A COMPREHENSIVE GUIDE TO CREATING HEADLESS CLIENT COMPATIBLE ARMA2 MISSIONS

by: **Monsoon**

rweezera@hotmail.com

February 22, 2013

Contents

1 Introduction

Bohemia Interactive recently introduced the ability to offload ArmA2 AI calculations onto what is called a 'headless client' or HC. It is essentially a connected client that does no rendering. The advantage this brings is twofold: first it reduces the workload on the server itself, and secondly it allows the AI their own dedicated processor - which makes them far more deadly and responsive. As a mission creator this opens up an entirely new realm of possibilities; previously if there were too many AI present then the game slowed down considerably, degrading client performance. Now we can leave the server process to handle communications while the HC handles the AI calculations. The improved responsiveness was most famously shown by ShackTac member Dslyecxi[1].

It is possible to 'retrofit' a mission to be HC compatible, but it takes some work. It is far easier to actually design your mission with this process in mind. Today I am going to walk through the creation of a simple mission that will be both HC and non-HC compatible. I will incorporate both an activated & de-activated DAC[2] zone, manually placed units that start on the HC, and manually placed units that will be transferred to the HC after the mission launches.

***Disclaimer*: I put this together due to a lack of comprehensive guides on setting up and using headless clients. All of what follows is from personal experience retrofitting my own mission. This means that there very well may be better ways to do this. If I'm doing something crazy, please let me know!**

2 Mission design

When designing a mission with a HC in mind, things become quite simple once you understand the process of which you go about it. Everything placed in the editor and remaining in the editor when you create the PBO file will initially be controlled by the server. If we want groups of AI to be controlled by the HC then we must somehow 'transfer ownership' to the HC from the server. There are two possible ways of accomplishing this: the first is to have the HC actually spawn the AI, and the second it to have the server transfer ownership to the HC after the mission is initialized. The second option is more of a pain, so for the majority of the time we will stick with the first option.

2.1 Placing the player and HC

Before we start worrying about AI, let's place our Player & HC units in the editor. Create a new mission and name it 'hc_tutorial', then place two units as shown:

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

[2] `http://www.armaholic.com/page.php?id=10621`

Figure 1: [hc_tutorial] Placing player - notice the name 's1' to allow DAC to recognize it.

Figure 2: [hc_tutorial] Placing the HC - notice the name 'HC' and Control=Playable, also disallowing damage via 'this allowDamage false;'.

You can make the HC any unit you like, but I chose a boar for fun. Once it is placed you should manually edit the mission.sqm by locate the 'HC' unit, then add the bolded line (*line 11*) at the bottom:

```
1   class Item0
2   {
3     position[]={3690.2991,19,3610.8123};
4     id=1;
5     side="AMBIENT LIFE";
6     vehicle="WildBoar";
7     player="PLAY CDG";
8     skill=0.60000002;
9     text="HC";
10    init="this allowDamage false;";
11    forceHeadlessClient=1;
12  };
```

Line 11 forces the headless client to join as this particular unit at mission start. Sometimes this doesn't work, so keep an eye on it if you're admin'ing the mission. This step is not necessary, but if not performed the admin will have to manually put the HC into the correct spot at each mission start.

2.2 Simple AI with waypoints

As a simple example let's place our first group of units, and give them some waypoints.

