Dynamic-AI-Creator [DAC] Version 2.0 Beta



ArmA 1.08 required / addons : Addons: EditorUpgrade V1.02, MAP_Misc

Warning, DAC has lots of functions, the documentation is accordingly comprehensive.
If you had a look at the 5 demonstrations it should be easier for you to understand the different topics.

If you want to create DAC-zones quickly with standard parameters I recommend you to read
the enclosed “quick start guide”. There you will learn how to build DAC zones in a few steps.

Content of the Zip:

DAC Readme.pdf
DAC Short instruction.pdf (short readme)
DAC AI Behavior Plan (infantry only)
DAC Demonstration (Part 1-5)
DAC Example Missions
DAC Core (the DAC folder and the init file)
EditorUpdate V1.02

Theme overview:

02 Advice to the DAC + prepare the DAC
03 The DAC script requestion for a zone
06 The selection of the side, the unit-, behaviour- and camp configuration
08 Creating several DAC zones (copy)
09 connect DAC zones with each other
10 generate waypoints
11 implant user defined waypoints
12 activate / deactivate DAC zones
13 reduce infantry groups
14 activate and configurate AI respawn
15 connect camps with zones
16 adjust zone values (size, position and configuration)
18 The DAC Artillery
19 implant editor groups
20 The config files
20 DAC_Config_Creator
24 DAC_Config_Units
25 DAC_Config_Behaviour
28 DAC_Config_Waypoints
29 DAC_Config_Camp
32 DAC_Config_Arti
34 DAC_Config_Events
36 DAC_Config_Marker
38 Tips, tricks and rules
38 thanksgivings
39 Attachment

Hint DAC

This version of DAC V2.0 is a beta version. Please be aware that bugs are bound to occur or that DAC could react faulty under certain circumstances.

It is very important to feed the DAC with **valid** data which are mainly stored in the **configuration files.**
When problems occur you always should try to feed the zones in your mission with default data.

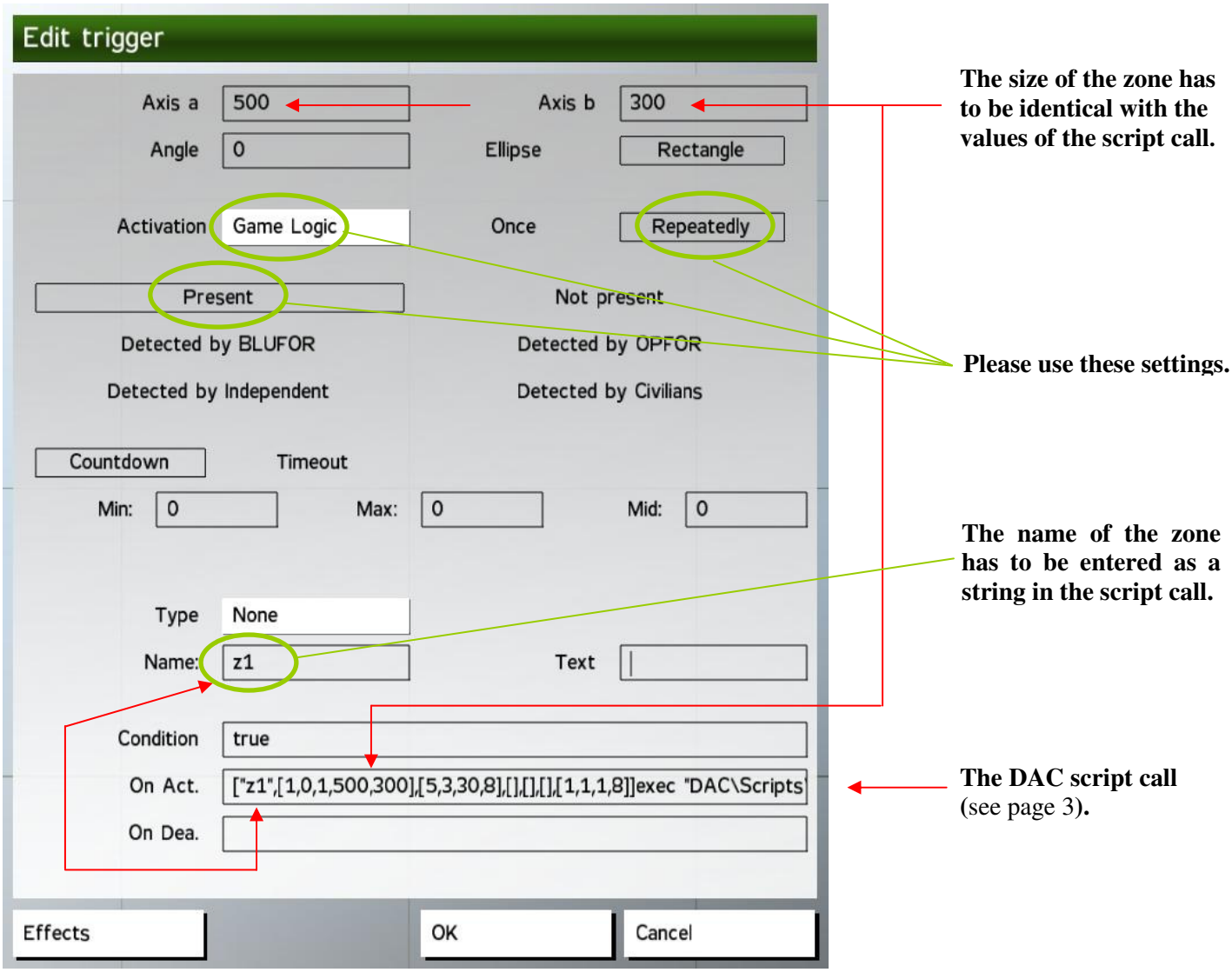
It can be very helpful to deactivate critical zones for a test by writing **false** into the on activation line of the zone when you have problems (the zone then is out of order)

Prepare the DAC

DAC V2.0 no more needs dummy units and you do not have to place markers. You solely have to copy the DAC-directory into your mission directory (**not the directory from the demos**).

DAC has to be initialised with the script call ***execVM "DAC\DAC_Config_Creator.sqf"*** .
It does not matter whether the script is called in the Init.sqs or in the init field of a logic unit.

Apart from the script call DAC additionally needs at least one trigger (a zone) to be able to start.
Such a zone has to be configured as follows (size, name and script call are variable):



Important note

For the group reduction and unit deletion to work 100% properly you have to name the player or all playable units in a MP mission and store them in an array.

The default values are **s1** to **s16**. Thus you do not have to do further customisation if you name your player **s1** and further players „**s2**,**s3**,**s4**“ for example. If you want to use different names, you have to put these names in the **DAC_STRPlayers** array as strings. The array is located in the **DAC\DAC_Config_Creator.sqf** script.

This array cleans itself, so as much (even invalidly) entries as possible can be registered in this array; e.g. **DAC_STRPlayers = [“s1”,”s2”,”w1”,”w2”,”ap1”,”ap2”,”ap3”.....]**

As soon as the mission starts, DAC reduces this array to used entries.

The DAC script call

The DAC script call contains the complete configuration of a zone. Without this entry in a zone or a trigger DAC will not start. The script call looks like this:

```
["z1"[1,0,0,400,200],[ ],[ ],[ ],[ ],[1,1,0,6]]exec "DAC\Scripts\DAC_Init_Zone.sqs"
```

First we have a look at the 4 empty arrays which I have coloured.
There the amount of units and waypoints are defined that should be generated in the zone.

As I said during the demonstration DAC can generate up to 4 unit categories per zone simultaneously whereas there are 5 different categories in total.

