7/13/2013

[FOCK]ERS

FOCKERS ARMA 3 SCRIPTING GUIDE – CHAP3

*Not – do not copy over the text – type it yourself as the " " are different in the editors.

Introduction

Why have I made this?

Well, I've been messing around with Arma and Arma2 for the last couple of years and although there are guides (See links below) out there to help along the way, a lot of learning was done through reading the various forums and getting sound advice from others. It all started after playing many hours of Domination, and then getting fed up of waiting for new versions to come out.

Being all enthusiastic I decide to make my own game from scratch. Realising that it wasn't a straight forward process, I started to understand why any Domination update took so long…A shit load of work is required. However, I battled on and made a whole game mode (Self Advertising – Arma2+OA http://www.armaholic.com/page.php?id=16835) just so that I could understand what was going on under the hood. Although at the time I was quite pleased with myself, I must stress this took ages, and I don't mean weeks, it took at least 6 months to get a sound, none major buggy, version up and running. Luckily my wife didn't mind ☺ either.

Back to why I made this. This really is just as an update to some of the tutorials out there for people willing to learn and who want to contribute to the community. This is not an advance tutorial, this is just to get help to those who have no computer programming knowledge, and just want to learn to script.

Chapter on is fairly wordy, but I have tried to make them as non-computer jargon as possible. People with scripting knowledge really wouldn't need to read this, but any suggestions for improvements from others are welcomed. Some terminology won't exactly fit either but I have tried to make this as non-complicated as possible.

If this manages to help at least one person then I'll be happy.

*Massive thanks for all the positive feedback I received for chapter one – really appreciated – I will place all chapters together on completion, and when correctly edited, I will release one big tutorial.

Regards

Mikie J [FOCK] – BIS forums Mikie boy

*Not – do not copy over the text – type it yourself as the " " are different in the editors.

CHAPTER 3 – EventHandlers, SetVariables/Getvariables + Examples

Part 1 – EventHandlers (and scope)

In the last chapter we finished with the creation of a patrol script, which worked ok but it wasn't anything too exciting. We will revisit that and a few other game friendly examples later on in chapter four. For now we are going to cover Eventhandlers.

Eventhandlers can best be thought of as a script/function/code being executed in response to an event. Event in the world of Arma can be thought of as one of many things. For example… getting into a car, getting out of a car, getting killed or respawning, clicking a button on the graphical user interface (GUI), or various other similar events. Basically as the Wiki states, Eventhandlers… allows you to automatically monitor and then execute custom code upon particular events being triggered.

SO… "In the event shit happens automatically do something"

AddEventHanlder - http://community.bistudio.com/wiki/addEventHandler - check the web link to the wiki. Not a lot of info here to help you understand at first glance, which is fine, but as we move along it does actually become clear.

I'd like to invite you at this point to go back to the lobby (of the create game page) in your arma game and click on <<New – Editor>> and click play. We are going to make a new mission to which will be named FockTest2. On clicking play you will go to the editor screen, before you do anything else click the save button and you will be prompted to enter a name – simply type FockTest2. I have then double clicked on the map (for me around the airport as its flat) and created a player. Please create a player now!!! Then click preview.

Once in the mission you should be back to normal running around. Throw two grenades on the floor and walk up to the – Yep starting this chapter with a bit of healthy suicide ☺

Boom! Your dead and the mission failed screen appears. Awesome or not!!

This is where one use for eventhandlers comes in handy – for example being able to work out who killed you – or you didn't realise you had thrown a grenade and walked into it – or even someone else lobbed a grenade and you walked into it – in any event it would be handy to know which mofo is to receive a whirlwind of abuse from you lol.

So that's where a sound example of an eventhandler can be implemented. On the wiki page you will see the code…

this addEventHandler ["killed", "hint format['Killed by %1',_this select 1]"];

Copy and paste this into the init box of your player. Click ok and then press preview. Commit suicide again and then see what it says on the top right hint of your screen.

Killed by B Alpha 1-1:1 (Mikie_J)

Yep, it has shown that I was killed by ME.

So breaking down the code by reviewing the wiki…

object **addEventHandler** [type, command]

Object – well that's us the player – and because it's in the init box of the player we are using the magic 'this' to represent us the object.

addeventHandler – Command word to initiate the eventhandler.

Type – "string" eventhandler type – there are predefined names for the eventhandlers, these go in here and must be in strings.

Command - code that should be executed once the event occurs. You can enter code or a string here.

So looking at what was inserted.

this addEventHandler ["killed", "hint format['Killed by %1',_this select 1]"];

Object – magic 'this' to represent us the object.

addeventHandler – Command word to initiate the eventhandler.

Type – "killed" eventhandler type – there are predefined names for the eventhandlers, these go in here and must be in strings.

Command – code or string - "hint format ['Killed by %1', _this select 1]"

The command code is within a string "hint… and the ends with…]" that's the bit where we are hinting who killed us. To know exactly how the Type "Killed" works you need to look at the eventhandler list (http://community.bistudio.com/wiki/ArmA_2:_Event_Handlers). This is a list of Types that can be used with the addeventhandler command.

If you scroll down to Killed (2.18) – NOT MPKILLED – it shows the following…

Triggered when the unit is killed.

Local.

Passed array: **[unit, killer]**

- unit: Object - Object the event handler is assigned to

- killer: Object - Object that killed the unit
  Contains the unit itself in case of collisions.

By now you should have some idea of what this is saying. In any event I'll explain it the best I can.

This is the code that makes the addeventhandler "Killed".

Triggered when the unit is killed. – well that's fairly obvious – when we die this eventhandler automatically activates.

Local. – this means it's a local command to the computer to which the eventhandler is assigned to.

Passed array: **[unit, killer] = [_this select 0, _this select 1]**

- unit: Object - Object the event handler is assigned to
- killer: Object - Object that killed the unit
  Contains the unit itself in case of collisions.