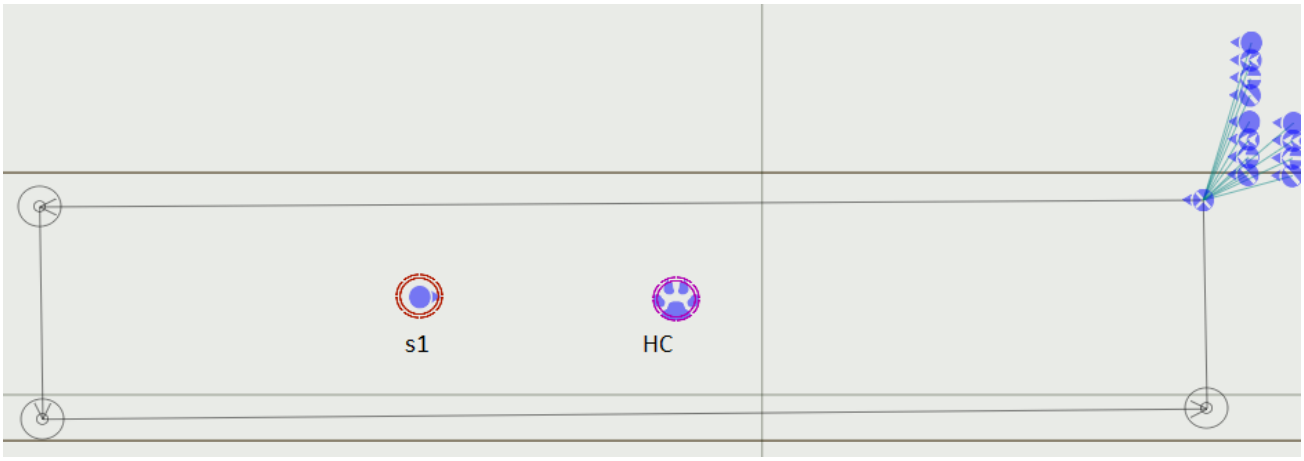


Figure 3: [hc_tutorial] I've placed a BLUFOR rifle squad and ordered them to move in a square along the airstrip.

Now we can think about offloading these AI to the headless client. However, we must first make adjustments to our description.ext and init.sqf files if we would like this mission to be both HC and non-HC compatible. The server currently has no way to dynamically determine if a HC is present or not, so we must set this by hand. We start by creating a lobby parameter that turns the HC off and on. From the mission directory open or create the file named 'description.ext'. Add this snippet of the code to the bottom of the file:

```
class Params
{
  class HeadlessClient
  {
    title = "Headless Client";
    values[]= {0,1};
    texts[] = {"OFF","ON"};
    default = 0;
  };
};
```

This creates a new 'Parameters' button in the mission's lobby which will allow us to set the headless client to ON(1) or OFF(2) wherein the default is OFF. Now we must retrieve this parameter in the 'init.sqf' file which will allow us to inform the server and all clients. Either create or open 'init.sqf' and add the following segment to the very top, we want this to be the first thing your mission executes:

```
1  if(isNil "paramsArray") then{ paramsArray=[0,0,0]};
2
3  //get mission parameter to see if
4  //HeadlessClient is present and assign its name
5  if(paramsArray select 0 == 1) then{
6    if(isServer) then{
7      HCPresent = true;
8      publicVariable "HCPresent";
9    };
10   if (!hasInterface && !isServer) then{
11     HCName = name player;
12     publicVariable "HCName";
13   };
14 } else{
15   if(isServer) then{
16     HCPresent = false;
17     HCName = "NOONE";
18     publicVariable "HCPresent";
19     publicVariable "HCName";
20   };
21 };
```

This code snippet will be executed by all clients, the server, and the headless client. The goal of this snippet is to determine if the HC is set to ON, and if it is then get the actual player name of the HC. *Line 5* is a simple *if* statement that retrieves the variable that will be set in the mission lobby. If 1 is returned then the server knows a HC is present. The server will then set 'HCPresent' to true and make the variable global so all clients can see the value (*lines 7-8*). We next check to see if each client has an interface and is *not* the server (*line 10*). This means that only the HC will execute the next two lines, which retrieve that player's name (*lines 11-12*). In the case of our example mission, the name of the HC is simply 'HC' as we set in the editor. If the HC is set to OFF in the mission lobby, then the server sets 'HCPresent' to false, and 'HCName="NOONE"' (*lines 16-19*). This will be very important later when we begin to construct DAC zones.

But for now let's focus on the small group of soldiers. We want to spawn these via script so they can easily be spawned on the HC or the server. The easiest and most automated way is to use A2MC[3]. You simply point this executable to your mission.sqf, and it essentially outputs a script version. The most convenient way I've found is to delete all NON-HC compatible (I'll discuss compatibility in Chapter 4) AI, players, triggers, logics, etc from the mission, and save it as a copy such as 'hc_tutorial_aionly'. So now the AI-only copy of the mission may look something like Figure 4:

