at startup (#78).
- When CocoMUD loses connection, it will attempt to reconnect and enter the username / password if a character is set (#67):

  This fix is not perfect, it remains difficult to handle connection errors.

- When no update is available, the message is correctly displayed (#75).

Build 40

- Add a setting to disable rich text control (#82).

  You will find this setting in the menu bar, **File** -> **Preferences**, **Accessibility** tab. Disabling the rich text control can be useful for accessibility, although it removes colors from the client.

- Creating a new world and closing the dialog does not lead to an error (#69).
- Fix a bug when closing all tabs in the client.

Build 39

- Add a button to import a world right in the connection window (#60):

  It is now possible to import a world before connecting (which makes more sense in most cases). Simply click on the **Import** button in the connection window and select whether you want to import a world online or on disk.

- Display a dialog box to announce the world was correctly installed (#63).
- Installing a world with channels doesn't create "popup" windows (#81).
- Restarting CocoMUD after installing a new world isn't required anymore.

Build 38

- Add notepads for each world and characters (#62):

  CocoMUD now keeps track of separate files, where you can store any information, like exploring landmarks, quest reminders and so on. You have a specific notepad for each world, which can be opened through the menu bar -> **Game** -> **Notepad** -> **For this world...** menu. Simply type in your text, press Escape when you want to close it and save it. The same system holds true for character-specific notepads, which you can open in the menu bar, **Game** -> **Notepad** -> **For this character...** menu. This second notepad will not be accessible through other characters of this world.

- Add a documentation for macros.
- Add the mark triggers in the trigger dialog (#30).
- Allow triggers without action (most useful for mark triggers).
- Fix several visual errors in the interface.

Build 37

- Add the ability to configure characters in worlds (#61):

  This feature allows to create several characters per world. Characters can contain more specific configuration (like aliases, macros or triggers), but also login information. A character will store the information in an encrypted file, to login more quickly.

  To create a character, choose a world from the list in the connection area, then press tab and select "any" (the default choice). Press RETURN or click the **connect** button. You have opened a random character associated with this world. To now save it, go to the menu bar, **game** -> **Change this character's setting...**. In this dialog, enter a name for the newly-created character, a username (or a list of commands to be sent before the password), the password itself and then a list of commands to be sent after the password (if any). Click **OK** to save in an encrypted file. The next time you connect, you should see this character in the available list (select the world, then press Tab to find this character). CocoMUD will enter the commands you have provided, and will do so if you ask to be reconnected as well.

- Fix a bug when trying to create a new world (#66).
- Add the crash report dialog:

  This dialog appears when an error occurs during a given task. It will provide you with additional information about the bug, and will explain you how to report this bug to the team of developers.

Build 36

- Add the marked trigger (#30).

  Marked triggers can ask to move the cursor to a specific line. For instance, if one receives a message, the cursor can be moved right on this message. This can also help to explore, to move the cursor on the list of exits, for instance.

- Update the client's design and window.
- Add support for handling colors (#65).

Build 35

- Add the feature to import a world, from a file or online (#60).

  There is now a new menu, "import", in the menu bar -> **file** menu. In it are two options, one to import a world from a file, the other to import one online. The second option tries to find the worlds already configured on the project's website. One can download and install them directly.

Build 34

- Add the system of channels (#50):

Channels can keep track of a list of events. They are particularly useful to log communication channels on the MUD. Through the trigger system, users can feed a channel. Through a macro or alias, this channel can be displayed in a list, in a separate dialog box.

- In the preferences dialog, **accessibility** tab, users now can set whether the TTS is interrupted or not (#40).
- When changing the TTS options in the preferences, the options are taken into account immediately (#55).
- Fix some bugs in the SharpScript engine.

Build 33

- Multiline aliases/macros/triggers do not bug anymore (#63).
- One can now enter SharpScript in macros (#59).
- Variables are now described in the alias documentation.

- The title of the window with several open tabs is now accurately updated (#51).

Build 32

- Add a syntax to write variables in SharpScript:

  The syntax is $variable. Variables have been added to aliases (#45) and triggers (#44).

- Fix a minor bug in command stacking.
- Add mute triggers, which will be useful to support audio prompts.

Build 31

- Improve the debug logging system.
- Restructure the catalogs for translation (#41).

Build 30

- Add a logging system to debug events.

Build 29

- Attempt to fix a bug using the #play function while several worlds are opened (#48).

Build 28

- Add command stacking, to send multiple commands at once, using the semicolon or another character (#32).

Build 27

- Open multiple worlds in tabs (#42):

  It is now possible to open several worlds in tabs, or even a world several times in tabs. In the file menu are three new options, to create a new world, open a world in a different tab and close the current tab. One can navigate between tabs using Ctrl + tab or Ctrl + Shift + tab as usual.

  The feature to change the window's title when unread messages are received (#20) now takes into account the selected tab only.

- New menus to disconnect and reconnect from a world (#43):

  A new menu item in the menu bar, named connection, has been added. In it are options to disconnect from the current world, and reconnect to it.

- The client doesn't lag if connection to a distant server takes some time (#21).

Build 26

- When outside of the window, if messages are received, the window title changes to let the user know notifications are waiting on the client (#20).

Build 25

- Add the command history

  The command history remembers all commands you have entered. You can use it by pressing CTRL + up or down to go up or down into your list of commands. Alternatively, you can use command history in lock mode, by pressing Escape, then navigating in the history using the arrow keys. You can leave lock mode by pressing Escape again.