Each of these arrays represents a different unit category:

1.empty array generates infantry
2.empty array generates wheeled vehicles
3.empty array generates tracked vehicles
4.empty array generates helis or spawncamps

Here is an example of a script call with all 4 categories:

```
["z1",[1,0,0,800,500],[8,2,50,8],[2,2,20,6],[5,1,30,8],[2,2,3],[0,0,0,0]exec "DAC\Scripts\DAC_Init_Zone.sqs"
                      |          |          |          |
                      8 groups with infanterie
                      (group size 2, whole 50 WP, 8 WP /group)
                                 |          |          |
                                 2 group with wheeled vehicles
                                 (group size 2, whole 20 WP, 6 WP /group)
                                            |          |
                                            5 groups with tracked vehicles
                                            (group size 1, whole 30 WP, 8 WP / group)
                                                       |
                                                       2 groups helos
                                                       (group size 2, 3 WP / group)
```

But first the description of the individual categories:

Infantry:

1. number of generating groups

2. size of the group
1 = **2 - 4** units/group
2 = **2 - 6** units/group
3 = **2 - 9** units/group
4 = **2 - 12** units/group

3. number of waypoints, which shall be generated (WP-pool, which use the infantry)
4. number of waypoints, that each group get from the WP-pool
(not exact number of wp, DAC counts +/-1. Example:number 5 would give each group 4-6 waypoints.)

example**: ["z1",[1,0,0,300,200],[5,2,50,6],[ ],[ ],[ ],[0,0,0,0]]exec "DAC\Scripts\DAC_Init_Zone.sqs"**

In this zone **5** infantry groups will be created, the group size is **2-6** units, waypoint-pool of **50** waypoints/zone (just for infantry), each group get **5-7** waypoints of this pool.

Wheeled vehicles:

1. number of generating groups

2. size of the group
1 = **2 - 4** units/group
2 = **2 - 6** units/group
3 = **2 - 9** units/group
4 = **2 - 12** units/group

3. number of waypoints, which shall be generated (WP-pool, which use the unarmed vehicles)
4. number of waypoints, that each group get from the WP-pool
(not exact number of wp, DAC counts +/-1. Example:number 5 would give each group 4-6 waypoints).

example**: ["z1",[1,0,0,300,200],[ ],[3,3,30,5],[ ],[ ],[0,0,0,0]]exec "DAC\Scripts\DAC_Init_Zone.sqs"**

In this zone **3** groups with unarmed vehicles will be created, the group size is **2-9** units, a wp-pool of **30** waypoints is created, each created group get **4-6** waypoints of this pool.

The number of vehicles is dynamicly generated. Example...
A group with 9 units is generated. Only after this procedure is final, the vehicles are produced. E.g. only if a jeep is generated, there only 4 units fit purely. Now the next vehicle is generated. If the remaining 5 units fit there purely, no further vehicles are generated. However e.g. again if a jeep was generated, still a unit remains, which does not find a place. This situation leads to the fact that a further vehicle must be generated.

Tracked vehicles:

1. number of generating groups

2. size of the group
1 = **1** vehicle/group
2 = **1 - 2** vehicle/group
3 = **1 - 3** vehicle/group
4 = **1 - 4** vehicle/group

3. number of waypoints, which shall be generated (WP-pool, which use the armed vehicles)
4. number of waypoints, that each group get from the WP-pool
(not exact number of wp, DAC counts +/-1. Example:number 5 would give each group 4-6 waypoints).

example**: ["z1",[1,0,0,300,200],[ ],[ ],[3,3,25,5],[ ],[0,0,0,0]]exec "DAC\Scripts\DAC_Init_Zone.sqs"**

In this zone **3** groups with armed vehicles will be created, the group size is **1-3** vehicles, a wp-pool of **25** waypoints is created, each created group get **4-6** waypoints of this wp-pool.

Helicopter:

1. number of generating groups (Helis)

2. size of the group
1 = **2 - 4** units/group
2 = **2 - 6** units/group
3 = **2 - 9** units/group
4 = **2 - 12** units/group

3. ~~number of waypoints, which shall be generated~~ (invalid for helicopter)
4. number of waypoints, that each group get from the WP-pool
(not exact number of wp, DAC counts +/-1. Example:number 5 would give each group 4-6 waypoints).

example**: ["z1",[1,0,0,300,200],[ ],[ ],[ ],[2,2,4],[0,0,0,0]]exec "DAC\Scripts\DAC_Init_Zone.sqs"**

In this zone **2** helicopters will be created, the group size is **2-6** units (at upon called conditions), a heli gets **3-5** waypoints awhile the flight, after finishing the waypoints, the heli will land.

The group size is only responsed, if the heli has cargo capacities. A AH64 e.g.won´t take more units. Just the pilots will be generated.

If you have these 4 arrays in the grasp, it is really simply to deal with it. Up to the array with the helos, the parameters are actually identical. If you would not like to provide a category to certain units, you must leave only the appropriate array empty.

.

Choice of side, the unit-, behaviour- and camp- configuration

Here these characteristics are summarized, since they stand together also in the script call in an array. In the script call is this array at the end (marked light blue).

["z1",[1,0,0,300,200],[8,2,50,8],[2,2,20,6],[5,1,30,8],[2,2,3],[0,0,0,0,0]]exec"DAC\Scripts\DAC_Init_Zone.sqs"

parameter 1 = choice of side for the units in this zone
parameter 2 = choice of unit configuration in this zone
parameter 3 = choice of behaviour configuration in this zone
parameter 4 = choice of camp configuration in this zone
parameter 5 = choice of waypoint configuration in this zone (optional)

choice of side

For the choice of side are 4 pref. options:

0 = east | **1** = west | **2** = resistance | **3** = civil

In our example **east** is in use, because the parameter is **0**.

choice of unit configuration

The choice of units of one zone is Regulated by numbers. One number is one entry in the script **DAC\Configs\DAC_Config_Units.sqf.** See the following figure:

```
//-------------------------------------------------------------------------------------------
    case 0:
    {
        _Unit_Pool_S =  [
                            "SoldierECrew","SoldierEPilot","SquadLeaderE","Soldiereb",
                            "Soldiereg","Soldiereat","Soldieremg","Soldiereaa","Soldiereat",
                            "Soldiereb","Soldieremedic","Soldieremg","Soldieresniper",
                        ];
        _Unit_Pool_V =  [   "UAZMG","Ural","UAZ","UAZ_AGS30","BRDM2","UAZMG"];
        _Unit_Pool_T =  [   "BMP2","T72","T72","ZSU","BMP2","ZSU","BMP2"];
        _Unit_Pool_A =  [   "MI17","KA50"];
    };
//-------------------------------------------------------------------------------------------
    case 1:
    {
        _Unit_Pool_S =  [
                            "SoldierWCrew","SoldierWPilot","SquadLeaderW","Soldierwb",
                            "Soldierwg","Soldierwmg","Soldierwar","Soldierwmg","Soldierwat",
                            "Soldierwmedic","Soldierwat","Soldierwsniper","SoldierWSaboteur","SquadLeaderW",
                        ];
        _Unit_Pool_V =  [   "HMMWV50","HMMWV","Truck5t","HMMWVMK","Stryker_TOW","Truck5tOpen","Truck5tMG"];
        _Unit_Pool_T =  [   "M1Abrams","Vulcan","M113","Vulcan","Stryker_ICV_M2"];
        _Unit_Pool_A =  [   "AH1W","UH60","MH6"];
    };
//-------------------------------------------------------------------------------------------
```

The pic reflect **cofiguration 0** (case 0:) and **configuration 1** (case 1:) This script can be extended at will. In order to provide determined units in a zone, you must indicate thus only the appropriate config number in the script call.

In each zone thereby another config number can be addressed.
You must only make certain that you also installed and/or loaded the selected units.

