ARMA3 HEADLESS CLIENT GUIDE

for mission creators & server admins

by: **Monsoon**

rweezera@hotmail.com

February 10, 2015

Contents

1 Introduction

As most of you already know, Arma is a very CPU (central processing unit) intensive game. There are a number of things going on during a session that requires the CPU's attention. Unfortunately, the CPU cannot do more than a handful of calculations at any point in time, resulting in a potential bottleneck for massive games like Arma.

Missions that have a substantial amount of AI (artificial intelligence), objects, or scripts can result in degraded server performance. When the server runs poorly the server must prioritize its tasks, which means that sacrifices will be made. One of the first things to go is the AI. They often become 'potatoes' in the sense that their reaction times are severely diminished. This ultimately leads to a poor experience for the player as the challenge evaporates. In addition to the poorly performing AI, user scripts and mod performance begin to break down and cause tremendous amounts of lag as the server tries to play catch up. The headless client is something that Bohemia Interactive introduced a number of years back to potentially mitigate this problem.

In this guide I attempt to outline the general concepts and ideas surrounding the headless client, or HC. My goal is to provide a base knowledge needed to build support for a HC within your very own Arma3 missions.

1.1 What is a headless client?

When an Arma mission is launched, there are two types of entities involved: the *server*, and the *client*. A *client* in this context is defined as any computer that is *not* the server. When you and your friends connect to an Arma mission, your computers all become *clients*.

The term 'headless' simply refers to a lack of an interface. Servers without monitors for instance, are considered to be headless. A headless client therefore is just a computer that connects to a server (*client*), but lacks any sort of interface and is therefore *headless*. The next natural question is: what purpose could this serve for an Arma mission?

As I mentioned above, the HC is a way to mitigate the problem of poor server performance when under heavy load. The general idea is to offload the AI from the server to a client. That client can then handle the AI calculations without the additional burden of network synchronization etc. This typically results in much more responsive, more human like AI, along with less server lag. The improved responsiveness was well demonstrated by Dslyecxi[1].

1.2 When should you use a headless client?

The easiest way to tell if your mission would benefit from a headless client is to monitor server performance during a play through. While logged in as admin to your server, open chat and execute the command:

```
#monitor 10
```

[1] http://youtu.be/4RUkh5GO1Cg

This will begin spouting server performance information in the lower left of your screen every 10 seconds (you can change this frequency by changing 10 in the command to a different value, and a value of zero disables the output). Within this information you will see an FPS (frame per second) value. This measures how well the server is performing and has nothing to do with any sort of rendering. Values of FPS ≥ 35 are generally considered good performance, while FPS ≤ 15 is poor performance. Obviously the higher the value, the better. If at any point during your mission you are consistently getting low FPS values, then you may want to consider implementing a headless client.

However, one can benefit from a headless client at any time. Even in missions that perform well, a HC can improve AI responsiveness and make the experience more challenging for those involved. It is for this reason that I recommend building your missions to work with or without headless clients. This takes a little planning and finesse, and will be covered more in depth in Section 2.

One last thing we must consider before proceeding is *where* will the HC be in relation to the server? With the HC being in control of the AI we must consider the headless client's ping in relation to the server. Ideally you want the lowest ping possible; many run their headless client's on the same network or in the same datacenter as their servers for this very reason. But above all else, test, test, and test some more! You may be perfectly satisfied with a 100ms ping HC.

2 Mission design

Before we can begin to discuss how to implement a headless client into your mission, we must first briefly discuss the concept of *ownership* in Arma. *All* units and objects (that are not part of the map) are owned by someone. Typically that someone is the server, but in some cases they may be owned by clients. Vehicles for instance, are owned by their driver. If you place an empty vehicle down in the editor it is initially owned by the server. Once you hop in the drivers seat, ownership is transferred to your computer, and remains there until you disconnect (ownership transferred back to the server), or someone else gets in as the driver (ownership transferred to the new driver). Killzone_Kid has some great tutorials [2] on locality I highly recommend if you are interested in learning more about this.