- One can now paste several lines to send multiple commands (#27).

Build 24

- Update the documentation of the basic features (#36).
- Remove the obsolete settings based on an input and output field (#35).
- When tabbing to tab-complete, the TTS speaks (and displays) the found result.

Build 23

- Add the tab-completion (#34):

  When the client receives messages from the server, it stores all words by frequency. When you begin typing a letter or more and then press tab, the client will try to finish the word you were typing. If you're not satisfied with this choice, you can press tab again and the client will display another result.

Channels in CocoMUD

CocoMUD offers an interesting feature of channels. Channels are basically lists of messages and you can put whatever you want in them. We often use CocoMUD channels to track game channels: when we receive a message on one of these channels (lie "public" or "ooc"), we can automatically capture this message and put it in a CocoMUD channel. A simple key press could allow to display this channel and read all the messages in it. This is pretty useful when you're exploring or figthing on the game, and don't have time to read all messages spoken by others, but you want to read them afterward.

We'll take this example for the rest of the documentation: an attempt to capture the context of a public channel. This example also relates to triggers, which is barely a coincidence.

Creating a new channel

The first step is to create a channel, an empty list that will contain the messages we want to put in it. Here, we'll create a channel named "public", for that's quite explicit. Remember that channels can be used to capture any information, however.

In the CocoMUD menu bar, open **game** -> **channels**. You find yourself in a dialog box showing current channels. It's probable you don't have any at that point. Go to the **add** button and click on it. You will be asked the name of the channel. Keep it short and explicit: here, we'll create a channel named public .

We can now send information to this channel using triggers. If you're not familiar with this feature, you might want to read the documentation about triggers first.

Feeding a channel with triggers

Okay, let's look at the two lines we may receive from the server. If we send a message on the public channel, it would be displayed like that:

```
[public] You say: my message to all
```

If it's someone else who sends to this channel, we would have:

```
[public] Somebody says: my message to all
```

What's common and different in our two lines?

- Well, both begin with "public" between brackets, followed by a space.
- Then there's the author. "you" or "somebody". So we don't know what it will be, we could use an asterisk.
- Then is the word "say" after another space. Wait! When it's me, it display "you say", when it's somebody, it displays "somebody says". What to do?
- After a colon and another space is our message.

To solve our issue between say/says, we have two options:

- Either we group both cases, writing "say*" (meaning say followed by nothing or anything).
- Or we could write two triggers.

Personally I would advise the latter, but I know some of you would prefer the former. So let's work with both.

One single trigger

What would our trigger look like in the end? Ready?

```
[public] * say*: *
```

Three asterisks! The first will contain the name of the one speaking. Either "You" or the name of the player speaking on this

channel. The second variable will contain either "s" or nothing. We shouldn't need it. The third variable contains the message itself.

So to create this trigger through the interface:

- Open the menu bar, **Game** -> **Triggers**.
- Click on **Add** to create a new trigger.
- Enter [public] * say*: * before pressing Tab.
- Select "Feed a channel with a message".
- Click on the **Add action** button.
- In the name of the channel to be fed, enter public (this is assuming the channel exists in CocoMUD).
- In the message to feed to the channel, enter:

```
$1: $3
```

- Click on **OK** several times to close the dialog and save the trigger.

  What was that $1 $3?

$1 contains the name of the one speaking. $3 contains the message. So when we receive the line:

```
[public] Jamie says: well done!
```

Our channel will be fed with the following line:

```
Jamie: well done!
```

You could have done the same thing with a single line of SharpScript:

```
#trigger {[public] * say*: *} {#feed public {$1: $3}}
```

That's a bit harder to understand, but if you're familiar with SharpScript, that's definitely quicker.

A last word regarding this trigger: you may have noticed that we don't play a sound when receiving this trigger. Nothing prevents you from adding another action to the trigger though. Similarly, to do it with SharpScript, you would enter:

```
#trigger {[public] * say*: *} {
    #feed public {$1: $3}
    #play sounds/public.wav
}
```

Two different triggers

As I have explained, to solve the problem of say/says, we could have created two different triggers. That makes for a longer configuration, a little bit, but that's easier to read, in my opinion. Sometimes it will not be possible to do otherwise, depending on the language the MUD is in.

Our two triggers would be:

```
[public] You say: *
[public] * says: *
```

I don't know about you, but I find it much easier to read. Our first trigger only fires when we send a message on this channel (perhaps we could say that we won't play a sound in this case) and the second one will fire when it's somebody else who's speaking. I'll give you the SharpScript instructions to create these triggers, but as usual, you could do the same through the interface:

```
#trigger {[public] You say: *} {#feed public {You: $1}}
#trigger {[public] * says: *} {
    #feed public {$1: $2}
    #play sounds/public.wav
}
```

What you'll choose is entirely up to you and the context, pick whatever feels more comfortable.

Bind a channel to a macro

Feeding a channel is certainly useful, but displaying it is even better. Most often, you will bind a macro to open a specific channel. It's pretty easy to do.

Start by creating a macro as usual. In the interface:

- Open the menu bar **Game** -> **Macro**.
- Click on **Add** to add a new macro.
- Press the key combination you want to associate with this macro. For instance, **CTRL + P**.
- Tab twice to find the list of possible actions. You should find in the list the "create or display a channel" function. Select it.
- Tab once to add the action. You will be asked the channel name. You will enter public here, in this case.
- Leave the dialogs by pressing **OK** multiple times to quit and save. Then you can press CTRL + P: the public channel should open. Magic!

