This document describes how you can do the following things:

* perform some action in your map script when an rcon "command" is issued
* in a single-player map, have a keypress cause some action in your map script
* have a cvar which controls some setting in your map (perhaps whether a certain weapon is available), which has an immediate effect on the setting

All of these things are variations on the one theme - "polling" (i.e. continually checking) a cvar and comparing it against some other value to see when it has changed. In the case of performing an action in a map script when an rcon "command" is issued, you will start with the cvar being empty, and then wait until it becomes non-empty, indicating that somebody has issued a command; you can find out what action to perform by looking at the value of the cvar, then reset it back to being empty so another command can be issued - this is how [MoH?](http://gronnevik.se/rjukan/index.php?n=Main.MoH?action=edit) Admin MOD (MAM) works. In the case of having a keypress cause an action in the map, it's the same situation, except instead of using rcon to set the variable, you set it locally, and bind a key to set the variable (or to an alias which sets it). In the case of controlling a setting in the map, rather than always having the cvar empty, you have it set to the current setting, and you keep a record of what the current setting is in another variable, and when your record of the current setting and the actual value of the cvar differ, it means the cvar has been changed and you should take the appropriate actions to change the value of the setting (e.g. disabling something).

**Perform some action in your map script whenever "command <command name>" is issued**

The following code monitors the cvar "command" (this is the name of the cvar used by MAM, so it's best not to use this cvar name yourself) for changes. When it sees a change, it looks at the value, calls a thread based on the value, and then resets the cvar back to being empty:

command\_thread:

while (1) { // this thread should keep running forever

//

// First, reset command to being empty; we could do this at the

// end of the loop because it will start out empty (unless the

// server config has "command" set to something in it!), but do

// it here for clarity.

//

setcvar "command" ""

//

// Now wait for someone to execute a command by using

// "command <command name>" on the local console or

// "rcon <password> command <command name>" remotely (they

// could optionally use "set command" or

// "rcon <password> set command").

//

local.command = getcvar "command"

while (local.command == "") {

//

// Wait for one second; we could wait for less time

// to reduce the amount of time between the command

// being entered and the action being taken, but it

// would increase CPU load.

//

wait 1

local.command = getcvar "command"

}

//

// The loop above terminated because "command" was changed.

// Determine what it is set to and take an appropriate

// action.

//

switch (local.command) {

case "foo":

thread foo\_command

break

case "bar":

thread bar\_command

break

default:

//

// Some other command was entered, so display an error

// message on the server console.

//

println "Error: invalid command " local.command

}

}

end

**NOTE: this code, and all of the code in this document, has not been tested**

The code above will call the thread foo\_command when "command foo" or "rcon <password> command foo" is entered, and similarly for "bar". If you wish to have a command that takes arguments, e.g. "command foo 5", where "5" has some significance to the thread foo\_command, you will need to write a scanner which tokenizes the variable, and scanners and tokenization is a subject that is too long to cover here. MAM provides support for doing this, so if you need arguments it would be best to take advantage of MAM's support for them in commands by making your code into a MAM add-on. MAM's interface for writing add-ons appears to be well documented.

**Perform one specific action once only**

Of course, if you only have one action you wish to perform, you don't need a 'case' statement to check the value of the variable. You could just have a cvar called, say, "blow\_up\_the\_bridge", which, when set to 1, causes something to happen, e.g.:

blow\_up\_the\_bridge\_thread:

setcvar "blow\_up\_the\_bridge" ""

local.command = getcvar "blow\_up\_the\_bridge"

while (local.command == "") {

wait 1

local.command = getcvar "blow\_up\_the\_bridge"

}

$bridge thread blow\_up

end

In this case, you trigger the blow\_up thread being called for the $bridge entity by setting "blow\_up\_the\_bridge" to any non-blank value, e.g. via "blow\_up\_the\_bridge 1", "set blow\_up\_the\_bridge NOW!", "rcon <password> blow\_up\_the\_bridge X", etc. Since there isn't a "while (1) { ... }" loop, you can only do this once per round (for Objective, Tug-of-War, etc.) or map (for TDM, FFA, etc.).