The point is, in order for the HC to take responsibility for any AI calculations, it must first *own* the AI. **The most effective way to accomplish this is having the HC spawn AI via a script.** It is possible to transfer ownership of units to the HC after the mission starts, but there are additional complications that come with this that I will discuss in Section 2.2.5.

[2] `http://killzonekid.com/arma-scripting-tutorials-locality/`

2.1 Adding a headless client to your mission

First we need to place a headless client virtual identity. In the editor place a new unit and select SIDE:Game Logic, CLASS:Virtual Entities, UNIT:Headless Client (see Figure 1).

Figure 1: Headless client placement within the Arma3 editor. It is important to make the unit playable, and give it a name.

To allow for the HC to actually connect, you will need to set CONTROL:Playable. Next you will need to *name* the unit; the reasoning will become apparent in the following sections. Since Arma3 supports multiple headless clients, I chose to name mine 'HC1'. Once the unit is placed, the module in the editor should resemble Figure 2.

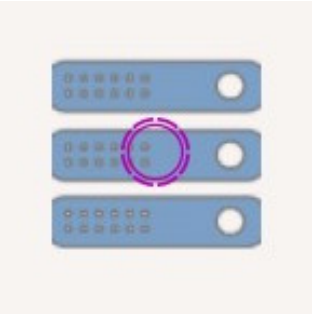

Figure 2: Headless client module in the editor when configured correctly. Notice the purple circle indicating that the unit is set as 'Playable'.

Once placed, a headless client connecting to this mission will *automatically* be assigned to the proper role in the mission lobby. However, only the current server admin will be able to see the HC.

2.2 Spawning units

2.2.1 Determining *where* to spawn units

I believe it is important to make your mission as dynamic as possible. That means ensuring that it works in situations with & without a headless client. To do that, we must figure out *where* to spawn our units. When the headless client is present we will spawn units on it, but in the absence of a HC we still want our mission to function, so we will spawn them on the server.

Now remember how we named our unit *HC1* in the previous section? We can use the command *isNil* [3] to determine if that spot is actually occupied on mission launch. This will tell us if the HC is present or not, and allow us to spawn the units in the correct place. The following code snippet should be placed into your *init.sqf* [4] so that it executes on *all* clients *and* the server. After the jump I will take you through what is happening here.

```
1  //define function to spawn units
2  _spawnUnits = {
3    //regular spawn method
4    [] execVM "spawn_viaScript.sqf";
5    [] execVM "spawn_EditorPlaced.sqf";
6
7    //DAC spawning
8    [] execVM "DAC\DAC_Config_Creator.sqf";
9    [] execVM "spawn_viaDAC.sqf";
10   True
11 };
12
13 //check if HC1 is present
14 HC1Present = if(isNil "HC1") then{False} else{True};
15
16 //spawn units on HC1 if present
17 if(HC1Present && isMultiplayer) then{
18   if(!isServer && !hasInterface) then{
19     [] call _spawnUnits;
20   };
21 }
22
23 //otherwise spawn units on the server
24 else{
25   if(isServer) then{
26     [] call _spawnUnits;
27   };
28 };
```

[3] https://community.bistudio.com/wiki/isNil
[4] https://community.bistudio.com/wiki/Event_Scripts

Lines 2-11

Here we define a function that will be responsible for spawning our units. It will be called by either the server **or** the headless client, not both. The content of each of these scripts will be discussed in the following sections.

Line 14

Here we check to see if the variable *HC1* is defined in the mission namespace, meaning that the unit is present. If it is defined, then HC1Present=True, otherwise HC1Present=False. This variable will be used to determine *where* to spawn the units within the next *if()* statement.

*Note: this unit will *always* be present in single player mode.

Lines 17-21

Here we check the value of *HC1Present* and *isMultiplayer*[5]. If it both are true then a headless is client present. Remember that every client and the server will execute this code; in order to spawn AI **only** on the HC we must first check two variables: *isServer* [6] & *hasInterface* [7]. If both of these variables are false (notice the !), then we are dealing with the headless client and we call the _spawnUnits function. All other clients and the server will bypass this function call because either *isServer* or *hasInterface* will be true.