Remove a channel

Removing a channel is done through the same interface:

- In the menu bar, select **Game** -> **Channels**.
- Tab until you find the **Remove** button. You will then be presented with a list of existing channels. Select one and tab again to remove it. If you press **OK**, the channel will be entirely removed and will not appear in your channels the next time you connect to this world.

Keep in mind, however, that triggers can still try to feed this channel. This will fail, since the channel doesn't exist, but you will have to remove the trigger manually, as it could do other things as well.

Characters in CocoMUD

CocoMUD offers to have several characters set on a world. Having a configured character has several advantages:

- Connecting to a character will enter your username and password automatically, and other commands if needed.
- Your username and password will be stored and encrypted, they will not appear in clear either in your command history or in one of CocoMUD's files.
- If you use a secure connection (like SSL or SSH), your password will be protected, from your computer to the MUD server itself.
- You can have separate notepads for different characters.

In short, when you select a world to connect to, you can choose to specifically connect to one of your configured characters. You can still, however, ask CocoMUD to connect to the world without any specific character.

Add a character to a world

By default you have no configured character in any of your world. To add a character, you should connect to a world. When you open CocoMUD and you find yourself in the list of configured worlds, select one and press RETURN.

CocoMUD will attempt to connect to the world you have selected. But it will not enter any information on login. To create a character, go to the menu bar, select **game** -> **Change this character's setting...**.

A dialog box will open, prompting you for several information:

- **Name**: the character's name. This is just for CocoMUD. The name that you enter will be shown on the connection screen (see below).
- **Username or other commands to send before the password**: this is the first command, or commands, that CocoMUD has to send when the connection is established to the game. Usually, this is a username. On some games, you might have other commands to enter. Don't enter the password here, just the commands to send before the password. If you have several commands to enter, put them on separate lines.
- **Password if any**: it's now time to enter the character's password. Not only is this a password field (into where you will see only asterisks), the information will be encrypted by CocoMUD and will not appear in clear in CocoMUD's configuration.
- **Commands, if any, to send after the password**: on most games, you'll have to just enter your username and password to connect. Other games will require additional commands however. You can enter them here. Like the commands to send before the password, you can separate them with a new line.
- **Set this character as the default choice**: if this box is checked, when you select the world on the connection screen, this character will be selected automatically. You can only have one default character per world.

Press **OK** to confirm the options. Then, you can close the tab and re-open it: in the connection screen, select the world. Then press the tab key: you should find yourself on another list, which is the list of characters associated with this world. The first item at the top of the list is "any character", which will connect you to the world but will not enter any command. If you have set a default character, however, the cursor will select this character, somewhere down the list. You can use the up and down arrow keys to choose another character and press RETURN to connect to it.

By default, when you move in the list of worlds (the list on which you are when CocoMUD opens), if you select a world, its default character is selected. You can press RETURN without pressing tab to select the character.

Editing the character's configuration

If you have changed the password to connect to the game, or any other information, you can just connect to the character, and then go back to **Game** -> **Change this character's setting**. The dialog box will open again, with the values you have set earlier. You can change them and press **OK**.

Adding several characters to a world

Adding several characters is quite similar to adding one: the only trick here is when connecting to the world. You have to select the world and tab to select "any character" (which means no character will be chosen). Then, you can go to **Game** -> **Change this character's setting** and just add the new character. When next time you open the connection screen, if you press tab on this world, you will find the new character in your list. Remember, you can only have one default character. Characters are stored in

alphabetical order, so you can easily navigate between the default character and alternatives. You can have as many characters per world as you like.

A word on configuration

For the time being, all configuration is set in a world. It means that all characters created on a world will share the same aliases, triggers, or channels.

Command stacking

Command stacking is a feature that allows users to send multiple commands at once. For instance:

```
say sounds;say good
```

This syntax is equivalent to writing:

```
say sounds
```

```
say good
```

Sending multiple commands at once can be useful in some situations. By default, the character for command stacking is the semicolon (;), but it can be changed in the preferences, input tab. If you wish to remove command stacking, simply remove the character in the setting.

You can double the command stacking character to actually send one. If you have set the semicolon (;) as a delimiter for command stacking (the default), you can type:

```
say one command;say another command with a wink emoticon ;;)
```

This command is equivalent to writing:

```
say one command
```

```
say another command with a wink emoticon ;)
```

Once the delimiter is present more than one time, it is not used as delimiter in the command stacking. Therefore:

Command	Equivalent
say ;;)	say ;)
say ;;;)	say ;;)
say ;;;;)	say ;;;)

Downloading and installing CocoMUD

CocoMUD is pre-built on Windows as a portable archive. It means that the only thing you'll have to do, if you are using Windows, is downloading the archive (see link below), extract it and open the **cocomud.exe** file. The details are provided in the following sections.

Downloading CocoMUD

Build	Windows
49	CocoMUD for Windows

If you are using another platform, you might wish to install CocoMUD from source. CocoMUD can run on Linux or Mac OS, although there's no pre-built version yet.

If you are accessing this page from a version of CocoMUD already installed, please note that the links given above may be out-of-date. You should refer to the online documentation to download the correct file.

Installing CocoMUD

Once you have downloaded the archive, you can extract it in a folder. CocoMUD doesn't require installation, it will run as a portable software (you can even put it on a USB flash drive and take it with you).

1. Extract the **CocoMUD.zip** file in your desired folder. Note that CocoMUD cannot run from the archive itself (if it does, it will not be able to write anything).