Standard units will be found at:

0 = standard east units | **1** = standard west units | **2** = standard resistance units | **3** = civil units

Warning:
Please be aware that in order to generate addon units with DAC you have to have loaded the addon.
Furthermore problems will occur when you generate units on wrong sides.

choise of the behaviour- configuration
This parameter determines the basic behavior of the units in a zone. Basic behavior means how the generated units behave on their ways without influence by enemy units.

The configuration for the behaviour is selected again by a number.
In the Script **DAC\Configs\DAC_Config_Behaviour.sqf** you find different configurations for it.
This Script can be extended also again by as many entries as desired .

Each zone can be configured with an individual basic behavior.

choise of the camp- configuration
The last parameter is responsible for the camp configuration. Thus it is only relevant if there is at least one camp generated in the zone (the parameter also has to be there even when no camp is generated).

Analogue to the behaviour configuration a number has to be entered here. It loads the according configuration from the script **DAC\Configs\DAC_Config_Camps.sqf**.

The complete structure of a camp is defined in the camp configuration.

choise of the waypoint- configuration
This parameter is **optional** and does not have to be set. It defines the waypoint configuration which will be used for the zone.

The default waypoint configuration is **0** which is sufficient in most cases. Only when for example waypoints need to be generated in a certain height you can set another waypoint configuration which is a valid value from **DAC\Configs\DAC_Config_Waypoints.sqf**.

Here the script call again with a quick explanation

```
["z1",[1,0,0,300,200],[8,2,50,8],[2,2,20,6],[5,1,30,8],[2,2,3],[1,1,2,7,2]]exec "DAC\Scripts\DAC_Init_Zone.sqs"
```

- side west
- standard west units
- basic behaviour configuration 2
- camp configuration 7
- waypoint configuration 2

A detailed description how to configure the basic behavior, the camps and waypoints and which possibilities can be adjusted you find also again at the topic **the config files**

Create multiple zones (copy)

Once you have created the zone, a zone with the name **z1** for example, you can easily create more zones by copying the zone **z1** and pasting it to another place (place mouse cursor exactly over the zone > press Ctrl+C > move mouse cursor to new position > press Ctrl+V).

Assuming you copy a zone called "z1", so its copy is automatically called by Arma "z1_1".

It is important to enter the zones´ Name in the script call

Tip: use short names, so the letterbox won´t fade.

The copied zone can now be changed in size, position and parameter if desired.Finally the new zone is ready to work. In this way you are able creating and configuring zones in less than a few minutes without leaving the editor ☺

Suppose you have copied a zone 2 times and already adapted the names of the zones and the scipt call
The scipt call from the zones then could look in such a way:

```
["z1",[1,0,0,300,200],[3,2,20,5],[ ],[ ],[ ],[0,0,0,0]]exec "DAC\Scripts\DAC_Init_Zone.sqs"
["z2",[1,0,0,300,200],[3,2,20,5],[ ],[ ],[ ],[0,0,0,0]]exec "DAC\Scripts\DAC_Init_Zone.sqs"
["z3",[1,0,0,300,200],[3,2,20,5],[ ],[ ],[ ],[0,0,0,0]]exec "DAC\Scripts\DAC_Init_Zone.sqs"
```

3 zones which only have to have a different name

This are 3 zones (**z1, z2, z3**), in which 3 infantrie groups will be generated.

If you want the zones working locally you need to change one parameter.
I marked the parameter **red**, this is the **ID** of the zone.
Give every zone a different number, see the following:

```
["z1",[1,0,0,300,200],[3,2,20,5],[ ],[ ],[ ],[0,0,0,0]]exec "DAC\Scripts\DAC_Init_Zone.sqs"
["z2",[2,0,0,300,200],[3,2,20,5],[ ],[ ],[ ],[0,0,0,0]]exec "DAC\Scripts\DAC_Init_Zone.sqs"
["z3",[3,0,0,300,200],[3,2,20,5],[ ],[ ],[ ],[0,0,0,0]]exec "DAC\Scripts\DAC_Init_Zone.sqs"
```

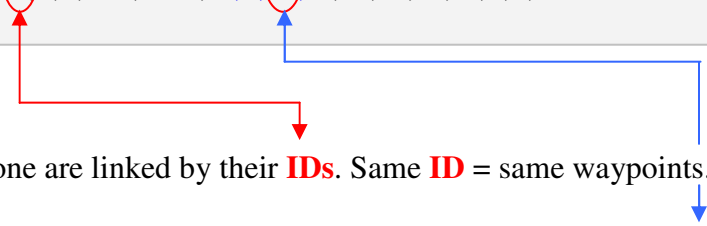
That is the zones' **ID-Nr**

It does not matter what number you enter there. But it is important that each zone that shall work local has to have an own number which is not used by any other zone.

Linking zones

Nothing else signifies linking zones than to fold up the waypoints of several zones. This means again that the units also have access to the waypoints of the linked zones. For linking zones the **ID-Nr**. is also used again. Here once more the example from completely on top:

```
["z1",[1,0,0,300,200],[3,2,20,5],[ ],[ ],[ ],[0,0,0,0]]exec "DAC\Scripts\DAC_Init_Zone.sqs"
["z2",[1,0,0,300,200],[3,2,20,5],[ ],[ ],[ ],[0,0,0,0]]exec "DAC\Scripts\DAC_Init_Zone.sqs"
["z3",[1,0,0,300,200],[3,2,20,5],[ ],[ ],[ ],[0,0,0,0]]exec "DAC\Scripts\DAC_Init_Zone.sqs"
```



The 3 zone are linked by their **IDs**. Same **ID** = same waypoints.

That does mean each of those 9 infantry groups has access to **60** waypoints. Which waypoints will be used by groups is choosen randomly.

With this state the units often switch between the zones :-)
In the **DAC demonstration** I have already introduced to you the linked zones.

more variations of linked zones:

```
["z1",[1,0,0,300,200],[3,2,20,5],[ ],[ ],[ ],[0,0,0]]exec "DAC\Scripts\DAC_Init_Zone.sqs"
["z2",[1,0,0,300,200],[ ],[ ],[3,1,25,8],[ ],[0,0,0]]exec "DAC\Scripts\DAC_Init_Zone.sqs"
["z3",[1,0,0,300,200],[3,2,20,5],[ ],[ ],[ ],[0,0,0]]exec "DAC\Scripts\DAC_Init_Zone.sqs"
```

This example also shows 3 linked zones, however in the zone **z2** tracked vehicles are provided. Although this zone is linked with the two infantry zones. The tracked vehicles won´t leave their zone, because in the zones **z1** + **z2** waypoints are not at disposal. Basicly you don't need to link zone **z1** with zone **z2**.

```
["z1",[1,0,0,300,200],[3,2,20,5],[ ],[ ],[ ],[0,0,0]]exec "DAC\Scripts\DAC_Init_Zone.sqs"
["z2",[1,0,0,300,200],[3,2,15,5],[ ],[ ],[ ],[0,0,0]]exec "DAC\Scripts\DAC_Init_Zone.sqs"
["z3",[2,0,0,300,200],[ ],[ ],[3,1,25,8],[ ],[0,0,0]]exec "DAC\Scripts\DAC_Init_Zone.sqs"
["z4",[2,0,0,300,200],[ ],[ ],[3,1,35,8],[ ],[0,0,0]]exec "DAC\Scripts\DAC_Init_Zone.sqs"
```

4 Zones named **z1** to **z4** are placed on the map. Above you see their script calls.
Zones **z1** and **z2** are linked by ID**1** → **35** waypoints at disposal
Zones **z3** and **z4** are linked by ID**2** → **60** waypoints at disposal

Create waypoint zones

These waypoint zones have the purpose to give further waypoints to the units from the master zone.
In these waypoint zones no units are produced.