*Note: if the *isMultiplayer* check is not performed here then the mission will *always* think that a headless client is present in single player mode and the _spawnUnits function will never be called.

Lines 24-28

If *HC1Present* or *isMultiplayer* comes back false, that means we have no headless client present, and this section of the code will execute. The goal now is to spawn the AI *only* on the server. To accomplish this, we simply check if the variable *isServer* [6] is true, then we allow for the _spawnUnits function to be called. This call will not happen on any non-hosting client because *isServer* will be false.

2.2.2 via Script

Let's take a look at how to spawn a small patrol via a script that will be executed on either a HC or the server. In order to spawn a squad we only need a starting location; my preferred method of defining this position is by an editor-placed marker. Pick where you want your patrol to start, and place an empty marker named "Patrol1". Now create a new file called *spawn_viaScript.sqf* in your mission's root directory and copy the following code:

[5] `https://community.bistudio.com/wiki/isMultiplayer`
[6] `https://community.bistudio.com/wiki/isServer`
[7] `https://community.bistudio.com/wiki/hasInterface`

```
1 _pos = getMarkerPos "Patrol1";
2 _cfgGroup = configFile >> "CfgGroups" >> "West" >> "BLU_F" >> "Infantry" >>
3             "BUS_SniperTeam";
4 _group = [_pos, WEST, _cfgGroup] call BIS_fnc_spawnGroup;
5 [_group, _pos, 100] call BIS_fnc_taskPatrol;
```

(where lines 2&3 should be on a **single** line). Here we are first getting the coordinates of marker "Patrol1", then temporarily storing the group name from the config file (can access this from the editor's Config Viewer). Next (line 4) we create the group via the BIS_fnc_spawnGroup[8] command. Last but not least, we tell the group to execute a patrol of radius 100m via the BIS_fnc_taskPatrol[9] command. Whoever executes this code (server or HC) will own the AI.

2.2.3 Editor-placed

Note that this section can be a tad overwhelming. Please remember to take your time, think things through, and refer to the sample mission for clarification.

Let's place some units down in the editor, and give them a set of waypoints. For this example I threw down a recon patrol and setup 4 waypoints with the last being of type CYCLE.

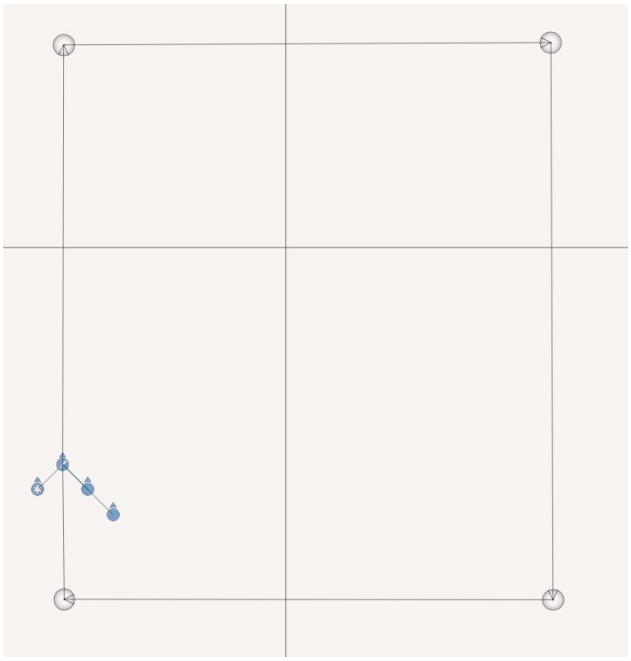

Figure 3: test

To spawn these units on either the server or HC, we must first get them into script form.

[8] https://community.bistudio.com/wiki/BIS_fnc_spawnGroup
[9] https://community.bistudio.com/wiki/BIS_fnc_taskPatrol

This is most easily accomplished with the help of a program called A2MC[10]. This little utility takes your *mission.sqm* file (this file holds all of the information for what you placed in the editor), and converts it into an *.sqf* file that we can safely modify.