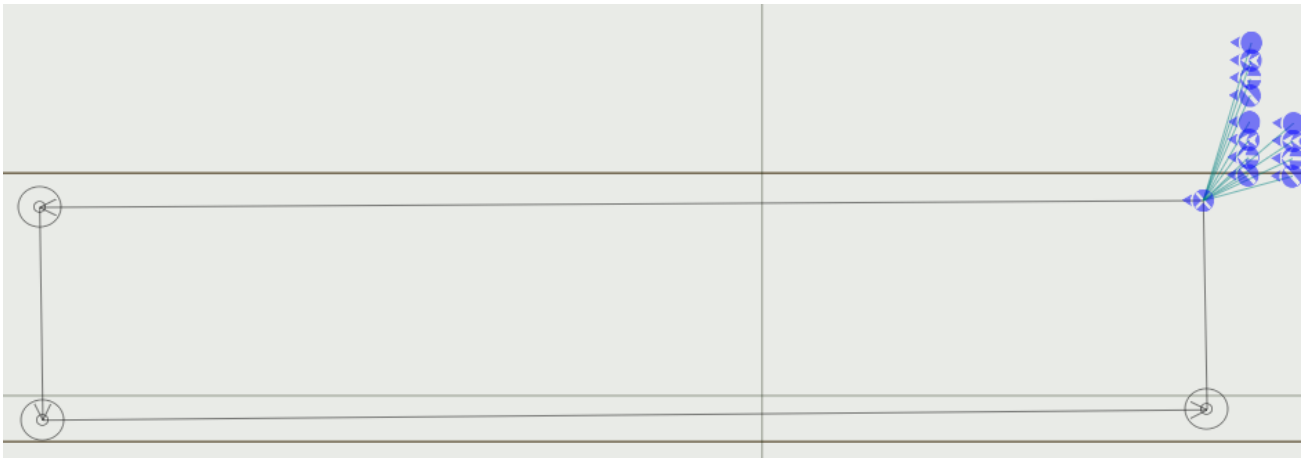
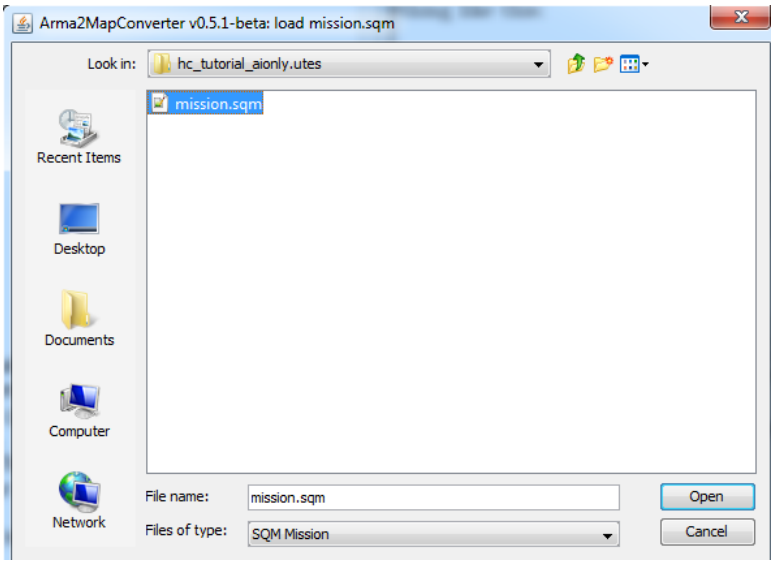


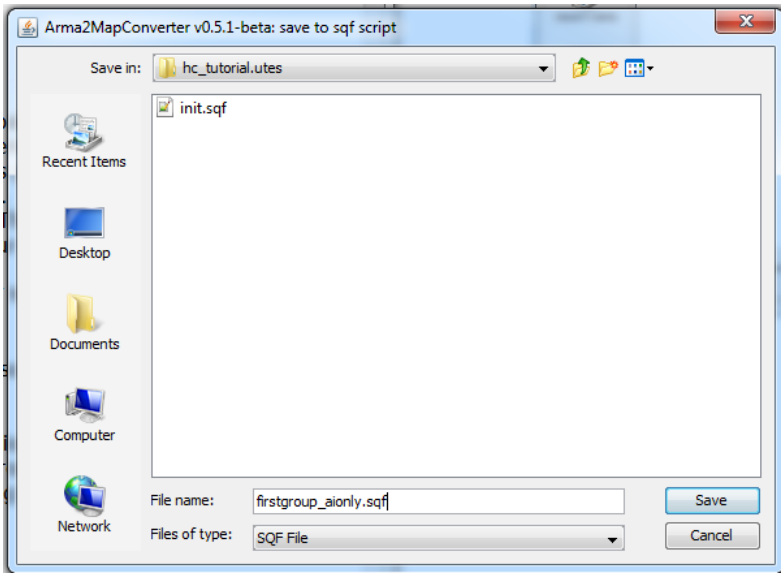
Figure 4: [hc_tutorial_aionly] Only AI and their waypoints remain (compared to Figure 3).