You find an example in the **dac demonstration**.

By providing waypoint zones, the dynamics increase. Here's an example:

```
["z1",[1,0,0,300,200],[15,2,10,15],[ ],[ ],[ ],[0,0,0,0]]exec "DAC\Scripts\DAC_Init_Zone.sqs"
["z2",[1,0,0,300,200],[20],[ ],[ ],[ ],[0,0,0,0]]exec "DAC\Scripts\DAC_Init_Zone.sqs"
["z3",[1,0,0,300,200],[20],[ ],[ ],[ ],[0,0,0,0]]exec "DAC\Scripts\DAC_Init_Zone.sqs"
["z4",[1,0,0,300,200],[20],[ ],[ ],[ ],[0,0,0,0]]exec "DAC\Scripts\DAC_Init_Zone.sqs"
```

The example shows again 4 zones **z1** to **z4**. The zones are linked over the ID **1** with each other. Zone **z1** is the master zone, the 4 parameters for the units are registered here:
15 infantry groups, group size **2** (2-6 units per group). In this master zone **10** waypoints are generated and each group receive **15** waypoints for its routes.

You see the waypoint zones **z1** - **z3** by the individual entry in that corresponding array. This entry indicates the number of waypoints, which are to be generated in this zone.

In our example are 3 zones with **20** waypoints. Therefore, the generated groups from the master zone have Access on a total of **70** waypoints (15+20+20+20).

Once again to understand:
An array with 4 parameters (or 3 parameter with Helos) produces waypoints **and** units and is the master zone or origin zone of the units. An array with just one parameter produces **only** waypoints for the appropriate category.

Here's an example, how you can combine two different master zones:

```
["z1",[1,0,0,300,200],[15,2,10,15],[ ],[ ],[ ],[0,0,0,0]]exec "DAC\Scripts\DAC_Init_Zone.sqs"
["z2",[1,0,0,300,200],[ ],[ ],[3,1,25,8],[ ],[0,0,0,0]]exec "DAC\Scripts\DAC_Init_Zone.sqs"
```

Indeed, this zones are connected for the **ID 1**, but the units have to come no possibility to the other zone, because there are no waypoints of the respective category in the other zone.

If the zones combined, the script call looks like this:

```
["z1",[1,0,0,300,200],[15,2,10,15],[ ],[20],[ ],[0,0,0,0]]exec "DAC\Scripts\DAC_Init_Zone.sqs"
["z2",[1,0,0,300,200],[20],[ ],[3,1,25,8],[ ],[0,0,0,0]]exec "DAC\Scripts\DAC_Init_Zone.sqs"
```

In zone **z1** **20** waypoints for tracked vehicles are generated additionally, and in zone **z2** **25** waypoints for infantry are produced. By this small change the zones can now exchange their units.

Merge user define waypoints

It is possible to integrate user-defined waypoints into the system.This function is always usefull, when you like to merge completely determined firm positions from which the appropriate units can choose.

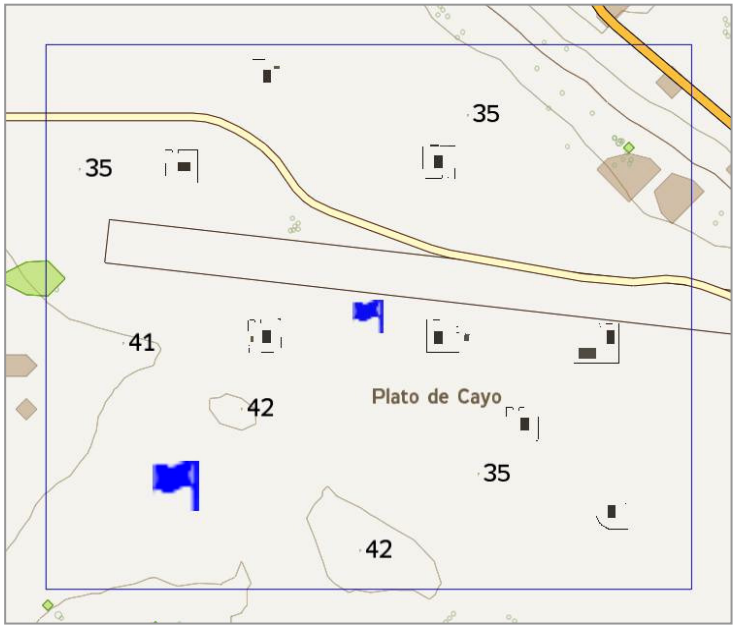
That positions can be in a village or also at mission relevant positions.

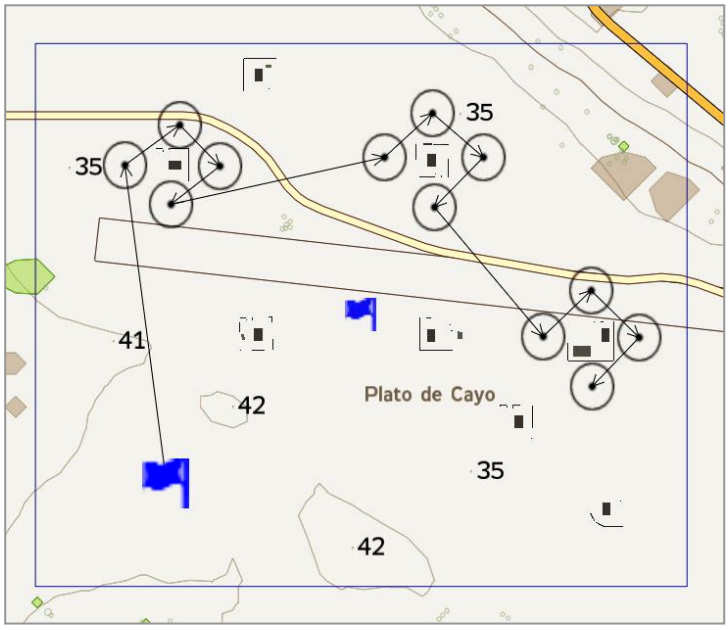
As a result of this function, 3 possibilities arise of filling zones with waypoints:

1. **only** generated waypoints
2. generated **and** user defined waypoints
3. **only** user defined waypoints

It is quite easy to build-in user defined Waypoints:

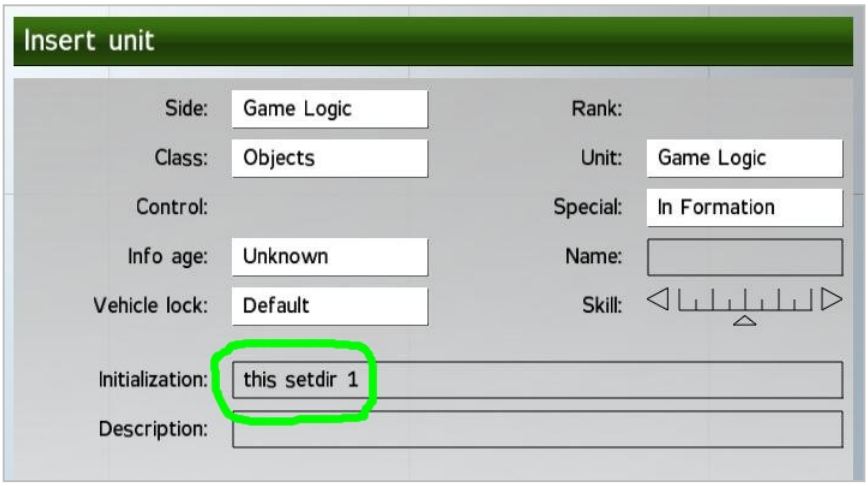
Place a logic-unit in the corresponding zone. Where doesn´t matter, just inside the zone of course.	Give the logic at certain position a few waypoints. The waypoints have to be in the zone.