Open A2MC, and point it to your *mission.sqm* file like I show in Figure 4.

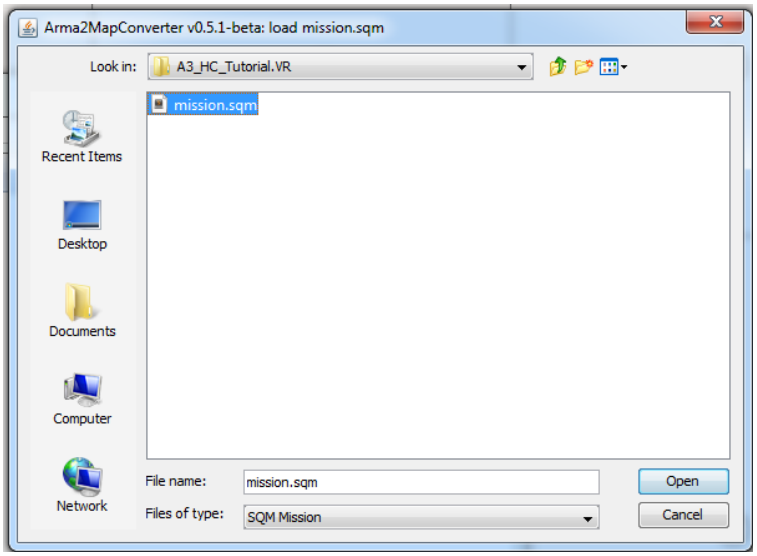

Figure 4: test

hit open and it should immediately bring up the save dialog as shown in Figure 5.

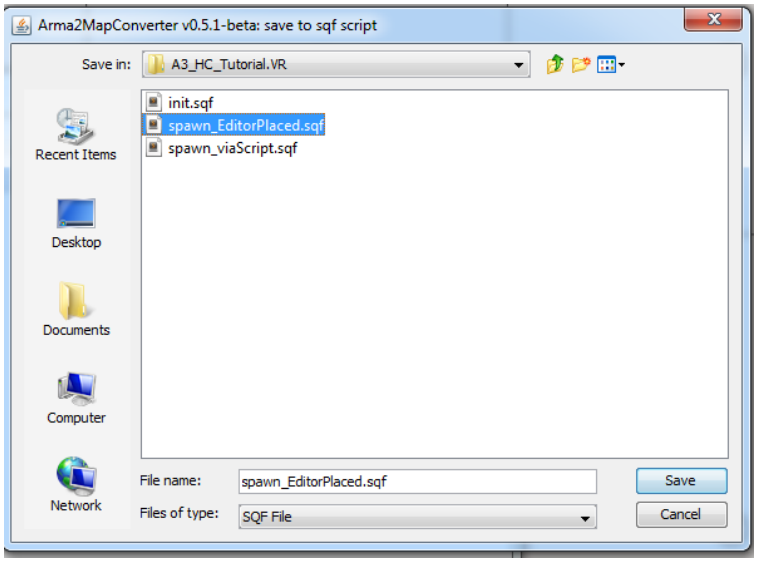

Figure 5: test

Now enter the file name *spawn_EditorPlaced.sqf* and hit save. When you open this file in a text editor, it will be composed of 4 main sections that are clearly marked:

[10] `http://www.armaholic.com/page.php?id=18012`

1. MARKER CREATION
2. EMPTY VEHICLE CREATION
3. UNIT CREATION
4. TRIGGER CREATION

Because our mission is so simple at this point, we can easily filter out what we do not need. The resulting file will contain ALL editor placed objects including units, waypoints, and triggers. We want to narrow down what is included in this file to *only* our Recon Patrol group and their waypoints. The first step is eliminating the parts that we do not need such as marker, empty vehicle, and trigger creation. Anything under each of these categories is safe to delete, BUT make sure you **keep the last line** that reads *[_createdUnits, _createdMarkers, _createdTriggers]*!