Open 'a2mc.exe' (must have java installed), and point it to your mission.sqf for the AI-only mission as shown here:



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

Click open, and it will prompt you with another dialog asking you where to save the new file. Save it anywhere you like, preferably in the MAIN mission directory. For this example I will name it 'firstgroup_aionly.sqf'



Open the file and take a look at the structure. It has a section for Markers, Empty Vehicles, & Units. Under the Units category you will see that it has created a new group followed by the creation of each unit within that group individually. After all units have been created you will find each group waypoint individually created. This is the script we will be calling from init.sqf to spawn these units in. But now that we have these units in a script, we need to *delete* them from the actual mission. The main mission should now look like Figure 5:

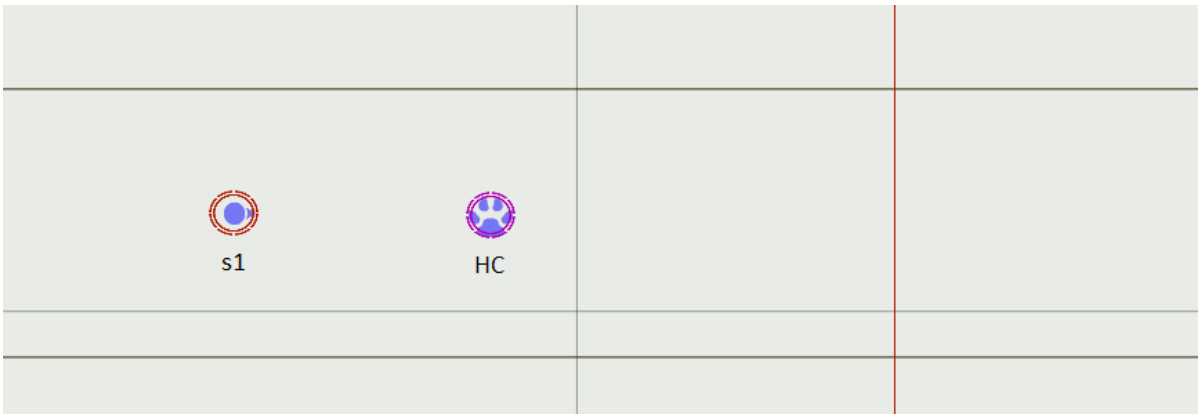


Figure 5: [hc_tutorial] AI have been deleted and only the player and HC units remain. (compared to Figure 3).

Remember how earlier we setup the mission lobby parameter to determine if a HC was present or not? Well now we're going to use those variables to determine *who* spawns this

group of units. In your init.sqf file insert this snippet of code below the previous (and below briefing if present):

```
1  if(!isServer) then {waitUntil{!isNull player}};
2
3  if(HCPresent) then{
4    if(!hasInterface && !isServer) then{
5      execVM "firstgroup_aionly.sqf";
6    };
7  } else{
8      if(isServer) then{
9        execVM "firstgroup_aionly.sqf";
10     };
11 };
```

Line 3 checks if HCPresent is set to true. If so, we execute *line 5* ONLY on the HC. This will spawn the units with their waypoints we created in 'firstgroup_aionly.sqf' on the HC. If HCPresent is set to false, then this script is executed on the server only (*line 9*), and the units are controlled by the server. Now we have a mission that will be fully compatible with HC and non-HC configurations depending on the parameter set in the mission lobby.

2.3 Transferring units to the HC after mission launch

Occasionally you'll want to transfer units to the HC from the server *after* the mission launches. One such example is the placement of units in a guard tower. As I'll mention in Chapter 4, units placed above sea level (ASL) do not work out so well when placed by a script.