Now you have to decide to which unit categorie the waypoint will belong to.

Manage this by the init line of your logic: →



this setdir **1** = **infantry**
this setdir **2** = **wheeled vehicles**
this setdir **3** = **tracked vehicles**
this setdir **4** = **helis**
this setdir **5** = **camps**

De/activate DAC zones

If necessary, each zone can be activated or deactivated at a later time.

Late **activating** means that everything is prepared for the zone, its units are however still held back, and are only produced when a certain condition is met.

Deactivation means that all units of a zone are deleted when a certain condition is met. Both methods primarily have the purpose to save and/or again release performance

Zones, that are needed only in the later process of the mission, since they are far off from the first mission goal, can so be set on "Standby". That will possibly save much performance.

The same goes for zones, which have no more meaning for the mission, these can be deleted to save performance too.

DAC therefore has a function which can be used multiple times on a zone. In other words you have the possibility to activate and deactivate a zone multiple times.

Please be aware that a zone which is activated/deactivated multiple times always is restored to its initial state. That means if you deactivate a zone in which some units died and you activate this zone again later, all units that had been defined in this zone will be generated.

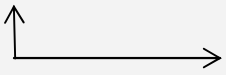
A zone will not be automatically deactivated once all units of this zone are down. You can reanimate such a zone by deactivating it and afterwards activating it again.

The function is called like this:

[NameOfTheZone] **call DAC_Activate** example: ([z1] call DAC_Activate or [z1,z2,z3] call DAC_Activate)
[NameOfTheZone] **call DAC_Deactivate** example: ([z1] call DAC_Deactivate or [z1,z2,z3] call DAC_Deactivate)

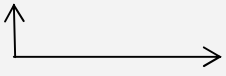
If you want that a zone from the beginning is inactive, you can use the following parameter :

["z1",[1,0,0,300,200],[15,2,10,15],[ ],[ ],[0,0,0,0]]exec "DAC\Scripts\DAC_Init_Zone.sqs"



The zone is **active** from the beginning

["z1",[1,1,0,300,200],[15,2,10,15],[ ],[ ],[0,0,0,0]]exec "DAC\Scripts\DAC_Init_Zone.sqs"



The zone is **inactive** from the beginning

Warning, when you **activate/deactivate** zones you always should keep linked zones in mind. Thus linked zones should be **activated/deactivated** together. Please be aware that there is a short break between the **activating/deactivating** (min. 3 sec.).

Reducing of infantry-groups

The **reduction** of infantry groups beyond a certain range is a way to save performance.

Since all AI-referred scripte run over the leader of a group, it does not matter, if the remainder of the group is not present. If the "reduced group" approaches a hostile unit or a player unit, the remainder of the group is again produced and the group is complete again.

Before the group is **reduced**, the necessary data of each unit are saved, being:

- → **type of unit**
- → **skill of unit**
- → **dammage of unit**
- → **ammo and weapons**

Only the number of magazines will be saved not each bullet ;-)

Groups of the category **Wheeled** will be reduced latterly, except one limitation:
Only groups which consist of a maximum of one vehicle.
Groups with multiple vehicles in will not be taken into account.

This means:
One group composed of **4** units and 1 **HMMWV**, **will be reduced**
One group composed of **12** units and 1 **Truck5t**, **will be reduced**
One group composed of **5** units and 2 **HMMWV**, **will not be reduced**
One group composed of **10** units and 2 **HMMWV** + 1 **Truck5t**, **will not be reduced**

In general the group is respawned at their leader. This circumstance will lead to a situation where you can hear the units reloading their guns. Depending on the distance you may hear the click sound
(multiple times for multiple units).

To avoid this problem the DAC offers following possibility:

Place a **logic** unit in the editor and name it **DAC_Pos_E**.
As soon as this **logic** exists the DAC will respawn all hostile units there.
The units will be transferred back when they have fully equipped and reloaded. Therefore the “annoying” sounds are gone ☺

It is recommended to place the **logic** far away from the action, but not in the water ☺
To make sure that the units won't interfere with each other you can twist the **logic**. The angle of the **logic** equals the units placing radius.

Place a second **logic** in the editor and name it **DAC_Pos_W** to provide friendly units a position to respawn
(not directly beside the **logic DAC_Pos_E**).

How to activate or deactivate the group reduction can be read under the topic **The configuration files**.

Activate and configure AI Respawn

In order to activate the Respawn, at least one camp on the appropriate side must be generated. So that the script call does not become more complex, I decided to pack the parameters for this Spawn camps into the array for Helicopter.

That means, in a zone either Helicopter can be generated, or Spawn camps. I think that is a good compromise.

Only the following units are respawned: **Infantry**, **wheeled Vehicle** and **tracked vehicles**

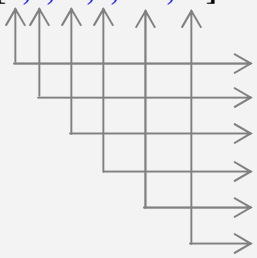
Here an example, as a Spawn camp (configuration **3**) is generated:

```
["z1",[1,0,0,800,400],[],[],[],[ 1,2,50,0,100,10],[0,0,0,3]]exec "DAC\Scripts\DAC_Init_Zone.sqs"
```

You need 6 parameters in order to provide one or more Spawn camps per zone.
For helicopter you need 3 optional 4 parameters. In addition, a valid configuration No. must be entered.

The meaning of the parameters:

[1,2,50,0,100,10]



the number of camps to generate.
the size of group stationed (The group size behaves exactly like with infantry).
the movement radius for stationed group (meter).
spawntype **0** = **infantry + vehicle** | **1** = **infantrie only** | **2** = **vehicle.only**
the possibility for respawn in **%** percent.
the number of spawns per camp in the zone, not the complete number in the zone

[3,2,50,0,100,5] That would mean e.g. that in the zone **3** camps are generated, and each camp has **5** respawns available, all of the same type. Thus the zone has **15** respawns altogether. If the group of a camp is extinguished, also the remaining spawns are lost.

Attention, the more camps you would like to generate in a zone, the more place is needed. Give the zone enough free space unless you want to have more then one camp in a zone. If there's more than one camp in a zone, the respawn will take place randomly in one of the camps.

Information:

The group which is located at a camp will be dissolved so far all respawns have been used.
The camp-group will join the next reachable group then. If a camp group will be totally destroyed so the number of respawns, which are still left, will be set on **0**.

More details on adjustment possibilities you ´ll find again under the topic: **The configuration files**

Linking camps with zones

When multiple camps are generated on one side, respawns are executed randomly per camp.
This behaviour is not very ideal, as the new generated units may have a very long way back into the mission area.

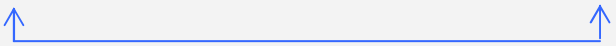
To narrow the randomness down as well as having more adjustment possibilities it is possible to link camps with zones. This means that you can define which camp is responsible for a zone.

Imagine you have two camp zones with a total of one camp and two master zones each, where only units are generated. Normally these two camps would supply both the master zones with new units.

But if you would like to have every camp beeing responsible for exactly one master zone, you can do so by using this function.

The concept is very simple:
In the camp zone you just have to specify the zone which shall be supplied by the camp zone:

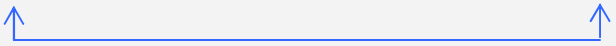
```
["z1",[1,0,0,800,400],[],[],[],[ 1,2,50,0,100,10,[z2]],[0,0,0,3]]exec "DAC\Scripts\DAC_Init_Zone.sqs"
```



You can see above that the camp zones (it can be several) from **z1** supply zone **z2**.