Now all we have left is under UNIT CREATION. Take a look at what this is doing; it first creates a group with the *createGroup* command, then creates the actual unit and sets the unit properties. If we continue to scroll down, you should notice a line that says "Waypoints for group _group_west_2", and they should be the only waypoints present. This means that the group we should be concerned with is called *_group_west_2* throughout the script. All units that do *not* belong to this group, should be deleted as we will be *keeping* them in the editor and do not want to duplicate them.

But which units are not a part of that group? If you look closely, it's the first two. The first is likely the player unit, while the second is the HC unit. Delete the two units between the bottom of UNIT CREATION, and "// group _group_west_2" and we should be set to go. We can now *delete* our Recon Patrol and their respective waypoints in the editor - remember they will be spawned via this script!

2.2.4 via DAC

Dynamic-AI-Creator, or DAC[11], is an excellent way to create a more dynamic and flexible mission. It basically involves placing down AI 'zones' which will spawn a specified number of units within said zone. There are numerous options and benefits to using DAC that I cannot cover here; I highly suggest you take a look at the documentation to learn more.

***Note: DAC from the web is not compatible with headless clients as it was not designed with it in mind. I recommend you copy the DAC folder from the example mission included in this tutorial as I have performed the necessary modifications for you.**

To take a DAC zone and convert it to script we will follow a procedure very similar to what was done in Section 2.2.3. First we will place a DAC trigger zone in the editor as is shown in Figure 6. To understand what is going on you will need to reference the DAC

[11] `http://www.armaholic.com/page.php?id=25550`

documentation, but ensure that the trigger name matches the name passed to the DAC_Zone function in the ON ACT box of the trigger. Once the zone is placed feel free to experiment in the editor to make sure that things are spawning properly on the server.

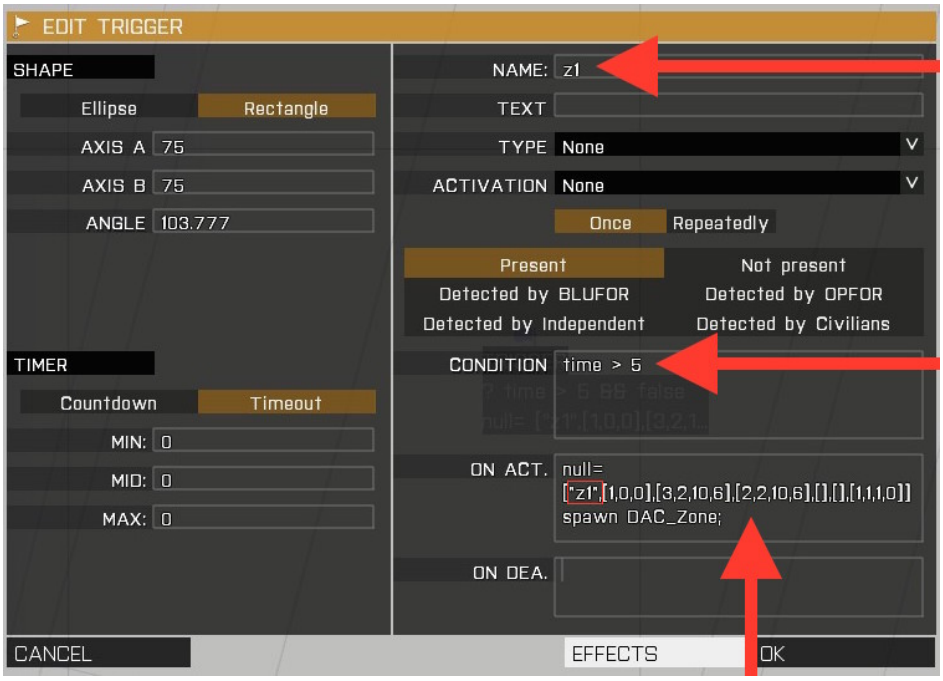

Figure 6: Setting up a DAC zone. We set the condition to a time greater than 5, this is just to add a short delay before DAC initializes. Notice the name of the trigger matches the name we pass to the *DAC_Zone* function in the trigger activation. For more details on how to spawn DAC zones please see the DAC documentation.

