Spark55

Script Collection Guide

1 - Installation

All scripts are pre-packed and integrated into the official XP Template. These pre-packed installations are condensed into the folder Spark_Master_X, with the suffix _X being a number referencing the current version of the included Spark55 scripts. At the moment two different pre-packed installations are provided:

a. SparkAI_SparkLift_Template
b. SparkAI_SparkLift_Template_Vcom

Both installations include the SparkAI spawning system, SparkLift system, which is a modified version of R3F logistics and the official XP Template. Both versions have optional scripts that can be activated as desired in a specific mission. These additional scripts include the Spark vehicle respawn system, Spark vehicle protection system and a modified version of BendeRs spectrum device scripts. The second installation file (b. above) contains additionally the VCOM AI behavior system which also has been slightly modified. Be aware that most AI behavioral scripts have a considerable impact on the server and client performance and should be used with care.

You can simply copy and paste the contents of one of the above-mentioned packages into your mission folder to install the desired package.

The included third-party scripts, such as R3F logistics, BendeRs spectrum device scripts and the XP template, are for private use only and are not to be made publicly available without the consent of the respective author. All other scripts are intellectual property of Spark55 and are not to be shared without the consent of the author. Neither shall they be made publicly available without permission by the author.

2 - SparkAI

SparkAI spawning system enables the user to quickly spawn a group of AI units in an attacking or defending role. The AI can be easily spawned by utilizing two custom lines of code that call the relevant spawning functions. These are:

a. Vuk_Fnc_Spawn_Defend
b. Vuk_Fnc_Spawn_Attack

Vuk_Fnc_Spawn_Defend: With this function you can spawn defensive AI that occupy a defined area. The AI will scatter around the area, patrol and crew static weapons. The function is very versatile and can be used on the fly in missions and can be called from every possible environment.

You will need to provide the scripts with the class names of the unit and vehicle types that you want to spawn. The script allows you to create up to 10 pools of different unit and vehicle class names from which the script will read all needed information from.

In order to add the desired units and vehicles you will need to open the file Spawn_SelectArray.sqf inside the folder SparkAI within the mission directory. You will find empty brackets into which you will need to enter the classnames. By default the first pool is filled with vanilla unit class names as placeholders to give you an example about how to enter your desired classes. Each pool is identified by a number that you can find in the comments at the end of each line. The comments will also inform you about which unit types need to go into which brackets.

The classnames must be inserted between the speech marks and need to be separated by a comma.

```
[ "B_Soldier_F", "B_Soldier_F" ],
```

The arrays can be expanded infinitely. Now follows the same bracket with four classnames as an example of expansion.

```
[ "B_Soldier_F", "B_Soldier_F", "B_Soldier_F", "B_Soldier_F" ],
```

There are three different arrays for vehicles. Light, medium and heavy. Fill them accordingly with matching vehicles. The process here is exactly the same as adding classnames to the infantry

array. Although The different categories of vehicles have no correlation to in-game properties and are purely intended to give you more control when spawning vehicles. Note that each unit is randomly selected from the class name pool by an equal distribution. This is a feature meant to give you more control about which type of unit is more common than others. The same class name can be added multiple times to increase its spawning chance.

Once all your units have been defined you can call the spawning function with the following line of code:

```
[ 0, 0, "Marker", 350, [ 5, 5, 5], [ 4, 4, 4 ], [ 2, 2, 2, 2 ], false ] spawn Vuk_Fnc_Spawn_Defend;
```

DO not use this if you wan to use this code from a trigger!

The first number is representing the class name pool used for this instance of spawning.

The second number is representing the side of the spawned units:

0 = West, 1 = East, 2 = independent, 3 = civilian

The next input required will be the variable name of a positional marker, "Empty" type is recommended, which will act as the center of the area that the AI will defend.

The following numerical input defines the size in meters of the area that will be defended. Be aware that choosing too small of a number in an urban environment might lead to complications when spawning vehicles.

The next three numbers define the amount of AI units in one squad of each type:

```
[ … [ Static, Patrolling, Building ] …]  spawn Vuk_Fnc_Spawn_Defend;
```

The the number of static infantry soldiers in a squad, the number of patrolling infantry soldiers in a squad and the number of infantry soldiers per squad that occupy buildings.

The next three numbers define the amount of AI squads total you want to spawn of each type:

```
[ … [ Static squads, Patrolling, Building ] …]  spawn Vuk_Fnc_Spawn_Defend;
```

The number of squads are defined by the number of static infantry squads, the number of patrolling infantry squads and the number of infantry squads that occupy buildings. Be aware that script does not work when there is no building within the defined area.

The following four numbers represent the amount of vehicles for each category of vehicles:

```
[ … [ light, medium, heavy,  air ] …]  spawn Vuk_Fnc_Spawn_Defend;
```

Said vehicles will be patrolling within the defined area. Be aware that fixed wing air vehicles will try to loiter around the area which can lead to complications for the AI pilots. Therefore it is recommended to primarily use helicopters.