An optional parameter was given in the camps array. This parameter has to be of type “Array” (to enter multiple zones) and the real zonename has to be entered, instead as “String”.

```
["z3",[1,0,0,800,400],[],[],[],[ 1,2,50,0,100,10,[z4]],[0,0,0,3]]exec "DAC\Scripts\DAC_Init_Zone.sqs"
```



This would be the other camp zone **z3**, which is responsible for zone **z4**.

Like I implied, you can enter multiple zones directly:

```
["z1",[1,0,0,800,400],[],[],[],[ 1,2,50,0,100,10,[z4,z5]],[0,0,0,3]]exec "DAC\Scripts\DAC_Init_Zone.sqs"

["z2",[2,0,0,800,400],[],[],[],[ 1,2,50,0,100,10,[z6,z7]],[0,0,0,3]]exec "DAC\Scripts\DAC_Init_Zone.sqs"

["z3",[3,0,0,800,400],[],[],[],[ 2,2,50,0,100,10,[z5,z7]],[0,0,0,3]]exec "DAC\Scripts\DAC_Init_Zone.sqs"
```

The camp zone **z1**, supplies the zones **z4 + z5**
The camp zone **z2**, supplies the zones **z6 + z7**
The camp zone **z3**, supplies the zones **z5 + z7**, which are already supplied by the zones **z1 + z2**.

Attention: The use of this function disables the global respawn. That means that camp zones which are not linked can only supply themselves but no other zones.

Changing zone values (size, position and configuration)

It is possible to change the size as well as the position of a DAC zone. Such a change always causes a recreation of new waypoints for the corresponding units from the zone.

The positions of the old waypoints are overwritten with the positions of the new generated waypoints.
This process can take several seconds depending on the amount of waypoints in the zone.

When the process is done, all groups have new waypoints then.
Depending on the DAC settings changes are broadcasted over the sidechat, even the groups report in when they received new waypoints.

Please note that camps or groups who are stationed at the camps, are not affected by changing the zones.
Camps are uniqe installments and cannot be moved

Some hints you should keep in mind when changing zones:

Always try to move a zone controlled to make sure that enough waypoints can be generated at the new position.
When in doubt test it with activated waypoint and zone markers.
Also activate the DAC system messages which will show you exactly which waypoints were generated and which not.

.

Please note that when a zone is moved and no waypoints can be generated at the new position because of high object density or the terrain is not suitable, the zone will be shown on the new position but the units will still move on their old waypoints.
It can therefore happen that infantry units will move towards the new placed zone but tracked vehicles will remain because no waypoints could be generated for them.

You can change not only activated zones but also deactivated zones.
It is possible to deactivate a zone (units in the zone become deleted), then move it and activate it again at its new position.

When zones are moved to far away it can happen that the units won't move anymore, especially when many groups are affected (I have no explanation for that yet).
Probably it helps to split up the zones and perform the zone changings a bit delayed.

Custom waypoints are lost when a zone is moved.

The scriptcall looks like this:

[[zonename, position, size, unitconfig, behavconfig, wpconfig]**] call DAC_fChangeZone**

1. The name of the zone	the real name of the zone (not as string)
2. The position	where the zone shall be placed [positions array]
3. The size	for a possible change of the size [x,y]
4. Specification of the unit configuration	a valid number from the DAC_Config_Units.sqf
5. Specification of the behaviour config.	a valid number from the DAC_Config_Behaviour.sqf
6. Specification of the waypoint config.	a valid number from the DAC_Config_Waypoints.sqf

Example 1: If you want to change only the size of a zone the scriptcall looks like this**:**

```
[z1,[],[300,400],0,0,0] call DAC_fChangeZone
```

The zone **z1** is changed to a size of 300x400 meters. As we won't change the position the position array remains empty. The different configuration values remain 0.

Example 2: The zones position is moved to the players position:

```
[z1,Position Player,[],0,0,0] call DAC_fChangeZone
```

The zone **z1** is moved onto the players position. As we won't change the size the size array remains empty.
The different configuration values remain 0.

Example 3: You want to change the size as well as the position of the zone:

```
[z3,Position Logic1,[500,1200],0,0,0] call DAC_fChangeZone
```

The zone **z3** is moved onto the position of the logic „Logic1" and the size is changed to 500x1200 meters.
The different configuration values remain 0.

Another option is that you can load a new configuration for a certain zone.
The following configurations can be changed:

```
DAC_Config_Unit      →   [z3,[],[],1,0,0] call DAC_fChangeZone
DAC_Config_Behaviour →   [z3,[],[],0,4,0] call DAC_fChangeZone
DAC_Config_Waypoints →   [z3,[],[],0,0,2] call DAC_fChangeZone
```

To load a new configuration for a zone, you have to enter a valid number from the corresponding configuration in the scriptcall.

Keep following in mind when changing a configuration:

When the configuration **Behaviour** is changed, the units of the zone will apply to the changes at once.
At once means in aprox. 10 seconds.

If you change the configuration **Unit**, the changes won't apply at once, rather when there are new units generated for the zone.
That's the case when a group is destroyed and a new group is generated in a camp.

You can replace a zone by a complete different unit configuration. Therefore you have to deactivate the zone (all units become deleted), then load a new units configuration followed by a re-activiation of the zone.

Attention. As the side of a zone cannot be changed, you have to pay attention that a unit configuration which fits the according side is loaded, otherwise there will be a massacre ;-)

So please do not generate west units on the east side !

The DAC artillery

The DAC Artillery is completely scripted and available for the AI as well as for the player.
All parameters needed for the DAC-Artillery are stored in its own configuration file.
There are three levels of conditions, which have to be fulfilled in order to get the artillery attack:

Condition Level 1 > refers to the **Target-Group** (the hostile group, detected by AI)
Condition Level 2 > refers to the **Call-Group** (the group which requests the artillery strike)
Condition Level 3 > refers to the **Artillery-Unit** (an unit which is considered for an artillery strike)

DAC artillery for the AI

The DAC artillery (AI) basically can only be requested by infantry units. The request is triggered when an AI group has made contact with the enemy and knows enough about it.
The request runs through the 3 levels of conditions and will lead to an artillery strike if successful.

The configuration of the artillery and the conditions are defined in the **DAC_Config_Arti.sqf**.
Like all the other configuration files, several changes can be done. For each zone you can load a seperate artillery configuration. If no configuration is loaded the units from this particular zone can't request artillery.

DAC artillery for the player

The DAC artillery (Player) can be called at anytime from everywhere. Thats why **condition level 1** is not applicable.

Unlike the DAC artillery (AI) which is triggered by the AI automatically, the DAC artillery (Player) has to be triggered manually. The DAC offers an easy script call for that:

[playerName,positionsArray,configurationNumber,artilleryRadius] **spawn DAC_fCallArti**

Example: [Player,Position LogA,3,50] spawn DAC_fCallArti
Artillery is called on the position of the logic named “LogA”.
Artillery configuration = 3, Radius max. 50m

This call itself does not lead to an artillery strike, it just triggers the artillery request.
If there is an execution, finally depends on the setup of the configuration.

When you enabled the DAC debug mode (refer **DAC_Config_Creator**), the DAC will show you an **error code** If an error occurred.The errors have the following meanings:

1 = Target-Group is moving too fast [only relevant for DAC artillery (AI)]
2 = number of units in Target-Group insufficient [only relevant for DAC artillery (AI)]
3 = insufficient vehicles in Target-Grpup [only relevant for DAC artillery (AI)]
4 = insufficient units in Call-Group
5 = insufficient distance of Call-Group to Target-Group [or to position, for DAC Artillery (Player)]
6 = required type of unit in Call-Group not present
7 = required skill of Call-Group is undercut
8 = given probability was not reached
9 = friendly units undercut the minimum distance to target position
10 = no valid Artillery-Unit present, or no Artillery-Unit ready or rather in range
11 = global condition not true

How to set up the artillery is explained in the topic **DAC_Config_Arti**.