Next we will convert this zone to a script using A2MC following the exact same procedure described in Section 2.2.3. The difference here will be that we will only concern ourselves with the DAC zones in the TRIGGER CREATION section of the script output by A2MC. Basically, you want to filter through the file and delete all markers, vehicles, units, and triggers that are *not* DAC related. *One major difference however, is that triggers converted via A2MC have an extra set of quotation marks for the CONDITION field of the trigger which causes these triggers not to fire. To resolve this you will need to modify each trigger's setTriggerStatements command and remove the extra set of quotation marks.* Notice the difference in these two lines:

```
1 z1 setTriggerStatements[""time > 5"", "null= [""z1"" ,...
2 z1 setTriggerStatements["time > 5", "null= [""z1"" ,...
```

Line 1 is directly from A2MC, and Line 2 is the correction with the extra set of quotes removed from around the *time > 5* statement.

With the trigger CONDITION statement fixed, we can go back into the editor and *disable* the in-game trigger from firing. Keeping it around is ideal since it is a source of

visual feedback in the editor, and will allow for easy changes in the future (edit the trigger, re-do the A2MC procedure). To disable the trigger from firing, we simply add '*&& false*' to the trigger's CONDITION statement as shown in Figure 7:

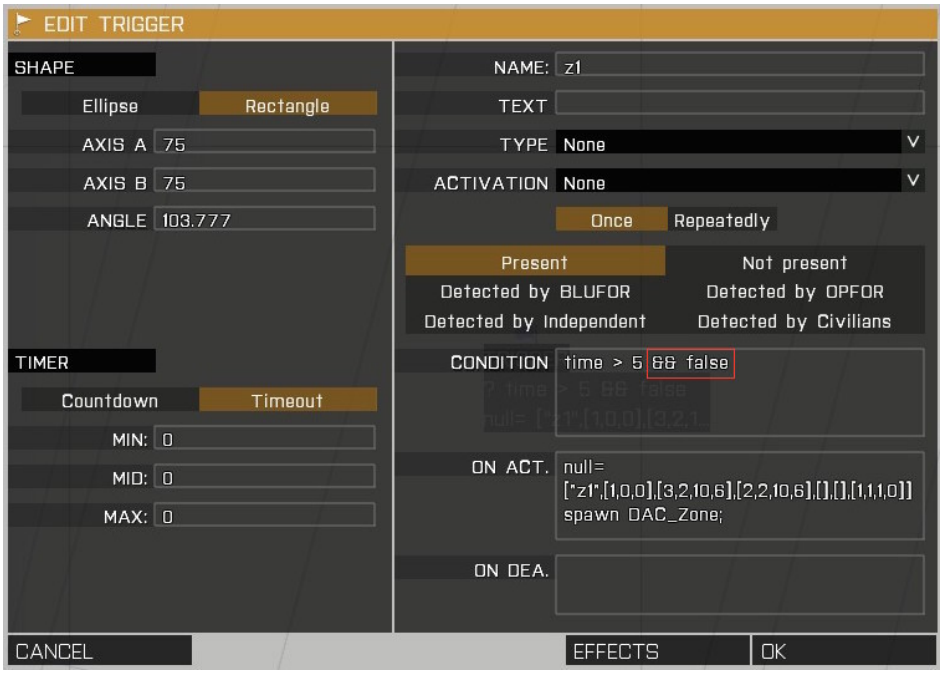

Figure 7: By adding the '&& false' the trigger will no longer fire - this is a desired behavior because our script is now creating this trigger on either the server or HC. Keeping the trigger around however, is useful for visual purposes.

To make sure this new script from A2MC gets executed name it spawn_viaDAC.sqf and place it in your mission's root directory. Since our unit spawning function within *init.sqf* (Section 2.2.1) already includes a call to this script, it should be executed on mission launch.

2.2.5 Transferring after mission start

Transferring units to the headless client *after* mission start means that you can leave them in the editor. This is **not** the preferred way of doing things, mainly because *these units will lose any waypoints, behaviors, or objects synchronized to them in the transfer.* Nevertheless, this can sometimes be helpful for specific units such as as those manually placed in a guard tower.