1. In the extracted archive, you should find a folder named **CocoMUD**. Open it.

1. In it is a file **cocomud.exe**. Click on it. CocoMUD should open and display the connection screen.

1. Should you like CocoMUD, don't hestitate to create a shortcut leading to **cocomud.exe** to make your life a bit easier.

To learn how to use CocoMUD, you might want to read the basics of CocoMUD. It will give you a step-by-step explanation and demonstration of some of the basic features. More advanced topics will be convered in the rest of the documentation.

Updating CocoMUD

CocoMUD has a built-in updater that opens when you launch CocoMUD. The updater will check for new versions of CocoMUD and, if found, will ask you if you wish to update. If you reply **yes**, CocoMUD will close and the updater will download and install CocoMUD, before starting the program again.

You can also see if there are updates by opening CocoMUD. In the menu bar, select **Help**, then **Check for updates**. The dialog might take a few seconds to load, since CocoMUD checks for updates on the website. If everything goes well, the updater should tell you that there is no update, or that there is one and ask you if you want to install it.

Several things can prevent the updater of running properly. Windows Firewall, Windows Defender, an anti-virus software or another software to protect your system might deny the updater the right to install CocoMUD.

If you run into problems while updating, CocoMUD might become corrupted, meaning it can't even start, or if it does, it displays aa lot of errors. If that happens, there are a few steps you can take to solve the issue:

1. Forcing update: in some cases, CocoMUD tries to update, but some folders are incorrectly changed. A software to protect your system might increase the risks of this situation to arise. With Windows firewall and Windows Defender, that situation doesn't occur very often. If you click on **cocomud.exe** after updating and nothing happens, or if CocoMUD opens but creates a lot of errors, try to force update. To do this, close CocoMUD (if it were open at all) and click on the file named **dbg_updater.exe**. **dbg** stands for **debug**, and it will run a debug update with minimal data. Even if it looks like you're up-to-date, **dbg_updater** will just download the lattest version of CocoMUD, keeping your configuration. This file will open a console with a progress bar, but you are not expected to do anything, just let it do its work. If everything goes well, CocoMUD should open in its new version after updating. If that still doesn't fix the problem, try to run it again before moving to step 2.

1. Manual update: sometimes, CocoMUD is so corrupted that **dbg_updater** can't run, or can't do much. A manual installation is then the next step. Simply download CocoMUD again (see the link above), and extract it in a folder. To transfer your old configuration in your fresh copy, just go in your old directory. Copy both the **settings** folder and the **worlds** folder. These

contains your configuration. Go into your fresh copy, remove these folders and replace them with the ones you had in your old configuration. Then launch CocoMUD.

If that doesn't fix the problem, you can definitely contact the team of developers by creating a bug report .

We wish you the best of gaming experiences with CocoMUD!

CocoMUD main Features

Here are CocoMUD client's main features.

- Macros

A new line.

Thanks for editing this document!

Macros

CocoMUD supports a system of macros as most MUD clients do, allowing to link a shortcut with an action (as a command to be sent to the server). This document explains the creation of simple and more complex macros.

Basic feature

In its most simple version, a macro is a link between a keyboard shortcut and a command: for instance, if you press Ctrl + F1, CocoMUD may send the "look" command to the server. In this documentation, we will see how to create a macro to link the F1 key with the "north" command, as some MUD clients are already configured to doing.

Adding a macro

There are two ways to add a macro, and as the interface is the easiest alternative, we will see it first.

Through the interface

In the menu bar, select **Game** -> **Macro**. A dialog box should open with your list of current macros. More likely than not, this list will be empty at first. Click on the **Add** button.

Another dialog opens. The cursor should be on a text entry, but you are not expected to type in it, just press the key combination. For instance, in our example, press the F1 key.

When you do, "F1" should appear in the text entry. You can try to press a lot of different keys. For instance, press the Ctrl key, hold it down and press the I key, then release both keys. You should see in the text area:

```
Ctrl + I
```

You can, in theory, use Shift, Ctrl or Alt in your macros. It's not recommended to use Alt, however, since it can be linked to menu shortcuts (like Alt + F to open the **file** menu). There's not many limitations on what to use for macros. For instance:

```
Shift + Backspace
Ctrl + F12
Ctrl + Shift + O
Ctrl + PavNum8
```

If you wish to connect a macro to a key from the numpad, you should switch it on before pressing the **Add** button, or at least, move in another area before doing so. Otherwise, CocoMUD will associate a macro with the Numpad Lock key, which isn't often desirable.

When you have pressed the desired shortcut to associate with this macro, press Tab to go to the next area. It's also a text area, this time for the command to be sent to the server.

So, following our example, you pressed the **Add** button, then F1, then Tab, you can now type "north".

If you tab again, you find a list of actions. We will study it later, it's only necessary for more complex macros, so you can press tab again until you find the **OK** button. Click on it to add the macro. You should see it in the list of macros.

If you want to change the command to send to the server for a specific macro, you need to edit it (selecting it in the list of macros, and clicking the **Edit** button). However, if you wish to change the shortcut associated with this macro, you can just select it, then press Tab. The cursor will be in another text area, or rather, in a shortcut area, when you can press a new key to associate with this macro. Simply do so, then navigate to **OK** to save the modifications. If you close this dialog box without pressing **OK**, the modifications you have done (including the macros you have added) will be lost.

Back in the client's main window, press the F1 key. The command is sent (silently) to the server and, if you are connected to a server that understands this command, you should receive an answer. In my case, I get:

> You cannot go in that direction.