- It's an array, and in the First position (select 0) is the unit or object that the eventhandler is attached to – in our case it was the magic work 'this' which for our example represents us.

In the Second position (select 1) is the object or unit that killed the object in the 1st position.

So going back to our line of code in the player's init we can see how this is set up…

```
this addEventHandler ["killed", "hint format['Killed by %1', _this select 1]"];
```

In relation to the command/string part "hint format… this is requesting '_this select 1' to be hinted on our screen. The magic '_this' is directing the computer to the array that comes with "Killed" – [unit, killer] – so '_this select 1' means the killer part of the array – that is the person who basically killed us! All the clever coding of how the computer calculates this is done in the game engine and should not concern us. Thus the end result is the hint displaying who killed us.
The reason 'Killed by %1' is encased with single (' ') quotes is that generally you will need to hint using strings, it's just a way to distinguish between the code strings that are required ("hint…]") to be wrapped round the command stuff. You could have used "hint format[""Killed by %1"",_this select 1]"] it should work just the same. Double speech marks instead of single are required, suppose it's an easier way.

Just for good measure ill mention the following… The wiki states "…You may add as many event handlers of any type as you like to every unit, if you add an event handler of type "killed" and there already exists one, the old one doesn't get overwritten. Use removeEventHandler to delete event handlers." So the command removeEventHandler must be used to remove the first eventhandler.

Now the principles detailed above can be used with various other eventhanlders that are shown in this list. Let's add another eventhandler to the game….

So back to the editor, and add any land vehicle – let's say a hunter. In that vehicles init add the following…

this addEventHandler ["GetIn", "if (player == (_this select 2)) then {hint 'Player entered vehicle';}"];

Click ok and then preview. Once in game, walk up to the vehicle and get in the vehicle. On doing so you should get the hint displayed 'Player entered vehicle'.

So what happened? Dumb question I known but breaking it down whilst looking at the wiki…

Triggered when a unit enters the object (only useful when the event handler is assigned to a vehicle). It does not trigger upon a change of positions within the same vehicle. It also is not triggered by the moveInX commands.

Global.

Passed array: **[vehicle, position, unit]**

- vehicle: Object - Vehicle the event handler is assigned to
- position: String - Can be either "driver", "gunner", "commander" or "cargo"
- unit: Object - Unit that entered the vehicle

This should now be fairly self-explanatory. Quick look at it indicates that the Type "GetIn" has an array attached to it – Array is called _this. Thus _this select 0 (select first position from '_this' array) is the vehicle object the eventhandler is attached to, _this select 1 is the position (must be a string) within the vehicle that has the attached eventhandler, and finally _this select 2 is the unit that entered the vehicle with the attached eventhandler.

One thing different of note is the global label attached to this Type. In this case Global represents "…the effects of the given scripting command **are** broadcasted over the network and happen on every computer in the network."

So with all that in mind, going back to our line of code…

this addEventHandler ["GetIn", "if (player == (_this select 2)) then {hint 'Player entered vehicle';}"];

In English this states, if a player gets into the vehicle with the addeventhandler assigned to it, if he is the player that entered the vehicle then hint just for him. The condition of using 'if' stops the hint being transmitted across the network and stops it hinting for everyone. That's because no one else is the player that entered the vehicle. For example….

this addEventHandler ["GetIn", "hint ""Player entered vehicle"""];

Place this code in the vehicle's init – (in place of your other code) within the editor and see what happens in the game. Nothing!!!! the same hint appears. The difference is that if someone else was playing with you, they too would see that hint. There is no condition to stop the hint from appearing anywhere else.

Now, go back to the editor and empty the init box of the vehicle. Inside the init box type the following…

veh = [this] execVM “VehicleEVHandlers.sqf”; (don’t copy and paste either – I just forgot and the friging strings are different remember – DOH!!!!)

Click save, and then alt-tab into the FockTest2 missions folder. Create a new sqf file in the mission folder called VehicleEVHandlers.sqf. Inside that add the following…

```
_vehicle = _this select 0;
_vehicle addEventHandler ["GetIn", "if (player == (_this select 2)) then {hint 'Driving Licence Engaged';}"];
```

Hopefully you can see what I have done here. By placing [this] execVM “VehicleEVHandlers.sqf” inside our vehicle, on the game initiation it will call our VehicleEVHandlers script, and pass [this] as itself to the VehicleEVHandlers script. Inside that script we have the same eventhandler of get in Type. Remember that the magic _this is used to resemble the vehicle that is calling the script. Thus _this select 0 is in fact our vehicle in game that is passing the code to the script. The ‘GetIn’ Type requires an object to be used to assign it to. Here we use _vehicle, which we have assigned previously in the line above the eventhandler. Thus once again, _this select 0 is our vehicle in game and now we have attached an eventhandler ‘GetIn’ to it via the script method.

Save the script and alt-tab back into the game editor. Preview it and see what happens when you get in to the vehicle. Yep, same old hint. Why bother with the script? Why don’t we just copy and paste the code into the vehicles init box? Well if you have more than once vehicle you can use this code multiple times. Exit back to the editor and create another vehicle – let’s use an ATV. Type the same code into its init box…

```
veh = [this] execVM “VehicleEVHandlers.sqf”;
```

Save the game and preview.

You can see that this code can be used continually by all vehicles, and when you have several on the map you can see the benefit and ease of use. In the line below I have amended the script to use hint format to show who actually got into the vehicle. You can use this at any point, by removing/amending the ‘if condition’, to show to others who got into the vehicle.

```
_vehicle addEventHandler ["GetIn", "if (player == (_this select 2)) then {hint format ['%1 entered Vehicle', name player];}"];
```

Let's add another eventhandler, this time using the GetOut type.