For this example I will go back to 'hc_tutorial' and place two units in a guard tower, they will *remain in the editor*. You can very easily nab building positions with the Visual House Position Module[4] by Grizzle; I *highly* recommend this addon for editing if you plan on manually placing units anywhere on the map. I've nabbed two positions in a nearby tower and added two units. The first guard placement can be see in Figure 6.

[4] http://www.armaholic.com/page.php?id=11288

Figure 6: [hc_tutorial] Placing tower guard named 'tower_guard1' using setPosASL.

The next part is relatively easy. I've included the script 'setowner.sqf' that should do the bulk of the work for you in transferring the units to the HC. It looks like so:

```
1  if !(isServer) exitWith{};
2  _HC    = _this select 0;
3  _list  = _this select 1;
4  OC_pass = 0;
5  OC_fail = 0;
6  for "_i" from 0 to ((count _list)-1) do{
7    _unit = _list select _i;
8
9    if(_unit setOwner (owner _HC)) then{
10     OC_pass = OC_pass + 1;
11   }
12   else{
13     OC_fail  = OC_fail + 1;
14   };
15 };
16 publicVariable "OC_pass";
17 publicVariable "OC_fail";
```

The goal here is to pass an array of object (units in this case) to this script along with the HC's name. Then, the script will only run on the server, and cycle through each unit (*line 6*) transferring ownership to the headless client (*line 9*). The variables OC_pass and OC_fail are simply for debugging purposes. Every client connected to the server gets a unique ID number. This ID number is only revealed to the server via the 'owner' command. In *line 9* we see that each unit is given the command 'setOwner', followed by the client ID number of '_HC'. If this comes back as true, then the transfer completed successfully and we add one to OC_pass variable.

To call this script, we simply create a trigger with the condition that HCPresent=true as is seen in Figure 7. Two conditions must be true in order to activate this trigger. The first is a specified time period which I change depending on the complexity of the mission, and the second is HCPresent=true determined in the init.sqf. In the 'On Act.' portion of this trigger there are two commands. The first calls the script, and the second simply prints out the value for OC_pass for verification and is not always needed. Once the trigger is activated we pass the object HC, and an array of units (tower_guard1 & tower_guard2) to the variables '_HC' and '_list' in the script itself. If HCPresent=true the script will execute transferring ownership to the HC, if not then the trigger will never be activated and those units will continue to be controlled by the server. The unfortunate part however, is that all waypoints and synchronizations will be *lost* when transferring to the HC in this manner. *So I would recommend that you only use this method when transferring stationary units manually placed in or around buildings.*

Figure 7: [hc_tutorial] Transfer of units to HC after mission launch.

2.4 DAC Zones

Spawning units via DAC on the HC can be a bit tricky. Things get even more complicated when you realize that you can spawn zones that are already activated (AI present), or zones that are deactivated (and activated later). Deactivated zones can be important when considering non-HC missions as it can free up CPU time for the server. In the next subsection I'll outline how to handle both activated and de-activated DAC zones.

2.4.1 DAC Zones in script form

Activated DAC zones are very similar to placing AI with waypoints as was shown in in Chapter 2.2. We are going to create a DAC zone in the editor, delete everything BUT that DAC zone, then export to a script via A2MC as we did before. To kill two birds with two stones, I'll show you how to handle de-activated zones as well, which are zones later activated by an in-game trigger. Let's first create two DAC zones, one BLUFOR (activated), and an adjacent OPFOR (deactivated). These can be seen in Figures 8&9 respectively.

Figure 8: [hc_tutorial] BLUFOR DAC zone creation

Figure 9: [hc_tutorial] OPFOR DAC zone creation

Nearby we'll create a trigger grouped to 's1' that will activate the deactivated zone (Figure 10):

Figure 10: [hc_tutorial] Activate the de-active OPFOR DAC zone. This trigger is grouped with the unit 's1'.