Through SharpScript

You can also add a macro through SharpScript. This is the way your macros added through the interface are stored, incidentally. A file, named "config.set", is created in your world settings, containing your macros, aliases and triggers, as well as additional configuration.

To create a macro, you should use the #macro action. It takes two arguments:

- The shortcut to associate with this macro.
- The command to send to the server.

In our example, we could have created our macro using the following SharpScript instruction:

```
#macro F1 north
```

You can actually paste this instruction right in your client's main window and press RETURN, just as if to send a command. CocoMUD sees that the command beings with a single sharp symbol and sends it to the SharpScript interpreter. If you add a macro in this way (assuming you made no mistake), it will be visible instantly in the **Game** -> **Macro** menu.

If one of the parameter is too long, don't forget to place it between braces:

```
#macro {Ctrl + Shift + O} {say that's a rather long line of text.}
```

Braces are necessary when your parameter contains spaces. You can put them even if your parameter doesn't contain spaces, the SharpScript engine will not mind.

Editing a macro

We saw how to edit a macro in the previous section. There's nothing really complicated about it, actually. Just remember that, if you only with to change a shortcut associated with a macro, you can simply select it in the list of macros (in the **Game** -> **Macro** menu) and press Tab, then press your new combination of keys to associate with this macro. If you wish to change the text to send to the server, however, you should select the macro and press the **Edit** button. The dialog is the same as described above to add a new macro.

Removing a macro

You can easily remove a macro. In the interface (menu bar -> **Game** -> **Macro**), select the macro you wish to remove and click the **Remove** button. You will be asked for confirmation. Don't forget to exit the dialog box by clicking on **OK**, otherwise the macro won't be removed.

More complex macros

Macros often connect a keyboard shortcut to a command being sent to the server. But they can actually connect a keyboard shortcut with more complex actions, like playing a sound, displaying a message or a variable, displaying a channel or more. The interest is not always obvious at first glance. The example of more complex macros can be found in some configurations, when the world provides audio prompting, for instance.

The general idea of audio prompting is that the line of prompt (where you can find health points and movement points and that appears at each command, or almost) is hidden. It is captured by a trigger. The information is stored in variables. That's beyond the scope of this document to explain how, but the important point is, you can configure a macro to display this variable. For instance, if you press Ctrl + H, you should get the health points (captured by the trigger).

We have used the second text area to send commands to the server. The truth is, you can also type SharpScript instructions in this area. But as writing SharpScript can be a bit technical, you can also use the SharpScript editor. It is not as complicated as it sounds: when you add or edit a macro, simply leave the second text area blank. Tab to the list of actions and select one. For instance, select "Display a message and send it to the screen reader". Click on the **Add action** button. You will be prompted for additional information (in this case, what message to display and how to display it). When you're done, click **OK**. In the second text

area will be your SharpScript instruction. You don't need it, it's just a landmark. Click on the **OK** button several times to exit the dialog by saving your modifications.

Triggers in CocoMUD

Triggers may very well be the most powerful feature of any MUD client. CocoMUD offers an advanced system of triggers, which will be discussed and presented with examples in this documentation.

What is a trigger?

Usually, when you enter a command, the server to which you are connected "replies" with one or more lines. This reply, just like the command you sent, is mere text (sometimes with little additions, like colors).

Triggers can intercept a specific line of text and react to it to do something. That's the basic definition of any trigger: I react to something (one or more lines sent by the server) and I do something in response (playing a sound, changing the text, sending another command, saving some parts of the line, moving the cursor and so on).

Since triggers are so powerful, their setting has been spread on several layers. You don't need to understand, or even know, the most advanced type of trigger to use this feature. This documentation will walk you through the basic setting of triggers, until the setting of most advanced ones.

Creating a trigger

Let's begin with something simple: a trigger that plays a sound when someone (anybody) speaks in a channel.

This will obviously differ on every MUD, so don't hesitate to adapt this example. For this example, we'd say we can use the "chat" command to send a message to everyone logged to the MUD. The command we will type should be like this:

```
chat hello everyone!
```

And if the MUD behaves, it should reply with:

```
[public] You say: hello everyone!
```

Similarly, if someone else speaks on this channel, you might see something like:

```
[public] Aaron says: hi there
```

In short, if we want to intercept the messages on this channel, we need to watch for lines that begin by "[public]" (that is the word "public" between brackets).

> Why should we intercept these messages, anyway?

If you're new to triggers, you might wonder why it's useful. The answer in that specific case could be that, if we receive a message on the public channel, the client could play a sound to alert us. That's on this task we're going to work.

So we're going to see how to create a trigger that:

- Reacts when it receives a line beginning with "[public]".
- Plays a sound when it does.

Like most features on CocoMUD, triggers can be created either through the interface or through SharpScript. We'll see the former first:

Through the interface

To add, edit or remove triggers, select **Game** in the menu bar, then **Triggers**.

You will find yourself in the list of current triggers configured for this world. This list might very well be empty at first. To create a trigger, click on the **Add** button.

A new dialog box opens. The cursor should be in an edit field, where you're asked to type your trigger. That's the reaction part, the place where you will specify what fires your triggers.

In our example, we've determined that our trigger should be executed whenever a line "beginning by [public] is detected. You can type [public] in this field, but don't leave it just yet: if we create a trigger [public], it will only fire when the server sends a line containing **only** [public]. That's not the case for us, we want for our trigger to fire when the line begins with, not only contains [public]. To do that, we'll have to add an asterisk (*) after [public]. If we type [public]* in our trigger, CocoMUD will know that we are looking for lines that begin with [public] followed by something (no matter what).