To transfer units to the headless client one only needs to know the client ID of the headless client. This is accomplished by having the server execute the *owner*[12] command. The next step depends on which version of Arma3 you are running; currently any version < 1.4 requires the *setOwner*[13] command, while versions ≥ 1.4 will switch to using the

[12] `https://community.bistudio.com/wiki/owner`
[13] `https://community.bistudio.com/wiki/setOwner`

setGroupOwner[14] command. See the HC development thread[15] on BI's forums for further details.

To make things easier, I have included a script called *moveToHC.sqf* with the included tutorial mission. The basic premise is to have each unit you would like transferred execute a script that transfers ownership, which looks *something* like this:

```
1  if(!isServer) exitWith{};       //only run on the server!
2  if(!isMultiplayer) exitWith{}; //only run in MP environment!
3
4  _unit = _this select 0;    //first passed variable is the unit
5  _HC   = _this select 1;    //second passed variable is the HC name (HC1 here)
6
7  if(isNil format["%1",_HC]) exitWith{};  //only run if HC1 is present
8
9  _HCid      = owner _HC;     //get HC client ID
10 _unitGroup = group _unit;   //get unit's group
11
12 //setOwner for versions < 1.4
13 { _x setOwner _HCid; }forEach (units _unitGroup);
14
15 //setGroupOwner for versions >= 1.4
16 _unitGroup setGroupOwner _HCid;
```

Place the following in each unit or group leader's INITIALIZATION box:

```
null = [this,HC1] execVM "moveToHC.sqf";
```

You should only allow either Line 11 **or** Line 16 to execute depending on which version of Arma3 you are running, **not** both. This means comment one of them out! Remember that all waypoints, behaviors, and synchronized objects *will be lost* in the transfer.

2.3 Additional considerations

There are a few additional nuances that one must consider when designing a headless client compatible mission:

- *playableUnits*[16] (multiplayer) & *switchableUnits*[17] (single player) will both return arrays that include the headless client if present.
- Any scripts that alter the behavior of the AI (such as TPWCAS[18]) should be executed on *both* the server & the headless client.
- You do not have to transfer *every* unit to the headless client. Striking a balance here is key.

[14] https://community.bistudio.com/wiki/setGroupOwner
[15] http://goo.gl/pWWIxh
[16] https://community.bistudio.com/wiki/playableUnits
[17] https://community.bistudio.com/wiki/switchableUnits
[18] http://www.armaholic.com/page.php?id=19467

2.4 Summary

- Spawning units via script allows for the most flexibility in *where* your units are spawned (Section 2.2.1).
- HC compatible missions should ideally work with or with*out* the HC present.
- Spawn scripts can be created manually (Section 2.2.2), or generated with A2MC (Sections 2.2.3 & 2.2.4).
- Units can be moved to the HC after mission start (Section 2.2.5) at the cost of losing all waypoints, behaviors, and synchronized objects.
- Multiple headless clients *are* supported by Arma3. The ideas and methods here can easily be extended to support multiple HCs.
- *hasInterface* is only *False* for dedicated servers and headless clients - all other systems will return *True* for this variable.

3 Preparing your server for headless clients

Servers are *not* configured by default to allow HCs to connect. You must make minor modifications to the server configuration in order to allow for their connection, and allow for unlimited bandwidth throughput. This is done by adding the HC's IP address to the server.cfg file like so:

```
headlessClients[]={"xxx.xxx.xxx.xxx"};
localClient[]={xxx.xxx.xxx.xxx};
```

Note however, that you need to replace 'xxx.xxx.xxx.xxx' with the actual IP address of your headless client! Multiple headless clients and/or multiple IP addresses are also supported; see the BI wiki for further information[19].

[19] https://community.bistudio.com/wiki/Arma_3_Headless_Client

4 Starting & connecting a headless client

In this section we will cover how to start and connect your HC to the server. This is done through either the 'arma3' or 'arma3server' executable. The basic premise is to add the following parameters to the executable:

- -client
- -connect=xxx.xxx.xxx.xxx
- -port=xxxx

Next I'll go into a bit more detail on how to launch the HC on both a Windows & Linux machine.