Alt-tab out of the editor and back to you VehicleEVHandlers.sqf and add another line…

```
_vehicle addEventHandler ["GetOut", "if (player == (_this select 2)) then {hint 'Player Left Vehicle';}"];
```

Save the file and look at the wiki for eventhandlers.

Wiki states…

Triggered when a unit gets out from the object, works the same way as GetIn.
Global.
Passed array: **[vehicle, position, unit]**

- vehicle: Object - Vehicle the event handler is assigned to
- position: String - Can be either "driver", "gunner", "commander" or "cargo"
- unit: Object - Unit that leaved the vehicle

This is the same as the 'GetIn' type, only difference is the _this select 2 – now relates to the unit that left the vehicle that the eventhandler was attached to.

So now when we run the game we should get a hint that appears when we get in the vehicle and get out of the vehicle. Fairly pointless at this stage, but it can be implemented with thought for other things at a later date. Right so let's have a laugh for five minutes using what we have learnt so far.

We know now that an automatic response occurs when the eventhandler is activated – in our case it's our VehicleEVHandlers.sqf. So let's play around with it and see what we see can do.

First let's create within our mission folder the standard init.sqf. Inside that lets execute our functions file to be preloaded.

```
Execvm "Functions.sqf";
```

Now save the init.sqf and go into your VehicleEVHandlers.sqf and comment out using // the following line…

```
// _vehicle addEventHandler ["GetOut", "if (player == (_this select 2)) then {hint 'Player Left Vehicle';}"];
```

This makes this line in the eyes of the computer as non-readable (still takes up space in memory but does not get executed)

Then add code to our 'GetIn' Eventhandler as follows…

```
_vehicle addEventHandler ["GetIn", "if (player == (_this select 2)) then {[_this select 2, _this select 0] spawn Fock_VehEject; hint format ['%1 entered Vehicle', name player];}"];
```

Addition code we added - [_this select 2, _this select 0] spawn Fock_VehEject;
This is our call to the function we will make below – Remember _this select 2 is the player that entered the vehicle and _this select 0 is the vehicle the player entered. We are now going to pass these two objects to our ejecting function.

Now create a new sqf file called Functions. Let's create a function that ejects our player from any moving vehicle when he hits a certain speed.

Let's call this Fock_VehEject. In the functions folder type the following.

```
Fock_VehEject = {
                        hint "Ejector Seat Script running";
                        _EjectPly = _this select 0;
                        _VehEject = _this select 1;
                        _driver= assignedDriver _VehEject;
                        while {_EjectPly == _driver} do
                        {
                         waituntil {speed _VehEject > 40};
                         _EjectPly action ["eject", _VehEject];
                         Hint "You have been ejected!";
                        };
                };
```

Save the script and the alt-tab back to the editor. Click preview and then jump into one of your vehicles with the attached eventhandler. You should get the hint on the screen that the ejector script is running. Now start driving and see what happens. LOL, childish ejector script complete.

Breaking down the ejector function – fairly simple…

```
    Player     ,    vehicle in
[_this select 2, _this select 0] spawn Fock_VehEject;
```

First we passed the two objects, player and vehicle, to the function.

```
Fock_VehEject = {
                        hint "Ejector Seat Script running"; //debugging
                        _EjectPly = _this select 0; - passed player
                        _VehEject = _this select 1; - passed vehicle player got in
                        _driver= assignedDriver _VehEject; - returns the assigned driver of vehicle
                        while {_EjectPly == _driver} do
                        {
                         waituntil {speed _VehEject > 40}; - waituntil vehicle speed is greater than 40
                         _EjectPly action ["eject", _VehEject]; - at this point eject player from vehicle
                         Hint "You have been ejected!";
                        };
                };
```

We then created the function as normal (no underscore in the name). We then set what the function accepts….

```
_EjectPly = _this select 0; - passed player
_VehEject = _this select 1; - passed vehicle player got in
```

Therefore the _EjectPly = player and _VehEject = Vehicle player got in. We then used the following line…

_driver = assignedDriver _VehEject; - returns the assigned driver of vehicle. Whoever is the driver of the vehicle assign him to the local variable _driver. (http://community.bistudio.com/wiki/assignedDriver) which states… "Returns the soldier assigned to the given vehicle as a driver." Basically we are asking the computer to let us know who is the driver of the vehicle that the player got into. We do this now before the while loop so that we can then check to see if at any point the player is no longer the driver. If the player is no longer the driver then the while loop will end.

while {_EjectPly == _driver} do - As stated, while {player is equal to the assigned driver} do. So while our player is in the driver position continue with the while loop.

```
{
 waituntil {speed _VehEject > 40};
_EjectPly action ["eject", _VehEject];
 Hint "You have been ejected!";
};
```

So you can see that we have entered the while loop and we have a waituntil command. This states wait until the speed (http://community.bistudio.com/wiki/speed) - Object speed (in km/h) – is greater than 40 km/h.

When the speed gets passed 40 km/h then the code moves onto the next line. This is where we look at the action command (http://community.bistudio.com/wiki/action). Makes a unit to perform an action. Here we have selected eject like the example show. The action command has various actions that it can call upon. Look through them at your own leisure and mess around with them.

In any case – we have written…

```
player action["eject", vehicle player got in];
```

Thus asking the computer to eject us out of the vehicle when the vehicle gets to 40 km/h. Then we get a simple hint telling us that. You will notice that the hint is called a few times in a split second and that's why you here that pinging sound. It's the sound of the hint repeatedly being called. This is because the while loop is going around very quick before the eject takes place. Two ways around this…First the trusty old sleep method. A simple sleep 0.05; will do the trick. Secondly is where I talk further about 'scope' and the command exitwith (http://community.bistudio.com/wiki/exitwith).