The overall mission should currently look something like this if you've been following along:

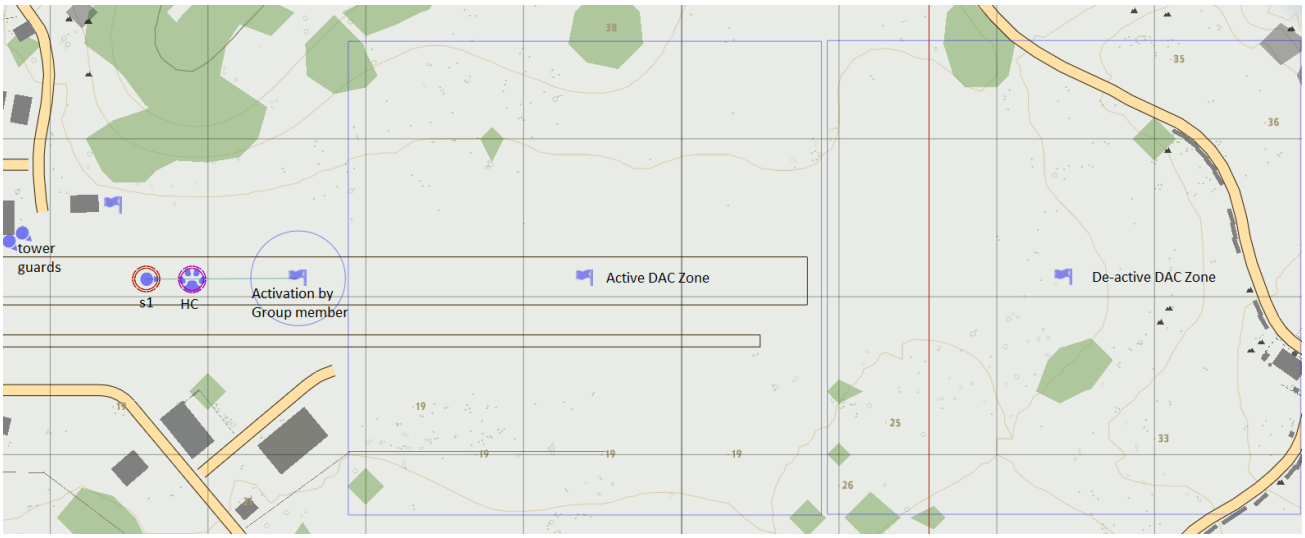


Figure 11: [hc_tutorial] Overview of BLUFOR & OPFOR DAC zone positions along with the activation trigger.

Now delete *everything* except for the DAC zones, and save a copy of the mission as 'hc_tutorial_daconly'. We can then run the mission.sqm of the DAC-only mission through A2MC, and produce 'dac_zones.sqf' in the main mission directory.

This file has some issues though. You have to change the **setTriggerStatements** sections; they currently look like this:

```
...
[""true;"", "null=[""activezone"",[1,0,0],[10,5,25,25],[10,2,20,10],[ ],[ ],
                  [1,1,1,0]] spawn DAC_Zone", ""];
...
[""true;"", "null=[""deactivezone"",[1,1,0],[10,5,25,25],[10,2,20,10],[ ],[ ],
                  [0,0,0,0]] spawn DAC_Zone", ""];
...
```

It must be changed to look like this:

```
...
["true;", "null=[""activezone"",[1,0,0],[10,5,25,25],[10,2,20,10],[ ],[ ],
                [1,1,1,0]] spawn DAC_Zone", ""];
...
["true;", "null=[""deactivezone"",[1,1,0],[10,5,25,25],[10,2,20,10],[ ],[ ],
                [0,0,0,0]] spawn DAC_Zone", ""];
...
```

Notice I changed from double quotes to single quotes on the trigger's init: ""true"" → "true". Without these manual edits the DAC zones will not spawn.

The last modification we must make, is to declare the zone objects as a 'publicVariable' when the headless client creates them. If this is not done, then de-activated DAC zones may not properly activate when triggered. At the bottom of 'dac_zones.sqf' add:

```
1 //if HC is inactive - make triggers global
2 if(!HCPresent) then{
3   _i = 0;
4   for "_i" from 0 to ((count _createdTriggers)-1) do{
5     publicVariable format["%1",(_createdTriggers select _i)];
6   };
7 };
```

immediately *after* the last trigger creation, and *before* '//return all created units in an array'.

Once this file is created & modified, we need to load up the main mission and *remove the DAC zones from the editor* as they will now be spawned via script (REMEMBER to keep the trigger grouped to 's1'). We'll add a script execute command in our init.sqf in the next section.