4.1 Windows

To launch a HC on your windows machine with MOD support you'll likely want to create a new batch file. You can do this within the confines of a shortcut, but if you have a good number of mods you will not be able to fit them all. We are going to create a batch[20] file instead.

Start by creating a new text file and name it *startHC.bat*. This batch file is essentially a shortcut, saving you the time of typing it all in the command line manually execute. Here's an example from my machine:

```
1 cd /D "D:\Games\Steam\steamapps\common\Arma 3"
2 "D:\Games\Steam\steamapps\common\Arma 3\arma3server.exe" -client
3 -connect=127.0.0.1 -port=2302 -password=password
4 -mod=@cup;@allinarmaterrainpack;@asdg_jr;@task_force_radio;
```

Line 1 changes your working director to the location of the arma3server executable. Since this is MY directory, you will need to modify this to point to YOUR Arma3 directory. Lines 2, 3, & 4 are actually **one** line (I just couldn't fit them all in one line for this document). It is responsible for actually executing the headless client command. You'll notice I include the -client, -connect, -port, and the -password commands. These values will need to be changed accordingly to allow the HC to connect to YOUR server. Also make sure to modify the -mod line to include the appropriate mods!

Once the file is edited and saved, double click on *startHC.bat* to start the headless client.

4.2 Linux

Launching the HC on linux is very similar to windows, except better (I am biased). First you'll need the arma3server executable. To get this, I recommend following the linux installation instructions on BI's wiki[21].

[20] http://en.wikipedia.org/wiki/Batch_file

[21] https://community.bistudio.com/wiki/Arma_3_Dedicated_Server

Once you have all of that straightened out, you can create a new bash[22] script for *startHC.sh* with the following contents:

```
1  #!/bin/bash
2
3  # define directories
4  HOME=/home/bob/local/A3DS
5  A3DIR=$HOME/arma3
6
7  # setup mods
8  MODS=""
9  MODS+="@cup;"
10 MODS+="@allinarmaterrainpack;"
11 MODS+="@asdg_jr;"
12 MODS+="@task_force_radio;"
13
14 # execute
15 cd $A3DIR
16 ./arma3server -client -connect=127.0.0.1 -port=2302 -pass=password -mod=$MODS
```

Once again you will notice I include the -client, -connect, -port, & -pass parameters exactly like the windows batch file. This particular bash script is tailored to my system; to get this up and running on your box you will need to modify Line 4 (home directory), Line 5 (arma3 directory), the individual MODS you would like to include ($MOD variable), along with the IP/port/password for the server you are connecting to.

Before you can execute this script you will need to give it executable permissions. Open a terminal and change directory to where the script is located, then give permissions via:

```
chmod +x startHC.sh
```

now execute with:

```
./startHC.sh
```

to start the headless client (keep the terminal window open!).

5 Closing thoughts

Be it creating a new mission, or retrofitting an old mission, implementing a headless client should be relatively straight forward with the tools and ideas presented here. The technology is constantly being developed by BI so keep an eye on the BI headless client dev thread[23] for the latest changes. I will do my best to keep this document updated as the game evolves, but by and large the concepts should remain the same. Enjoy!

[22] http://en.wikipedia.org/wiki/Bash_%28Unix_shell%29
[23] http://goo.gl/pWWIxh

6 References & helpful links

- BI Wiki: https://community.bistudio.com/wiki/Main_Page
- BI Forums: http://forums.bistudio.com/forum.php
- HC forum thread: http://goo.gl/pWWIxh
- Killzone_Kid: http://killzonekid.com/
- Armaholic: http://www.armaholic.com/
- Dslyecxi: http://dslyecxi.com/
- DAC: http://www.armaholic.com/page.php?id=25550
- A2MC: http://www.armaholic.com/page.php?id=18012
- Cover photo: http://cdn.neatorama.com/jill/Headless-Horseman_55646-l.jpg