The scope as you should recall is the { code } between the curly brackets. Now in our example there is a while loop continually checking the speed. We get the hint because as soon as the speed hits > 40 the code below executes once, then twice as we are still going > 40. Thus we need to end it on the first time around. Although this is not the greatest use of exitwith, it does explain a huge misconception of its use. For example, you will often see people typing the following…

```
If {isserver} then exitwith {};

_unit createvehicle blah blah….
```

Although isserver is to do with multiplayer – people assume that if the server calls a script, this bit of code will chuck it right out of the script and all players that call the script, that are not the server, get to execute the rest of the code (the create vehicle bit). Well this is wrong. All the exitwith is exit from the current scope/loop. As there isn't any scope in the above example it will simply jump onto the next bit, so regardless if it's a server or not, everyone will execute the createvehicle bit… The wiki states…If the result of condition is true, code is evaluated, and current block with result of code. It only exits the

current scope…. It then goes onto say…When you use exitWith not inside a loop, the behaviour is undefined - sometimes it may exit the script, sometimes some other scope, but this is not intended and designed behaviour of this command, and it is not guaranteed to work reliably.

It exits the loop only, not the script.

So there you have it only use it to exit a loop not a script. And to give an example of this we will upgrade our eject script using this method. Firstly let's get rid of the repeat hint, amend your script so it looks like the following…

```
Fock_VehEject = {
                        hint "Ejector Seat Script running";
                        _EjectPly = _this select 0;
                        _VehEject = _this select 1;
                        _driver= assignedDriver _VehEject;
                        while {(_EjectPly == _driver) && (Alive _EjectPly)} do
                        {
                        waituntil {speed _VehEject > 40};
                                if (speed _veheject > 40) exitWith
                                        {
                                         _EjectPly action ["eject", _VehEject];
                                         Hint "You have been ejected!";
                                        };
                        };
                };
```

Save it, alt-tab back to editor and preview game. Jump into car and see what happens. Yep one hint and your ass gets kicked out of the car. If the above doesn't show you how exiting the scope works, change the position of the hint. Amend your script to show the following changes…

```
                        {//scope to exit from
                          waituntil {speed _VehEject > 40};
                                if (speed _veheject > 40) exitWith
                                        {
                                         _EjectPly action ["eject", _VehEject];
                                        };
                          Hint "this should not be seen";
                          sleep 2; //to slow down the hint to appear
                          };
                Hint "You have been ejected!";
```

Save and preview once again. Same thing happens – ejected from the car and "You have been ejected" hint shows. You don't see the "this should not be seen" hint. Why, because we exited the loop before we could get to that and jumped out of the brackets and onto the next bit of code. In many ways it's a process of skipping anything inside that loop.

Tada – you can pretty much do anything with this now.

Further example - create a car bomb if it goes over a certain speed. ☺ I know more examples….

Underneath the Fock_VehEject function, let's make a new function called Fock_CarBomb. Copy the below function into your functions.sqf.

```
Fock_CarBomb = {
	hint "Car Bomb Script running";
	_EjectPly = _this select 0;
	_VehEject = _this select 1;
	_driver= assignedDriver _VehEject;
	while {(_EjectPly == _driver))} do
	{
	 waituntil {speed _VehEject > 40};
		if (speed _veheject > 40) exitWith
		{
			"M_Mo_82mm_AT" createvehicle getpos _VehEject;
		};
		hint "this should not be seen";
		sleep 2;
	};
	 Hint "Warning Car bomb has detonated!";
};
```

Save the functions.sqf file and open up your VehicleEVHandlers.sqf. In that file amend the 'GetIn' eventhandler to show the following… (Amendments in red).

```
_vehicle addEventHandler ["GetIn", "if (player == (_this select 2)) then {[_this select 2, _this select 0] spawn Fock_CarBomb; hint format ['%1 entered Vehicle', name player];}"];
```

This time we are calling the car bomb function when the player gets in. Save the VehicleEVHandlers.sqf and alt-tab back to the game and preview. Jump in one of the vehicles and then see what happens. Yep a smallish explosion occurs. Depending on what vehicle you are in depends on the damage done. If you haven't died from the explosion, you will notice the script only runs once, this is because it exited the scope on hitting the correct speed. Thus you will have to get out of the vehicle and back in again to ensure it goes off for a second time. Yep BOOOOM! Second childish function created.

Hopefully you should have understood this line…
`"M_Mo_82mm_AT" createvehicle getpos _VehEject;` - simply create a bomb of stated type at the coordinates of the vehicle we were in. Awesome.

Right I am going to quickly touch upon removal of such eventhandlers - once they have been added….
One thing extra of note when adding event handlers, as mentioned previously "… You may add as many event handlers of any type as you like to every unit…" but when you do you must understand that on every addition of eventhandlers to object, that event handler gets an index added to it, starting at Zero (0) and incrementing as each new eventhandler is added.

For example…

```
_vehicle addEventHandler ["GetIn", "if (player == (_this select 2)) then {[_this select 2, _this select 0] spawn Fock_VehEject; hint format ['%1 entered Vehicle', name player];}"];
_vehicle addEventHandler ["GetOut", "if (player == (_this select 2)) then {hint 'Player Left Vehicle';}"];
```

The ‘GetIn’ eventhandler is at index Zero (‘0’), and the ‘GetOut’ eventhandler is at index one (‘1’). This becomes relevant for the command removeEventHandler (http://community.bistudio.com/wiki/removeEventHandler). The command removes eventhandlers that were added with the command addEventHandler. When removing the eventhandler the index is also removed along with it.

```
_vehicle addEventHandler ["GetIn", "if (player == (_this select 2)) then {[_this select 2, _this select 0] spawn Fock_VehEject; hint format ['%1 entered Vehicle', name player];}"];
_vehicle addEventHandler ["GetOut", "if (player == (_this select 2)) then {hint 'Player Left Vehicle';}"];
Sleep 2;
_vehicle removeEventHandler ["GetIn ", 0];
```