The next input controls whether the spawned ground vehicles will patrol or be stationary until they are being engaged. The input can be either true (units will be static) or false (units will patrol).
The function is supposed to be run server side only. Therefore, you can either call it from the initServer.sqf file (any other script file works as well), from the server console as Zeus, or from within the onActivation field of a trigger that is set to Server Only mode.

Note that when using the function from a trigger the function requires a similar yet different line of code that must be used or otherwise some features of the function will not work properly. But lastly both lines of code are customized exactly by the same method. For usage in triggers:

```
call { [ 0, 0, "Marker", 350, [ 5, 5, 5], [ 4, 4, 4 ], [ 2, 2, 2, 2 ], false ] spawn
Vuk_Fnc_Spawn_Defend;};
```

Vuk_Fnc_Spawn_Attack: With this function you can spawn attacking AI that will assault a given position until all enemy units around the position have been killed. Once the position is captured the AI will continue to defend the capture position and will automatically establish patrols and static defensive positions. Everything explained and shown above for Vuk_Fnc_Spawn_Defend applies to Vuk_Fnc_Spawn_Attack as well. The only differences are the slightly altered input requirements:

```
[ 0, 0, "Marker1", "Marker2", 4, [ 5 ], [ 2,2,2,2 ], false ] spawn Vuk_Fnc_Spawn_Attack;
```

The first number is representing the class name pool used for this instance of spawning.

The second number is representing the side of the spawned units:

0 = West, 1 = East, 2 = independent, 3 = civilian

The next input required will be the variable name of a positional marker, "Empty" type is recommended, that will act as the position from which the attack will initiate from.

Followed by the variable name of another positional marker which will be the position that the attackers are assaulting.

The next number defines the amount of AI units in one squad.

The number of attacking squads is defined by the following number that stands alone in the extra set of brackets.

The last four numbers represent the amount of vehicles for each category of vehicles that will be attacking the desired area:

```
[ ... [ light, medium,  heavy,  air ] ...]  spawn Vuk_Fnc_Spawn_Attack;
```

Note that his function also requires a different line of code when being called from a trigger:

```
call { [ 0, 0, "Marker1", "Marker2", 4, [ 5 ], [ 2,2,2,2 ]] spawn Vuk_Fnc_Spawn_Attack; };
```

SparkAI - No backpack patch

This small patch will remove all backpacks from spawned infantry units. This patch is particularly useful when planning to restrict resupply on the players and prevent extensive resupply through battlefield pick-ups.

In order to install this patch simply navigate to the folder Spark_Master_X\SparkAI_Patch and copy the two files into your SparkAI directory located at YourMissionFolder\SparkAI. Replace and overwrite both files if prompted.

3 - SparkLift

SparkLift system, which is a modified version of R3F logistics, enables the players to lift various objects and vehicles with helicopters and enables other additional logistics based systems such as paradrop of vehicles and implements a “towing” system of vehicles or objects.

By default the logistics system is disabled for all objects placed in the editor. If you want to initialize the system on desired objects you will have to add the following of code into the init field of an object or vehicle placed in the editor:

```
this setVariable ["R3F_LOG_disabled", false];
```

Note that both the helicopter and the liftable object need to have this line pasted into their init field in order for the system to work. Initializing the system on ammo boxes or smaller objects will result in them being movable and storable in vehicles. Additional vehicles can be added by their class name in the init.sqf file.

Only specific vehicles are supported by the script. For a full list of vehicles refer to the according section of this guide. However more can be added easily.

Supported vehicles

The following vehicles are supported by the SparkLift logistics system:

All vanilla ArmA 3 and dlc vehicles and objects			
CUP-Helicopters	**CUP-Vehicles**	**RHS-Helicopters**	**RHS-Vehicles**
CH-47F	M113	MH-6M Little Bird	M2A3 (BUSK iii)
CH-53ESuper Stallion	M60A3	UH-60M	M1A2SEPv1 (TUSK ii)
AW159	X-66 Mammoth	CH-47F	BTR-80
MH-47E	LAV-25A1	Mi-8	BTR-80A
MH-60S Knighthawk	LAV-C2	Mi-24	M1230
MH-60S Seahawk	M1130 CV M2		M6 Linebacker

MH-6M MELB	M1128 MGS		Mk. V SOC
UH-1Y Venom	M1167 (TOW-2)		
UH-60M	Mastiff PPV HMG		
Mi-6T	Wolfhound TSV HMG		
Mi-8	M1165 GMV		
Mi-24 Hind	Frigate (With Patch)		

IF you want to add more vehicles that use the sparklift script all you must do is go into your go to your init.sqf folder in your mission file. Using the script, you will see “Private_A to Private_G” you just add the vehicles or objects class name into the “” next to the private name you want to work with. For example add CUP_C_Ikarus_Chernarus into section Private_B allows the vehicle to be towed.