You might recognize this syntax, it's the same we use to create aliases with variables. And that's not a coincidence, we'll see later why.

For the time being, you can write in this edit field:

```
[public]*
```

Press Tab: you find yourself on a list of possible actions to be performed when this trigger fires. In our example, we want our trigger [public]* to play a sound. So browse this list until you find:

```
Play an audio file
```

Then press Tab again and click on the **Add action** button.

You are now asked to configure this action. What audio file do you want to play when this trigger fires? You have a **Browse** button if you want to select the file in your file system. And once selected, you have a **Test** button, to check that CocoMUD can play this audio file just right (CocoMUD supports .wav and .ogg files). If everything works, just press **OK**. The action will be added. The cursor is moved into a list where you see only one item:

```
#play whatever.wav
```

Of course, the "whatever.wav" will be replaced by your file path and name.

This is the sharp editor: it is used by every feature (aliases or macros for instance) that needs to execute complex actions at some point. The window can be a little intimidating at first, but as you've seen, adding an action isn't that complicated. So here is a brief summary of the trigger dialog with the sharp editor:

- The first thing to set is the trigger itself (the part that will fire the trigger, [public]* in our case).
- Below is a list of actions currently associated with this trigger. It's a list of actions (what to do when this trigger fires)? You can add more than just one action. Note that this list is disabled if it's empty.
- An **Edit** button, to edit the selected action. For instance, in our case, if you've changed your mind and want to play a different audio file, you need to select this line and click on **Edit** to change the audio file.
- A **Remove** button, to remove this line of action.
- Still below is another list of actions. This time, it's a list of actions you could desire to add. That's the list we browsed through when trying to play a sound if this trigger fires. We won't describe all of them, but you'll see other examples in this documentation.
- Then is a button to **Add** the action you have selected. Remember that you can connect a trigger with zero, one, two or more actions, there's no real limit.
- There are other checkboxes and options in this dialog that we will discuss later.

You can now press on the **OK** button, since we did what we wanted: our [public]* trigger is connected to the action to play a sound.

You will find yourself in the list of triggers again, with your newly created trigger. You can press **OK** again to close this dialog and save the trigger to your configuration. If you don't press **OK**, the trigger will not be saved.

To see if it works, let's try to fire this trigger! You could send to the server something like:

```
chat does it work?
```

If everything goes well (assuming, one more time, that the server answers the way we have planned), it should send you something like:

```
[public] You say: does it work?
```

And you should hear a sound, the sound you have selected. There! It wasn't that complex, was it?

Through SharpScript

As usual, the interface allows to manipulate potentially complex settings, but they all are converted to SharpScript in the end. Setting the trigger we have worked on above in SharpScript is quite simple. You can enter the following command in your MUD client, or write it in your "config.set" file:

```
#trigger [public]* {#play path/to/sound.wav}
```

It's a SharpScript instruction. From left to right:

- #trigger is the name of the action. Here, #trigger just creates a new trigger.
- [public]* is what should fire our trigger. Just like above, we have specified [public]* , which means [public] followed by anything.
- Then we specify the action or actions to be executed when this trigger fires. Here, we have #play path/to/sound.wav. #play is another SharpScript action that just plays the sound you give it in argument, if it can. Notice that we have surrounded our call between braces, because there is a space between the name of the function (#play) and the argument of the function (path/to/sound.wav).

Adding a trigger using the SharpScript syntax may be quicker, but it will not forgive easily if you make an error of syntax. That's one of the reasons why using the interface might be safer.

Editing a trigger

In the dialog box of triggers (menu bar, **Game** -> **Triggers**), you can edit a trigger if needed. You will need to do so if you want to change the part that fires the trigger, or the actions associated with this trigger (for instance, the audio file to be played, in our example).

Removing a trigger

Removing the trigger can also be done through the interface. Just press the **Remove** button and confirm that you do want to remove this trigger. Do not forget to click on **OK** to close the list of triggers. If you don't, the trigger will not be removed.

Using variables in triggers

It is time to look more closely at our previous trigger, in particular at the asterisk sign (*). It is not a coincidence that you may recognize this syntax from the documentation on aliases, for it is exactly the same principle: * means anything.

So here is a list of possible syntax:

Syntax	Meaning
Welcome*	Any line beginning by Welcome .
*dude	Any line that ends with dude .
spoil	Any line containing spoil, at the beginning, the end or the middle. It will also fire if the line contains spoiler for instance.

* spoil *	Any line that contains the word spoil surrounded by spaces. The word spoiler would not fire the trigger in this context.
You earned * credits in *.	Lines like You earned 80 credits in combat. or You earned 10 credits in management. will fire this trigger. Also note that the line You earned some unknown credits in something. will also fire.

In short, an asterisk sign (*) means anything, including a number, a letter, a word, a space, or whatever else... including nothing.

A little word of caution: the syntax of your trigger is really important, and you should check when the trigger would be fired.

```
*public*
```

This trigger would fire when there is the word public everywhere in the line. It means this trigger would fire when you receive the following line:

```
[ooc] Modo says: please talk on the public channel, not here, if you're stuck with a quest.
Yassen tells you: where should I write, if I want my message to be publicly readable?
To send your book to publication, press S.
```

Remember that a trigger can be easily restricted... but can easily be fired. That all depends on you.

Back to variables. The asterisk does two things:

- It determines when the trigger should fire (specifying "anything").
- It writes in variables.