In the above example, we use the command to remove the ‘GetIn’ eventhandler. You notice that after declaring what eventhandler type we want to get rid of, I have placed a Zero (0) after it. This directs the command to remove the ‘GetIn’ eventhandler with index of Zero (‘0’). On the completion of this, the ‘GetOut’ eventhandler remains at position one (‘1’) even though there are no other eventhandlers before it. Now you may be wondering why that is necessary. Well you can add several eventhandlers of the same type to an object…

```
this addEventHandler ["killed", "hint format['Killed by %1',_this select 1]"];
this addEventHandler ["killed", "hint ‘you suck at this’"];
this addEventHandler ["killed", {_this execVM "playerKilled.sqf"}];
this addEventHandler ["killed", {_this execVM "annoyingDeathMusic.sqf"}];
```

So if we have four (4) eventhandlers placed on a player and we removed the third one down (position 2) (the execVM “annoyingDeathMusic.sqf” eventhandler) via the removeEventHandler command, then the last eventhandler (position 3) will stay at position three (3) and there will be no eventhandler at position Two (2). Thus there is only index 0, 1, and 3 left. Hopefully you can see now why you need the index to not only remove the correct eventhandler type, but the correct one of that type. The best method to remember the eventhandlers is to give them a handle variable…

```
_ev0 = this addEventHandler ["killed", "hint format['Killed by %1',_this select 1]"];
_ev1 = this addEventHandler ["killed", "hint ‘you suck at this’"];
_ev2 = this addEventHandler ["killed", {_this execVM "playerKilled.sqf"}];
_ev3 = this addEventHandler ["killed", {_this execVM "annoyingDeathMusic.sqf"}];
```

That way you can simply remove the eventhandler via the assigned variable.

```
This removeEventHandler [“Killed”, _ev1];
```

Removed the second eventhandler down…

```
_ev1 = this addEventHandler ["killed", "hint ‘you suck at this’"];
```

And finally…

removeAllEventHandlers(http://community.bistudio.com/wiki/removeAllEventHandlers). "Removes all event handlers of given type that were added by addEventHandler."

No index is required for this – simply removes all the eventhandlers attached to the specified object.

That my friend(s) was a moderate introduction to Event Handlers. These can be used for various useful things, like 'respawn' and 'killed' event handlers for a revive script, or 'GetIn' for a rank/type of soldier allowed in vehicles script - We will do this in Part 2 ☺

Part 2 – SetVariable and GetVariable

SetVariable (http://community.bistudio.com/wiki/setVariable) Wiki states…Set variable to given value in the variable space of given element. Yep that's fairly straight forward…WTF? My thoughts too. Well I will explain to you myself but first have a read of this…(http://forums.bistudio.com/showthread.php?130487-SetVariable).
See if you can get it from that.

The reason I have shown you this is twofold. One – it's a fairly good example and two an introduction to the BIS forums. I've not mentioned them before in depth as I didn't want anyone asking dumb ass questions on there, but at this point you should now have a sound understanding of how things work. These forums are a brilliant example of good work and community spirit (at times) and you should all register to get involved. I was only fecking around with the dumb ass questions, but you should all search thoroughly before posting (admins I have done my part). If at all the wiki (in most cases) doesn't provide enough info, then searching here is a must. Don't just search in the Arma 3 forums, search in the Arma 2 forums as well. The code (sqf) is the same and Arma 3 code is built on top of the Arma 2 code with new commands having been introduced. In addition to that – those who have downloaded this tutorial either from fockers website or Armaholic then you can ask me questions in this thread (http://forums.bistudio.com/showthread.php?154047-Arma-3-Scripting-Tutorial-For-Noobs&highlight=scripting+guide).

So, just in case you didn't understand what was said regarding setvariable ill progress the topic.

```
_unit = player;
```

From this we all know that we have assigned the player object to the local variable _unit. So every time we use the variable _unit we refer to the player object.

```
_teamScore = 0;
_teamScore = _teamScore + 10;
```

We should also know that in the above example we have assigned a value to _teamScore on two occasions. First we gave it a value of Zero ('0'). Then we said _teamScore should now be the value of _teamScore plus Ten (10). Thus giving us a total of Ten (10). _teamscore now represents the value Ten (10)… (don't worry I am going somewhere with this)…

What happens if you wanted to give a player a value. That's right not a variable, but an object. It would be like saying…

Player = 10;

We should know that we can't give an object a value (as shown above) as this would produce an error and more so that doesn't make any frigging sense. And for those of you that just tried it – DOH! ! But let's say you wanted to give a variable to an object and then give that variable a value, then that is possible. Now I'm sure some of you are confused at this point by this concept, so think of it this way...

Object = variable = value

Say we wanted our player Dave to have points added to him each time he drives a vehicle. Let's say these points are held on his driving licence for the time he is in game. So to do this we would first want to give Dave a driving licence; we do this by way of the command setVariable...

```
Dave setVariable ["drivingLicence", value];
```

Dave is the object, the command is setvariable, and the variable we are assigning to Dave is "drivingLicence". This variable could be anything, but it must be encased within strings. "drivingLicence " could be thought of as _drivingLicence, which you are then going to assign a value to.

The value is then added by way of 'any type' – meaning you could enter a string, bool, or int. In our case we want to add points to Dave's driving licence, so we will be using integers.

```
Dave setVariable ["drivingLicence", 0];
```

So now Dave's variable "drivingLicence" holds the value of Zero (0). You should see that we have set a variable to an object and then given that variable a value. You could call our player Dave – and every time you wanted to check the status of his name variable, it would return Dave…

```
player setVariable ["Name", "Dave"];
```

So back to Dave's Driving licence…If you asked Dave at this point how many points he had on his licence he would say Zero (0). You would do this by way of a variable that would hold the return value of getting the variable (handle).

```
_howmanyPointsOnYourLicence = Dave getVariable “drivingLicence”;
```

Therefore _howmanyPointsOnYourLicence would be equal to Zero (0). So really it would look like this…

```
_howmanyPointsOnYourLicence = 0;
```

So now Dave has Zero (0) “drivingLicence” points. Now we must give Dave points for driving around.