Embedding editor placed groups

It's possible to embed groups which have been placed in the editor into the DAC system.
This can be done on mission start or at any particular time.

Following categories are supported:
Infantry, **Wheeled vehicles**, **Tracked vehicles**

As soon as a group is integrated into the DAC system the following DAC attributes are broadcasted:

- The group uses DAC waypoints or rather the corresponding movement routines.
- The group is reduced.
- Dead units of the group are deleted (depends upon the settings of the delete routine).
- The group can call support + artillery.
- The group itself can be called as support.
- The group can use the DAC events.
- The group will use a DAC behaviour configuration.

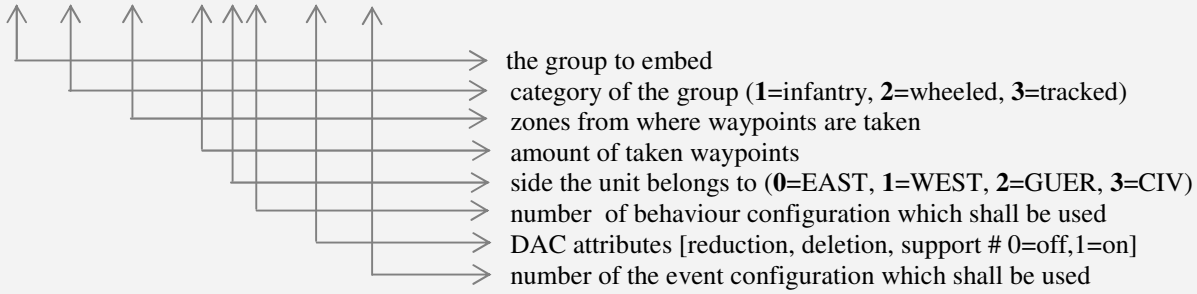
There are few limitations:

- The embedding in the DAC system is irreversible.
- The embedded group must not have own waypoints.
- An embedded group cannot be respawned.

The embedding of an editor group is done with the following scriptcall:

Example:

[_group,1,[z1,z2],10,1,1,[1,1,1],0] spawn DAC_fInsertGroup



Small example of use:
Place a transport helicopter in the editor (maybe set fuel to zero to force it to stay on the ground ;-)
Then place an infantry group which you move into the helicopter by MoveIn command.
In the init line of the groups leader you enter: **HeliGroup1 = group this**
In a certain situation you let the chopper fly into the area of operation (dont forget about the fuel) and disembark the infantry group there. This would be a good moment to embed the group into the DAC (to prevent the group from just standing around ☺):

[HeliGroup1,1,[z1],10,1,1,[1,1,1],0**] spawn DAC_fInsertGroup**

The configuration files

The configuration files are just common scripts which can be adjusted or extended individually by you.
(Except the **DAC_Config_Creator)**

The DAC gets its data input and parameters from them ;-)
In this respect they are an important component of the whole system.

There are the following configuration files:

DAC_Config_**Creator**	The basic settings of the DAC
DAC_Config_**Units**	The units configuration defines the unit types of the 4 categories per side
DAC_Config_**Behaviour**	The behaviour configuration defines the DAC groups behaviour
DAC_Config_**Waypoints**	The waypoint configuration defines the parameters for the waypoint generator
DAC_Config_**Camp**	The camp configuration defines the look and the settings of the DAC camps
DAC_Config_**Arti**	The artillery configuration defines the conditions and the settings of the DAC artillery
DAC_Config_**Events**	The event configuration allows changes on different points in the DAC

The current configuration files described in detail:

[DAC_Config_Creator]

The **DAC_Config_Creator.sqf** handles all basic settings of the DAC. You can also define & start own scripts from there, aswell as defining whether a script is started serverside and-/or clientside.
Here´s the description for variables & arrays in detail:

DAC_STRPlayers = ["s1","s2","s3","s4","s5","s6","s7","s8","s9","s10"**]**

The array where all valid playernames are given. Attention, entrys must be put as string.
Invalid entrys will be deleted automatically.

DAC_AI_Count_Level = [[2,4],[2,6],[2,8],[2,12],[6,0]**]**

Here you can define the group sizes. The first value is used to specify the minimum size, the second value represents the maximum size of a group. The group size will later be fetched from the DAC-Scriptcall:

["z1",[1,0,0,300,200],[5,**2**,50,6],[ ],[ ],[ ],[0,0,0,0]]exec "DAC\Scripts\DAC_Init_Zone.sqs"

The **2** equals the 2nd entry in the **DAC_AI_Count_Level** array, thus making it 2-6 units

DAC_Dyn_Weather = [240,60,1,0.6**]**

This parameter controls DAC´s dynamic weather system. If you do not want to use this feature,
you have to set the **1st** value to **0**.

1. Timeframe for weather changes to set in
2. Determines weather change persistency
3. Maximum allowed weather intensity
4. Maximum allowed fog density

DAC_Reduce_Value = [600,650,0.1**]**

Here you´ll find the parameters for unit/squad reduction. If you dont want to use this feature
you have to set the **3rd** value to **0**.

1. Distance setting defines when groups are being built up
2. Distance setting defines when groups become reduced (this value should be a tad higher than the first one).
3. The timeout DAC takes between units, when a group is recreated.

DAC_AI_Spawn = [[10,30,10],[20,60,15],0,360,100**]**

The basic settings and delay times of the AI-Respawn are given here. It is mandatory to have at least one
DAC-Camp generated for the AI-Respawn to work at all.

1. Respawn delay for **infantry** = seconds min. + random variation + idling period until next respawn is allowed at camp
2. Respawn delay for **vehicles** = seconds min. + random variation + idling period until next respawn is allowed at camp
3. Respawn global = 0, Respawn local = 1
4. Optional respawn camp destruction once no more respawns are available:
0 = Camp objects will stay, >0 = seconds until self destruction starts
5. Minimum required distance of all player units to a respawn camp. If player units get closer than that no respawns will be executed meanwhile.

DAC_Delete_Value = [30,150**]**

This array defines the removal of dead units. You can either use only one paramter
or combine both: [60,0] = only time is checked, [0,200] = only distance is checked, [30,150] = time first, then distance

1. The soonest time (in sec.) after which dead units can get deleted
2. Minimum distance to all player units before dead units get deleted

DAC_Del_PlayerBody = [10,150**]**

If needed you can activate the removal of dead player units (in MP) here.
Occassionaly there´s just too many corpses of dead players lying on the battlefield, depending on the used MP-Respawn.
This function gets rid of them.

1. soonest time in seconds for a dead player unit to be deleted
2. minimum distance to all player nuits before a dead player unit gets deleted

DAC_Com_Values = [1,2,0**]**

With this array you enable/disable the output of system messages:

- The DAC system messages give you information about groups being reduced, units being deleted, zones being moved, groups being respawned, etc.
- During DAC initialization a popup hint will display waypoint & unit generation details aswell as DAC initialization time.
- DAC radio messages will inform you about actions/reactions of the AI, i.e. groups calling out contacts, request reinforcements, loose contacts, etc.