Let's use the same example of our trigger [public]*. What happens when, say, you receive a line like:

```
[public] Edgar says: I don't get it at all, could someone help me?
```

First, the trigger [public]* fires, then the part after [public] is sent to a variable. A variable can store information, and that's just what it does here. Variables are numbered starting with $1, $2, $3 and so on. So in our example, when we receive the above message, the variable $1 is created containing the text:

```
 Edgar says: I don't get it at all, could someone help me?
```

What can we do with it? About anything. Every parameter in our actions could use variables. We will see a concrete example below. For the time being you could display it:

```
#say $1
```

Which should display:

```
 Edgar says: I don't get it at all, could someone help me?
```

Why does it begin with a space?

If you wonder about it, just put the trigger with the line side-by-side, that might help:

- Trigger: [public]*
- Line: [public] Edgar says: I don't get it at all, could someone help me?

Can you see it? Our line puts "public" between brackets, then a space, then the name of the one speaking... while our trigger just says "public" between brackets and anything after that. Which includes our space.

The solution? To slightly change our trigger:

```
[public] *
```

This time, we put a space between the right bracket and the asterisk sign. So now if we receive the line:

```
[public] Edgar says: I don't get it at all, could someone help me?
```

And we display $1, we'll see:

```
Edgar says: I don't get it at all, could someone help me?
```

Spaces may be the main source of confusion in your triggers. The best advice is to look closely at the lines you receive from the server, and use only the * sign when you don't know what will be put there.

CocoMUD channels in triggers

CocoMUD has a more interesting feature of channels. A channel is basically a list of events. It will store these events and you can display this list whenever you want. It might be useful, for instance, when exploring the game and not paying particular attention to some messages (on the public channel, or the ooc channel, for instance). Afterward you might want to read them, so you display the list of messages.

You will find more informatioon in the documentation about channels.

Mute triggers

In some cases, when you receive a line, you don't want for it to appear.

> Does it really happen?

In some contexts. For instance, some MUDs send regular ambiance messages to the client. That can be great to set the ambiance, but that's usually not very user-friendly for people using screen readers.

```
A cloud of sparks from your campfire soars toward the darkening sky.
```

Nice and, well, yeah, when you're near a campfire, it cracks and smokes and sends sparks, but with a screen reader, that's not really useful. So we could just remove this line.

To do so, create a trigger. Through the interface:

- Go to the menu bar, **Game** -> **Trigger**.
- Add a trigger.
- Paste the line: A cloud of sparks from your campfire soars toward the darkening sky.
- No need to select an action, we won't do anything (except if you want to add a crackling sound when that happens).
- Tab until you find the "mute trigger" checkbox. Check it.
- Press **OK** several times to validate and save.

If the client receives this line, it will just ignore it and not display it.

You could have done the same thing in SharpScript:

```
#trigger {A cloud of sparks from your campfire soars toward the darkening sky.} {} +mute
```

Notice that the second parameter is empty (just {}, meaning we don't perform any action). The third parameter is a flag, beginning by + or - and followed by the name of the flag. Here, +mute means we activate the mute flag to set our trigger as a mute trigger.

As pointed out, you can have a mute trigger that performs actions, like playing a sound, displaying something, sending a command, putting the line (or part of the line) in a channel, and so on.

Mark triggers

Mark triggers can be useful for accessibility. They will put the cursor directly on the line that fires the trigger. It is useful in some contexts: for instance, when you're exploring, you want the cursor to be put on the list of exits, rather than at the bottom of the window where you will have to press the up arrow key several times.

It's the same principle to create this trigger:

- Open the menu bar, **Game** -> **Triggers**.
- Click on **Add** to add a trigger.
- Put the part that should fire the trigger. In our example, perhaps something like: Obvious exits: * .
- Tab to check the "mark trigger" checkbox.

The next time you will receive the line beginning with "Obvious exits", the cursor will be moved directly on it.

The same thing in SharpScript would be:

```
#trigger {Obvious exits: *} {} +mark
```

Triggers with substitution

In some cases, when you receive a trigger, you want to modify the line that fired the trigger. One of the common case is to shorten the message. Some MUDs have very long messages for some channels, like this:

```
Somebody publicly speaks on the 'ooc' channel in a worried voice: is it safe?
```

While this is great, it would be nice to shorten it and perhaps write it differently. Like:

```
[ooc] Somebody: is it safe? (with a worried voice)
```

The way to do that is to create a trigger, and add a line of substitution. The line of substitution will replace the line that fired the trigger in the client.

- Open the menu bar, **Game** -> **Triggers**.
- Click on **Add** to add a new trigger.

- Write the part to fire this line. Here we might have something like:

```
* publicly speaks on the '*' channel *: *
```

- Notice that we have $1 containing the name of the speaker, $2 containing the name of the channel, $3 containing the tone of the voice and $4 containing the message.
- Tab until you find the empty text field "Message to substitute to the trigger line, if any".
- In it put:

```
[$2] $1: $4 ($3)
```

Does it look understandable? Take the time to read the trigger, the line that should fire it and the substitution.

The same trigger in SharpScript would be:

```
#trigger {* publicly speaks on the '*' channel *: *} {} {[$2] $1: $4 ($3)}
```

Important note: we have three arguments to the #trigger action here. The first one is still the name of the trigger, the second one is still the action to be performed (empty in our case). The third one is our substitution. If the field is empty, there's no substitution. That's just what happened for all of our previous tests.