2.4.2 Turning DAC on

Before DAC will execute on a HC, we must make some modifications to the actual DAC scripts. They're a bit of a pain so I recommend using the DAC folder included in the sample mission directory as I have made the modifications for you. But simply put you would need to change the *if(isServer)* statements to *if(isServer || (name player == HCName))* throughout the code. But as I said before, the DAC directory in the sample mission folder has this already done for you.

Next we must modify our init.sqf to initialize DAC, and our new DAC zones. The bottom of your init.sqf should be changed to look like so:

```
1  if(!isServer) then {waitUntil{!isNull player}};
2
3  DAC_Zone = compile preprocessFile "DAC\Scripts\DAC_Init_Zone.sqf";
4  DAC_Objects = compile preprocessFile "DAC\Scripts\DAC_Create_Objects.sqf";
5  execVM "DAC\DAC_Config_Creator.sqf";
6
7  //execute DAC on HC if present, otherwise on Server
8  if(HCPresent) then{
9    dac_on = false;
10   if(!hasInterface && !isServer) then{
11     execVM "dac_zones.sqf";
12     execVM "firstgroup_aionly.sqf";
13   };
14 } else{
15     if(isServer) then{
16       execVM "dac_zones.sqf";
17       execVM "firstgroup_aionly.sqf";
18     };
19 };
```

Here I've added the initialization of DAC (*lines 3-5*), and the script to spawn the DAC zones for the HC (*line 11*) or Server (*line 16*).

3 Execution

3.1 Starting & connecting the headless client

Starting the HC is quite simple. You just need to alter your batch file to include the '-client' flag, along with the IP address and password of your server. I would also recommend making a new player profile for the HC. Here's an example batch file (be sure to include any needed mods!):

```
arma2oa.exe -client -connect=127.0.0.1 -password=YourServerPassword
  -cpucount=2 -nosound -profiles=HC -name=HC -mod=@ACE;@ACEX;@ACEX_RU...
```

3.2 Server settings

The HC itself is treated like a regular client by the server. Consequently the bandwidth between the HC & server will be limited. This will result in some latency between the two, and hence latency with your HC AI. Fortunately, it is a very simple fix. Edit your server's configuration file and add the following line:

```
localclient[]={"127.0.0.1"};
```

This tells the server to not restrict bandwidth coming from this IP address (127.0.0.1 is known as the localhost and points back to the same computer). If you're like me and you sometimes change which computer handles the HC, you can add multiple IP addresses delimited by a comma like so:

```
localclient[]={"127.0.0.1,192.168.1.200"};
```

Another consequence of connecting with the '-client' switch is that the HC needs its own CD-Key for Arma or the server will detect a duplicate and reject the connection (assuming you are connecting as well). To avoid this you can have your server stop reporting to the GameSpy servers via altering the 'reportingIP' line in your server's config file to read:

```
reportingIP="<>";
```

WARNING: your game will *not* show up in the game browser, people must connect manually. It will also allow banned, pirated, and duplicate CD-Keys to connect. Obviously only do this if you trust your players.

4 Compatibility & considerations

When converting units, triggers, and game logics via A2MC there are some things you need to consider:

- Units placed using 'setPosASL' on or around buildings will not end up in the correct spot
- Any synchronized or grouped trigger will not transfer to the HC properly.

- Units transferred to the HC after mission launch will lose all waypoints.
- There may be other things I am missing, but that's all I can think of for now.

5 Final thoughts

Any mission can be adapted to work with HC once you understand the locality of who controls what. Things need to be either created on the HC or Server via script. Otherwise you need to transfer ownership after mission launch if you want them to reside on the HC. Remember though, *you don't have to put everything on the HC*. A well designed mission will have AI residing on both the server & the HC. Headless clients are not going to solve all of your problems, but they sure do help.

Thank you for reading, I certainly hope this has helped in some way shape or form. If you have any questions or comments please feel free to shoot me an email: rweezera@hotmail.com.