**Have a keypress cause some action**

If you'd like to have a keypress cause an action, you can simply bind the action of setting a cvar to a key. This of course only works for single-player maps, since for multi-player maps, the map script is polling the cvar on the server, while the bound key will set the cvar on the client side. Technically, it would be possible to have clients issue the 'setu' command, which sets a variable which can be seen on the server via the 'dumpuser' command, and then have a program running on the server which issues 'rcon' commands to the server to find the values of the variables and pass them onto the map script via 'rcon' commands, but that would be quite complex. Contact me if you wish to implement a system like this and you're capable of programming in a language other than MOH scripts and I can provide some additional information.

**Have a cvar which controls a setting where changes are reflected immediately**

You might, for example, have code in your map script which causes bombers to fly over the map and bomb a certain position on the map. You may wish to have a cvar to control this feature since some players might not like it. You could easily use a cvar to prevent the thread from starting when the map starts, e.g.:

bomber\_thread:

local.disabled = getcvar "bombers\_disabled"

if (local.disabled) {

end // exit the thread so we don't have bombers

}

while (1) { // run forever

wait 30 // 30 seconds between bombing runs

thread do\_bombing\_run

}

end

However, if the administrator sets the variable to disable the bombers (or if there is a voting option to control this, and players vote to disable them), the bombers will keep making bombing runs until the next round starts in a round-based mode, or until the next map in a TDM/FFA-type mode. You can make the change happen immediately by testing the cvar inside the loop:

bomber\_thread:

while (1) { // run forever

wait 30 // 30 seconds between bombing runs

local.disabled = getcvar "bombers\_disabled"

if (!local.disabled) {

// we only do a run if the bombers are enabled

thread do\_bombing\_run

}

}

end

In this case, changing the value of the variable to 1 (or anything non-blank) won't have an effect if there is a bombing run in progress (since the 'do\_bombing\_run' thread will already be started), but it will prevent further bombing runs from occuring.

Another type of thing you may wish to control is the presence of some object. For example, you may have a usable Flak88 gun that is spawned into the map. As for the bombers, you could easily check a cvar at the start of the script and not spawn the Flak88 if, say, 'flak88\_disabled' is set. However, having the change take effect immediately is more difficult, since you have to spawn the Flak88 whenever 'flak88\_disabled' is **changed** to be blank, and you have to remove it whenever the variable is changed to be set. Here is one method of doing that:

flak88\_thread:

while (1) { // run forever

//

// Wait for the Flak88 to be enabled

//

local.disabled = getcvar "flak88\_disabled"

while (local.disabled) {

wait 1

local.disabled = getcvar "flak88\_disabled"

}

//

// The Flak88 has been enabled, so spawn it and wait for it

// to be disabled.

//

spawn ... targetname $flak88 // full code for spawning Flak88 omitted

local.disabled = getcvar "flak88\_disabled"

while (**!**local.disabled) {

wait 1

local.disabled = getcvar "flak88\_disabled"

}

//

// The Flak88 has been disabled again, so remove it and

// then return to the top of the loop and wait for it

// to be re-enabled.

//

$flak88 remove

}

end

**NOTE:** I don't know if removing a turret weapon while there is a player using the turret can cause problems. You should test this before using the code n a production server.

This code should have the same effect as the above, but it's written in a different way - instead of having two "wait" loops, one which we sit in while the Flak88 is "disabled" and hence not spawned (the first one) and one which we sit in while the Flak88 is "enabled" and hence spawned (the second one), we have a variable which has two possible values - the Flak88 is "spawned" (1) or "not spawned" (0):

flak88\_thread:

//

// Initially the Flak88 is disabled, i.e. not yet spawned

//

local.spawned = 0

while (1) { // run forever

local.disabled = getcvar "flak88\_disabled"

if (!local.disabled && !local.spawned) {

//

// The cvar says it's NOT disabled, but currently it

// is NOT spawned, so we should spawn it to make it

// "enabled".

//

spawn ... targetname $flak88 // full code for spawning Flak88 omitted

local.spawned = 1

}

if (local.disabled && local.spawned) {

//

// The cvar says it IS disabled, but currently it

// IS spawned, so we should remove it to make it

// "disabled".

//

$flak88 remove

local.spawned = 0

}

wait 1

}

end

Here is a bit of programming theory for you: in the first code snippet, the state of the Flak88 (spawned vs. not spawned) is **implicit** in that our location in the code (which 'wait' loop we're sitting in) encodes it. In the second code snippet, the state of the Flak88 is **explicit** in that we keep it in the local.spawned variable. I think it's probably easier to have the state implicit than explicit - it requires less thinking :)

-- [dcoshea](http://gronnevik.se/rjukan/index.php?n=Profiles.Dcoshea)