Let’s create a real script using our ‘GetIn’ eventhandlers. Go to your VehicleEVHandlers.sqf, comment out the lines currently in the file (// before each line) and add the following…

```
_vehicle = _this select 0;
_vehicle addEventHandler ["GetIn", "if (player == (_this select 2)) then {[(_this select
0),(_this select 1),(_this select 2)] spawn Fock_DLicence; hint 'Driving Licence Engaged';}"];
```

Save that file and then add the following function to your Functions.sqf…

```
Fock_pVehSpeed = {
                        _veh = _this select 0;
                        _vSpeed = speed _veh;
                        _vSpeed
                       };
Fock_Dlicence = {
                _unit = _this select 0; //Vehicle the event handler is assigned to
                _pos = _this select 1; //seating position
                _unitdriver = _this select 2; //Unit that entered the vehicle
                _TotalDrive = 0;
                _isDriver = assignedDriver _unit;
          while {((alive _unit) and (_unitdriver in _unit) and (_isDriver == _unitdriver))} do
                {
                        waituntil {isplayer _unitdriver};
                        _playerspeed = [_unit] call Fock_pVehSpeed;
                        while {(_playerspeed > 0)} do
                        {
                         sleep 2;
                        _TotalDrive = _TotalDrive + 1;
                         hint format ["Driving Points: + %1", _TotalDrive];
                        _playerspeed = [_unit] call Fock_pVehSpeed;
                    };
                   sleep 1.5;
                };
```

```
        _pDrivLicence = _unitdriver getVariable "DLicence";
        _unitdriver setVariable ["DLicence", _pDrivLicence + _TotalDrive,true];
        _pDrivLicence = _unitdriver getVariable "DLicence";
        hint format ["Driving licence Points: %1", _pDrivLicence];
};
```

And finally in your init.sqf add this line…

```
player setvariable ["DLicence ", 0];
```

Save all the files. Alt-Tab to the editor and press preview. Jump into your vehicle and see what happens …

And hopefully when you drive around….

And when you disembark…

Tada, we have a usable driving licence system. If you have messed around in game and got back into the vehicle and driven around some more, when you get out you will see that your score has increased from the first time you disembarked.

Righty, let's explain the code/functions…

```
_vehicle addEventHandler ["GetIn", "if (player == (_this select 2)) then {[(_this select 0),(_this select 1),(_this select 2)] spawn Fock_DLicence; hint 'Driving Licence Engaged';}"];
```

The above should be reasonably understandable. All we have done here is checked to see if the player is equal to the (_this select 2) – the unit that entered the vehicle. If the player is the unit that entered the vehicles, then pass all the 'GetIn' eventhandler variables (vehicle, position, unit) (http://community.bistudio.com/wiki/ArmA_2:_Event_Handlers#GetIn) to our function Fock_DLicence. Thus the Fock_DLicence should receive what the vehicle is that has activated the eventhandler, who is in what position inside the vehicle, and what unit has entered the vehicle. We then have added a hint to show the driving licence is working. At the point of the player entering the vehicle the Fock_DLicence is activated…

So, Fock_DLicence, let's look at that code.

```
_unit = _this select 0; //Vehicle the event handler is assigned to
_pos = _this select 1; //seating position
_unitdriver = _this select 2; //Unit that entered the vehicle
_TotalDrive = 0;
_isDriver = assignedDriver _unit;
```

You can see that the three variables passed from the eventhandler (vehicle, position, unit) have been passed to the script and they have been assigned accordingly. _unit is the vehicle we entered, _pos is the position we got into within the vehicle – in this case the driver and _unitdriver is the player that got into the vehicle.

Fairly straight forward I think, and then we have…

```
_TotalDrive = 0;
_isDriver = assignedDriver _unit;
```

Well _TotalDrive = 0 is simply us setting the value of _TotalDrive to Zero (0) every time we get into the vehicle. This variable will count the driving licence points whilst the vehicle is moving. Every time we get into this vehicle it will be set to Zero (0). _isDriver = assignedDriver _unit; assignDriver (http://community.bistudio.com/wiki/assignedDriver) RECAP - Returns the soldier assigned to the given vehicle (_unit) as a driver. No real surprise there as we have seen this previously. So this command simply gives us the player assigned/sat in the driver seat or the vehicle stated. If there is no player in that driver seat then it will be empty and nothing returned. Thus whoever is sat in the driver position will be assigned to _isDriver. In this case it should be _isDriver = player;

Next…

```
while {((alive _unit) and (_unitdriver in _unit) and (_isDriver == _unitdriver))} do
{
        waituntil {isplayer _unitdriver};
```

```
        _playerspeed = [_unit] call Fock_pVehSpeed;
        while {(_playerspeed > 0)} do
          {
                sleep 2;
                _TotalDrive = _TotalDrive + 1;
                hint format ["Driving Points: + %1", _TotalDrive];
                _playerspeed = [_unit] call Fock_pVehSpeed;
          };
   sleep 1.5;
};
```

(_unitdriver in _unit)… 'In' in this case refers to vehicles - Checks whether the soldier is mounted in the vehicle. In (http://community.bistudio.com/wiki/in_vehicle).