Triggers fired by a regular expression

This section is more advanced, beware.

The asterisk symbol is great, but it doesn't offer much flexibility. Hopefully, we can connect a trigger with a regular expression . Regular expressions won't be described in this documentation, that's clearly off topic, but there are plenty of resources out there to learn it.

As far as CocoMUD is concerned, there's not much to know: if the trigger begins with a ^ symbol, it's a regular expression and CocoMUD will treat it as is.

```
^You receive \d+ XP.$
```

This will trigger when you receive the line "You receive ... XP.", with "..." being one ore more numbers. This won't fire the trigger if you receive the following message: "You receive a lot of XP."

An important thing to note when using regular expressions in triggers, however, is that, if you want to capture parts of the line, you have to use groups:

```
^You receive (\d+) XP.$
```

Then the number of XP will be put in $1. You can also use named groups and call them with $myname afterward.

Advanced triggers in Python

The SharpScript engine is really great... but it doesn't allow everything. And its aim is to remain light and not allow everything. It's not a programming language. But Python is. And CocoMUD is developed in Python. The SharpScript engine is designed to send code to Python when the user asks it to do so. That means you can have triggers that execute much more complex actions. At the same time, the Python code can access all functions defined by the SharpScript engine, which keeps the code simple and potentially very powerful.

We'll take an example as usual, but keep in mind the possibilities are really endless.

The game sends a line when earning XP, but it also displays the total number of XP needed to level up.

For instance:

```
You receive 37 XP and need 500 to level up.
```

Let's say we want to extract these two numbers and display a percentage. xp / total * 100. That's not going to work with pure SharpScript.

For the time being, it's not possible to use the interface to manipulate Python code. So we'll need to do it in the "config.set" file directly.

```
#trigger {You receive * XP and need * to level up.} {+
    # $1 contains the number of XP
    # $2 contains the total number to level up
    xp = args["1"]
    total = args["2"]

    # We're going to try and convert these numbers
    try:
        xp = int(xp)
        total = int(total)
    except ValueError:
        # The numbers can't be converted, but do nothing
        pass
    else:
        percentage = xp * 100.0 / total
        say("You receive {}/{} XP ({}%).".format(xp, total, int(percentage)))
}
```

Wow! That was some trigger! Let's quickly review it:

- The line that should fire the trigger shouldn't be much of a surprise by now.
- The second parameter begins by a {+ (a left brace followed by a plus sign). This tells CocoMUD that what follows is Python code.
- Note that all the code is indented. This is not just for readability this time, this is a Python requirement. You can use a single space or a tabulation to indent, that's your choice, I usually use 4 spaces because it's easier to read, but that's only a convention.
- Some comments. Don't underestimate their positive impact.
- We extract the two numbers (XP and total). To access them, we use args which is a dictionary containing all variables. Remember that the first variable is $1, we need to access it through args["1"].
- We need to convert these numbers. Why? There are still strings at that moment, it's as if they had been sent by the user. And CocoMUD doesn't do anything about it, so you need to do it manually. That's why we convert these numbers in a try/except/else statement. Notice that if the conversion fails, we do nothing.
- Next we create the percentage. It's Python 2, so we need to explicitly state that it should take into account floating points.
- And finally we use the say() function. That's just like using the #say function in SharpScript, it's the same thing (not the same syntax because we're in Python). All SharpScript functions can be accessed like that, so you could use play() or send() or feed() and more.

I'm not really happy about my previous trigger... I'm not against some lines of code, especially since I find them much more readable than when you try to do everything in script, but... we could definitely work on something safer with the help of regular expressions.

```
#trigger {^You receive (\d+) XP and need (\d+) to level up.$} {+
    # $1 contains the number of XP
    # $2 contains the total number to level up
    xp = int(args["1"])
    total = int(args["2"])
    percentage = xp * 100.0 / total
    say("You receive {}/{} XP ({}%).".format(xp, total, int(percentage)))
}
```

Okay, using regular expressions in your trigger is a bit more technical, but the gain for our code isn't to be dismissed.

If you want more help about using Python code in SharpScript, just refer to the section describing SharpScript.

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.

CocoMUD client

CocoMUD client is a MUD client designed for ease-of-access with a screen reader. It plans to offer just as many features as another MUD client, while remaining as accessible as possible through several features, including a Text-To-Speech (TTS) for speech and Braille messaging.

Home Download Quick start New issue

Other languages: French.

Start using now

To download CocoMUD, go to the download page. You might also like to read CocoMUD basics to learn how to use the main features of CocoMUD.

Main features

These features are the main tasks of the CocoMUD client roadmap. You can click on one of these issues to see its progress as the project evolves.

- A MUD client with a steady networking system (#5).
- An accessible client with most screen readers (#6).
- A portable client on Windows, Linux and Mac OS (#7).
- A translated interface into different languages (#8).
- A simple but powerful mechanism to customize the user experience (#9), with aliases, macros, triggers and more.

Contact and discussions

You have a bug to report, a feature to suggest? You just want to say hello or talk about some ideas you have?

- You can post on CocoMUD forums, assuming you have an account here (if not, click here to register) .
- You can open a new issue, assuming you have an account here (if not, click here to register) .
- You can subscribe an post to the mailing list of CocoMUD. You don't need an account to do so, but you will need to subscribe by entering your email address.

Licensing

CocoMUD is distributed under the 3-clause BSD. Its source code can be found on its Github repository . Note that the project itself (discussions, issues, forums, documentation) is hosted on PlanIO.