SparkLift - Frigate patch

This small patch will enable helicopters to lift the frigate vessel added by the CUP mods. Note that this patch changes certain margins of the R3F logistics system which will result in the player being able to hook certain vehicles from further away. Intern it allows for big objects like the CUP frigate to be lifted without clipping into the helicopter lifting it. This patch should only be used when it is necessary for the frigate to be lifted.

In order to install this patch simply navigate to the folder Spark_Master_X\SparkLift_Patch and copy the folder R3F_LOG into your mission directory. Replace and overwrite all files if prompted.

4 - Spark Vehicle Respawn

Spark vehicles respawn is a custom system that allows a specific vehicle to respawn with the pylon loadout, the custom skins that were applied in the editor, and the custom inventory in the vehicle. The system is activated by pasting the following line of code into the init line of the vehicle that is intended for respawn:

```
Null = [ this, 60, 30 ] execVM "SparkAdditionals\Spark_RespawnVehicle.sqf";
```

The first input of this line of code is referring to the object that is being respawned. When calling the function from the init line of a vehicle the first input must be this.

The following numerical input refers to the time in seconds until the vehicle is being respawned after its destruction.
The following numerical input refers to the time in seconds until the vehicles destroyed body will disappear.
Important:

- Do not let the green number be lower than the orange number.
- You can add the Sparklift script to the vehicles in it as well and it will allow even the respawn of the initial vehicle to benefit from the Sparklift script.

5 - Spark Vehicle Protection

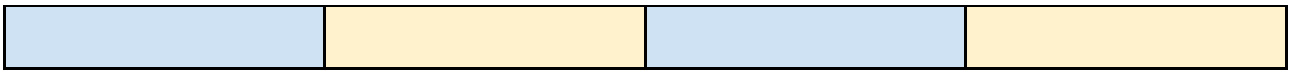
Spark vehicle protection is a custom system that protects a vehicle from taking fatal damage which would lead to its explosion and instead only renders it disabled. The system is being activated by copy and pasting the following line of code into the init.sqf file of your mission. All vehicles to which this script will be applied must be given a variable name in the editor which is needed to reference the vehicle within the line of code.

```
Null = [ V1, V2, V3, V4, V5, V6 ] execVM "SparkAdditionals\Spark_ProtectVehicle.sqf";
```

The inputs are the vehicle variable names that are supposed to be protected from destruction. The amount of vehicles can be reduced or expanded by adding a comma between the variable names. In this particular case 6 vehicles were added for demonstration.

The following vehicles are supported by the Spark vehicle protection system:

CUP-Vehicles	RHS-Vehicles	Vanilla/DLC Vehicles	TMTM
RG31E M2 Olive Drab	M1A2 Tusk 2 Woodland		TMTM LT Racing
RG31 M2 Olive Drab	M1A2 Tusk 2 Desert		TMTM LT Civilian
LSV Unarmed	M6 Linebacker Woodland		
Ikarus Bus Chernarus	M6 Linebacker		
Hilux, BMP1 turret			
Hilux, BTR60 turret			



6 - Spark Spectrum Device

The Spark spectrum device system adds functionality to the spectrum device that was added by the recent “contact” dlc. The set of scripts were originally written by BendeR and were slightly modified for the inclusion in this package. Any object in the game can be made a beacon which emits a signal on a specific frequency that can be picked up with the advice. The system is activated by pasting the following lines of code into your init.sqf file and editing the inputs accordingly:

```
Null = [[    [ B1, 550 ], [ B2, 570 ]    ]]  execVM "SparkAdditionals\Spark_Spectrum_B.sqf";
Null = []  execVM "SparkAdditionals\Spark_Spectrum_A.sqf";
```

You can define a beacon and the frequency of its signal by giving the object that is the beacon a variable name in the editor. This variable name must be then entered into the first line of code.

The frequency of a specific beacon is a numerical input that follows the variable name of the beacon. Both properties are contained in one set of brackets.

The amount of beacons can be expanded indefinitely by copying said brackets and their content, separated by a comma.

Additionally, a hold action can be applied to a specific beacon in order to make it removable once found by the player. Simply copy and paste the following line of code into the initServer.sqf file in order to add this customizable hold action to the beacon:

```
Null = [ VarName, "Title", "Message after completion", 10 ] execVM
"SparkAdditionals\Spark_Spectrum_HoldAction.sqf";
```

The first input is the variable name of the beacon that the action will attach to.

The second input is the title of the action that will appear to the player.

The next input is the message that will be displayed to the player who activated the action.

The last numerical input is the duration that the button needs to be pressed until the beacon is deactivated.

Advanced users can open the files Spark_Spectrum_A and Spark_Spectrum_B respectively, which are both located in the folder SparkAdditionals and fine tune the device settings and ranges. All the necessary equipment to detect active beacons can be added to a playable unit simply by putting the following code into the units init line:

```
this addWeapon "hgun_esd_01_F";
this addHandgunItem "muzzle_antenna_02_f";
this addItemToVest "muzzle_antenna_03_f";
```