Starting with the While loop – we are saying (while the vehicle we got into is alive) and (while the player that got into the vehicle is still in the vehicle) and (while the assigned driver is still the player) then…

```
waituntil {isplayer _unitdriver};
```

We instantly ask the computer to pause until isplayer _unitdriver. isplayer (http://community.bistudio.com/wiki/isPlayer) checks if given person is a human player. Thus, waituntil {human player is the unit that got into the vehicle}. Once the computer establishes that the player who got into the vehicle is a human player then…

```
_playerspeed = [_unit] call Fock_pVehSpeed;
```

Handle _playerspeed holds the return value from our call to function Fock_pVehSpeed. This little function simply returns the speed at which the vehicle the player is in is travelling at. So the handle _playerspeed resembles the speed at which the vehicle is travelling. Whatever speed the function Fock_pVehSpeed returns, the handle variable _playerspeed will hold. Thus…

```
while {(_playerspeed > 0)} do
```

While the speed of the vehicle (returned from Fock_pVehSpeed) is greater than Zero (0) do...
We have now entered the next while loop. If the vehicle is not moving, and the _playerspeed is not greater than Zero (0) then we would skip this while loop and got back to the start of the
"while {((alive _unit) and (_unitdriver in _unit) and (_isDriver == _unitdriver))} do" Loop. This is why we have sleep 1.5; at the end. If the speed is not greater than Zero (0) then we

sleep momentarily for 1.5 seconds and then go through the loop again to see if the speed is greater than Zero (0).

Let's say we are moving forward and the speed is increasing. Now we enter the next while loop successfully and will remain in this while loop until the speed of the vehicle is Zero (0). The following things occur whilst inside that while loop…

```
sleep 2;
_TotalDrive = _TotalDrive + 1;
hint format ["Driving Points: + %1", _TotalDrive];
_playerspeed = [_unit] call Fock_pVehSpeed;
```

Firstly we pause for Two (2) seconds, this is so that every two seconds the driving points can be added – thus for every Two (2) seconds you are driving in the vehicle you will get a point – well that's the idea.
And we give the points by adding one (1) to _TotalDrive variable. Remember that I said every time you get into the vehicle, _TotalDrive is cleared to Zero (0). It will therefore start at Zero(0) and increment every two (2) seconds whilst the speed of the vehicle is greater than Zero (0).

```
_TotalDrive = _TotalDrive + 1;
```

Whatever the current score of _TotalDrive, add one (1) to it. We then hint the current sum of points earned whilst driving via hint format.

```
hint format ["Driving Points: + %1", _TotalDrive];
```

This is the hint you see on the screen to see how many point you have gained whilst driving (not your total for the length of the game – just for this journey).

```
_playerspeed = [_unit] call Fock_pVehSpeed;
```

We then call to our function Fock_pVehSpeed again to check the speed of the vehicle. This allows the while loop to keep checking our score. If the speed goes to Zero (0) then the 'while loop' will know that we have gone to Zero (0) and will exit that loop. If not we go back around to the start of the 'while loop', wait two (2) seconds and then add one (1) to _TotalDrive, and so on. This will go whilst the player is in the driver seat of the said vehicle and the speed is greater than Zero (0).

Once the speed of the vehicle goes to Zero (0) and we are no longer in the driver position, and no longer in the vehicle with the eventhandler, we exit the outer 'while loop'. Moving onto the code outside of the scope of both while loops.

```
_pDrivLicence = _unitdriver getVariable "DLicence";
_unitdriver setVariable ["DLicence", _pDrivLicence + _TotalDrive,true];
_pDrivLicence = _unitdriver getVariable "DLicence";
hint format ["Driving licence Points: %1", _pDrivLicence];
```

As we are no longer in the vehicle we will use the getVariable command to recall our "DLicence" points. Remember that we set it to Zero (0) in the init.sqf, so that every time we joined it was Zero (0)…

```
player setvariable ["DLicence ", 0];
```

We now use the variable _pDrivLicence to hold the returned value from getting "Dlicence" value stored within the _unitdriver. Well we know _unitdiver is in fact the player that entered the vehicle, and that player "Dlicence" has been set to 0, so the returned variable should be Zero (0).

```
_pDrivLicence = 0;
```

We then have…

```
_unitdriver setVariable ["DLicence", _pDrivLicence + _TotalDrive,true];
```

Here we are simply setting the player's "Dlicence" variable to the value of _pDrivLicence (which should be Zero (0) on the first time driving the vehicle) and then adding the total amount of points gained from driving the vehicle, which was stored in the local variable _TotalDrive.

So let' say we got 23 point from driving around. _pDrivLicence therefore is equal to 0, and _TotalDrive is equal to 23. Thus it really looks like this…

```
player setVariable ["DLicence", 0 +  23,true];
```

We are therefore now setting the player's DLicence to 23. So the next time you get into this vehicle and we get to this line…

```
_pDrivLicence = _unitdriver getVariable "DLicence";
```

It should return 23, as this is what we have stored. Now let's say we drove around some more and had obtained Ten (10) points from driving (remember _TotalDrive gets reset to Zero (0) each time, it does not retain the score after the script is run). It should therefore resemble something like…

_pDrivLicence is equal to 23, and _TotalDrive is equal to 10. Thus it really looks like this…

```
player setVariable ["DLicence", 23 +  10, true];
```

To ensure we can see the correct total we have added…

```
_pDrivLicence = _unitdriver getVariable "DLicence";
hint format ["Driving licence Points: %1", _pDrivLicence];
```

In order to show the correct total we request the total value of "Dlicence" once more now that _TotalDrive has been added to it, then simply hint to show our total "Dlicence".

That's it. Every time you are in the driving seat and the vehicle speed is greater than Zero (0) you will get points for it. Some of you are probably thinking "what's the frigging point of that?" – Well there are numerous things you can do with it. For example, when you get to Multi-Player scripting this can be a good way to increase the player's score for doing all the driving. You could give the player who drives vehicles with others in it 'score' points (http://community.bistudio.com/wiki/score) as a reward.

```
Player addscore _pDrivLicence;
```

Or you can use it to boot players out of vehicles if they haven't got enough points to fly/drive that vehicle. Here's an additional script I made way back to limit what vehicles you can get into. I amended it so that it will work with our example and driving licence points.