-Monsoon

In the included example mission, the two DAC zones are set to spawn hundreds of infantry and vehicles for both BLUFOR and OPFOR. Here's how things look without the HC:

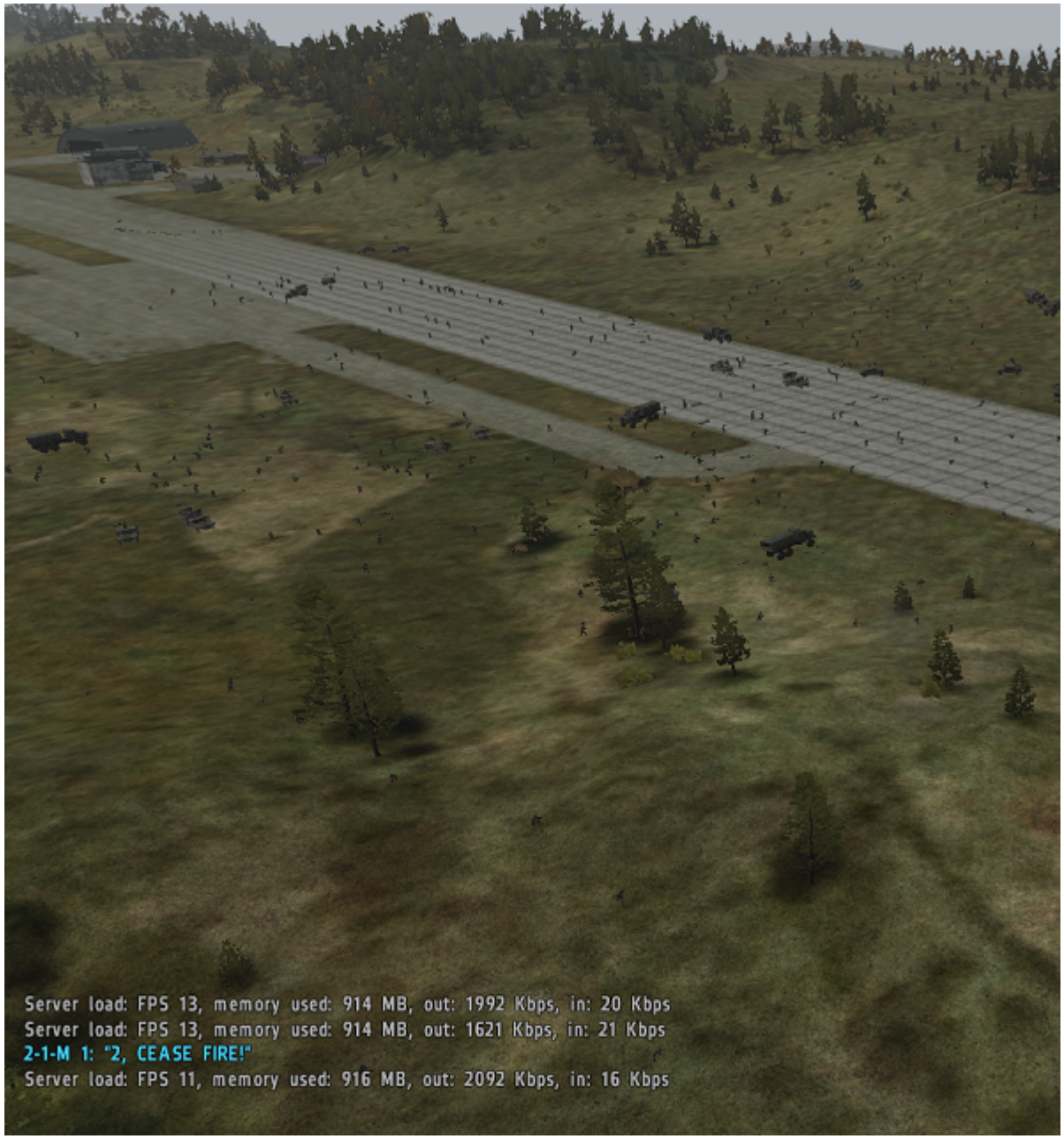


Figure 12: [hc_tutorial] hundreds of units controlled by the server. Average server FPS∼12.

Here's the server with the AI offloaded to the HC. In this shot the HC's processor was at 100% and could no longer spawn any units, but the server's performance was still excellent:

Figure 13: [hc_tutorial] hundreds of units controlled by the HC. Average server FPS∼43.