In your init.sqf add this line…

```
execVM "vehrank.sqf";
```

Save that file. Create a new file called vehrank.sqf and copy the following…

```
waituntil {vehicle player != player};
private ["_corprank", "_sergrank", "_lieurank", "_caprank", "_majrank", "_colrank",
"_corpveh", "_sergveh", "_lieuveh", "_capveh", "_majveh", "_colveh", "_vehchoice"];
//nothing to do with rank - innermost scope

                    //RANK – level to reach to get in vehicles – Dlicence points needed
```

```
_corprank = 10;
_sergrank = 20;
_lieurank = 30;
_caprank = 40;
_majrank = 50;
_colrank = 60;

//If vehicle is not mentioned in the list below then any rank can use them. MAKE SURE YOU COVER OPFOR ANFD BLUEFOR!!!!!
_corpveh=["B_MRAP_01_F","B_MRAP_01_gmg_F","B_MRAP_01_hmg_F","B_Truck_01_transport_F","B_Truck_01_covered_F"];
_sergveh = ["B_APC_Wheeled_01_base_F"];
_lieuveh = ["B_APC_Wheeled_01_cannon_F"];
_capveh = [];
_majveh = [];
_colveh = [];
    while {true} do
                {
                 waituntil {count playableUnits >= 1};
                 waituntil {vehicle player != player};
                 _dLicence = player getvariable "DLicence";
                 _vehchoice = vehicle player;
                        {
                 if (driver _vehchoice == player) then
                         {
                                 if (_vehchoice iskindof _x) then
                                 {
                                         if (_dLicence < _corprank) then
                                                 {

                                                 player action ["ENGINEOFF", _vehchoice];
                                                 player action ["eject", _vehchoice];
                                                 sleep 0.5;
                                                 _vehchoice engineOn false;
                                                 waituntil {vehicle player == player};
                                                 hint format ["you need have achieved at least %1 points on your driving Licence",_corprank];
                                                 };
                                 };
                         };
                        } foreach _corpveh;
        //serg rank
                {
                 if (driver _vehchoice == player) then
                 {
                         if (_vehchoice iskindof _x) then
                         {
```

```
                                        if (_dLicence < _sergrank) then
                                        {
                                                player action ["ENGINEOFF", _vehchoice];
                                                player action ["eject", _vehchoice];
                                                _vehchoice engineOn false;
                                                sleep 0.5;
                                                waituntil {vehicle player == player};
                                                hint format ["you need have achieved at least
%1 points on your driving Licence",_sergrank];
                                        };
                                };
                        };
                } foreach _sergveh;
};
```

**copy serg rank and change as you need – next rank would be _lieurank – which relates to the string array of vehicles in _lieuveh = [“ ”]; do this as many time as you need – just place the vehicle names into the necessary arrays. **

Save the file, alt-tab back to the editor and ensure you have added an ATV vehicle and a standard Hunter vehicle and add the event handler line to their init lines…

```
veh = [this] execVM "VehicleEVHandlers.sqf";
```

Press preview and try getting into the hunter before getting into the ATV…

You get kicked out!!! Jump into the ATV and you should be able to get into that. Once in, drive around until you have 10 points on your “Dlicence”. Once you have Ten (10) points try getting into the Hunter again. This time you should be allowed in. Not bad – handy. I’ve

actually used this for a Database system (Arma2Net, Arma2netMysql) for dedicated servers. It stops people jumping onto your game and taking all the good choppers and vehicles and trashing them. This way people get points and bonuses for helping out and driving/flying people around. A lot of people like to be chopper pilots and don't want to run around, this is a very good team base method to increase such a thing. ☺

The getvariable and setvariable methods are useful for all sorts of things. For example…You can begin to make IEDs yourself that you can set to armed simply using…

```
_ied setVariable ["Status", "1", true];
```

Then you could check when anyone goes near them if the value of them is either Zero (0) or one (1). You could then code… if the getvariable returns one (1) give the player 5 seconds to disarm or to run away. You can see how I created FOCK_IEDs in Arma A2/OA (http://www.armaholic.com/page.php?id=17260#comments) – have a look and see how it is put together, although the setvehicleinit command no longer is acceptable. Please change to BIS_fnc_MP if you are using this for MP.

In any case this is just an idea and something worth considering.

Chapter 3 Conclusion…

You have now learnt how eventhandlers work. You should be able to add eventhandlers, remove individual eventhandlers, and remove all event handlers. You should have an understanding of the construction of the event handlers, and now have enough insight to try other event handlers.

We also touched upon scope and exitwith command. You should understand the concept on different loops and scopes, and how to exit form a specific loop depending on the conditions.

We covered the use of get and set variables. You should have a sound understanding of how you would assign an object a variable and then give that variable a value. You should understand that to get the holding value of an object's variable, you need to provide a handle variable that holds the returned value. You should note that setvariable works with other things and not just objects.

We made several functions so you should have a solid notion of how they are called and used correctly. And from these functions we made several childish scripts.

We briefly touched upon the Action command and the actions available to us. Please have a go at using this command, can be very useful.

And finally we made a driving licence script/function that stops people jumping straight into any vehicle and driving off. This can be used for helicopters, boats, and planes.

Remember the Biki, and the forums. Search well and thoroughly, if all else fails pose a question on the forums with a suitable title, and focus your question to include all necessary code that you are struggling with.

Now crack on and create some more scripts in prep for the next chapter.

Feedback on fockers website, on email at fockersteam@hotmail.co.uk. , or (http://forums.bistudio.com/showthread.php?154047-Arma-3-Scripting-Tutorial-For-Noobs)

Thanks Mike J [FOCK] - http://fockers.moonfruit.com/