1. DAC System messages: 0 = deactivated, 1 = activated
2. DAC Initialization-Hint: 0 = deactivated, 1 = minimal, 2 = maximal
3. DAC Radio messages: 0 = Messages deactivated, 1 = opposing activated, 2 = friendly activated, 3 = all activated

DAC_AI_AddOn = 1

At the moment only this setting is possible. Also the DAC AI cannot be deactivated.
You can do the following however: Make a copy of the **AI_1** folder and rename it to **AI_2**.
Thereafter this folder will be used whenever you set **DAC_AI_AddOn =** 2
This is useful if you want to modify some of the AI scripts without having to touch the original scipts.
That way you can easily switch between multiple AI folders.

DAC_AI_Level = 4

Values 1,2,3 & 4 are valid - with a value of 4 representeing the highest setting.
This setting influences AI behaviour. Lower values make the AI´s reactions more sluggish and they will not notice you as fast. The maximum number of reinforcement units is also influenced by this setting.

DAC_Res_Side = 1

This variable defines the side which RACS are fighting for. You must (!) apply the same settings that are given in the editor.

0 = resistance friendly to EAST
1 = resistance friendly to WEST
2 = resistance friendly to NOBODY

Attention! With this setting RACS will fight alone against EAST & WEST, whereas EAST & WEST are friendly to each other !!!

DAC_VehAllowed = [1,2 **]**

This line declares the types of (empty) vehicles that can be acquired by infantry units.
Apply as many config #´s from the **DAC_Config_Units** as desired.
DAC reads all vehicles from the given configurations and combines them in a pool.
Any empty vehicle given in the pool will be consiodered usable by the AI.

DAC_Marker = 1

This paramter activates / deactivates the DAC marker.
You can load various marker configurations at any time by deactivating the markers and loading another configuration afterwards. **Hint**: In MP games, only Unit and zone-marker are displayed.

0 = DAC-marker deactivated, **>0** = valid configuration No. from the **DAC_Config_Marker**.
Attention, you have to wait at least 3 seconds between activation & deactivation of markers.

DAC_WP_Speed = 0.01

Waypoint generation speed. This value only affects this initialization phase.
Attention, value 0 causes a short stutter during initialization!
Upon moving of a zone always the default value 0.01 is used.

DAC_GunAllowed = ["M2StaticMG","M119","DSHKM","AGS","D30"**]**

This line declares the types of (empty) static weapons that can be acquired by infantry units.
The function itself can be configurated in **DAC_Config_Behaviour.**

DAC_SayArrayE = [[ ],[ ],[ ],[ ],[ ],[ ]]
DAC_SayArrayW = [[ ],[ ],[ ],[ ],[ ],[ ]]

Sounds for East and West groups which will be played by the group leader in different situations can be entered here. (Attention! Sounds must be defined in the Description.ext).
Sounds will also be played in MP ☺

DAC holds a total of **7** different Situations. Thus making it 7 Arrays being available for use.

These are the situations:

1. Group reaches waypoint
2. Group spots enemy units
3. Group is under fire
4. Group requests support
5. Group lost enemy contact
6. Group requests Artillery
7. Group affirms supprt request

An example (Info: each entry stands for a soundfile):

DAC_SayArrayE = [[“rus1“,“rus2“],[“rus3“, “rus4“ “rus5“],[ ],[“rus6“],[“rus9“, “rus11“],[ ],[“rus12“, “rus14“, “rus15“]**]**

- The group leader shouts **rus1** or **rus2** upon reaching a waypoint.
- The group leader shouts **rus3**,**rus4** or **rus5**, when his group spotted an enemy unit.
- The group leader shouts **nothing** when his group is under fire (empty array).
- The group leader shouts **rus6** when his group requests support.
- The group leader shouts **rus9** or **rus11** when his group lost enemy contact.
- The group leader shouts **nothing** when his group call in an artillery strike (empty array).
- The group leader shouts **rus12**,**rus14** or **rus15**, when his group affirms a support request.

More entrys in an array for each situation will offer more variation.

DAC_BadBuildings = ["CampEmpty","CampEast","Land_dum_istan4","Land_dum_istan4_detaily1"]

One can enter types of buildings here which are either not used to be entered by the AI, or even not liked to get used by the creator for the mission.

Expcept for the arrays **DAC_SayArrayE, DAC_SayArrayW + DAC_BadBuildings** you can access all settings directly via Game Logic. To do so place a logic on the map and in it´s initialization field, for example, you type:

DAC_AI_Level = 2

This will bypass the value given in the **DAC_Config_Creator** script.
That way you can easily change some settings without leaving the editor.

[DAC_Config_Units]

The unit classes are defined in the **DAC_Config_Units**, which can be generated with the DAC. These classes are apportioned in several blocks. Every block has an unambiguous number.

The 4 unit categories are defined in such a block, which will be asked when the DAC-script gets caused.
This script is as much expandable as one wants, so you can define your favourite units in there also.

This option enables one to combine units out of different Addons to save unit combinations individually.
But make sure that the respective addons are loaded.

On the image below you can see an example of such a block:

```
case 0:
{
    _Unit_Pool_S =  [
                        "SoldierECrew","SoldierEPilot","SquadLeaderE","Soldiereb",
                        "Soldiereg","Soldiereat","Soldieremg","Soldiereaa","Soldiereat",
                        "Soldiereb","Soldieremedic","Soldieremg","Soldieresniper",
                        "Soldiereaa","Soldiereminer","SoldierESaboteur","SoldierESaboteurPipe",
                        "SoldierESaboteurBizon","SoldierESaboteurMarksman"
                    ];
    _Unit_Pool_V =  [   "UAZMG","Ural","UAZ","UAZ_AGS30","BRDM2","UAZMG"];
    _Unit_Pool_T =  [   "BMP2","T72","ZSU","BMP2","ZSU","BMP2"];
    _Unit_Pool_A =  [   "MI17","KA50"];
};
```

Here you can see a DAC-script order which generates units out of the unit-configuration **0**:

["z1",[1,0,0,300,200],[5,3,30,8],[ ],[ ],[ ],[0,0,3,4]]exec "DAC\Scripts\DAC_Init_Zone.sqs"

The different Pools does have the following consequence:

_Unit_Pool_S	=	The units classes fort he Category (1) **Infantry**
_Unit_Pool_V	=	The units classes fort he Category (2) **Wheeled vehicles**
_Unit_Pool_T	=	The units classes fort he Category (3) **Tanks**
_Unit_Pool_A	=	The units classes fort he Category (4) **Helicopter**

Attention, the **_Unit_Pool_S** contains one condition, which always neede to be true, even if there are no Infantry units generated: There always have to be 3 entries defined!

The reason is: The first 3 entries are reserved fort he DAC

The 1st entry defines the type of units for the crew of **Tracked Vehicles**
The 2nd entry defines the type of units for the Pilots of **Helicopters**
The 3rd entry defines the type of units for the leader of **Infantry** groups
The rest of the group will be generated randomly (DAC will get everything which is defined behind the 3rd entry)

Tip: You can enter unit classes even repeatedly inside a Pool (as one can see above).
This increases the likelihood that those unit types will be generated.

[DAC_Config_Behaviour]

In this configuration script different basic types of behavior can be stored and loaded for each zone separetely. "Basic behavior" means how the generated units of a zone behave when they execute their waypoints without enemy contact.

Here is a "block" of this script…

```
case 1:
{
    _setSkill   = [0.3,0.9];
    _setCombat  = ["white","yellow"];
    _setBehav   = ["safe","aware"];
    _setSpeed   = ["normal","full"];
    _setForm    = ["line","vee","column"];
    _setFleeing = [0,50];
    _setHeliVal = [45,100,0.7,1];
    _setPause   = [[3,5],[3,5],[5,10],[20,30,5,5],[0,1],[1,5]];
    _setBldgBeh = [0,50,120,600];
    _setPatrol  = ["25 + (100 * (skill _leader))","(30 + (random 30)) + ((skill _leader) * 50)"];
    _setSearch  = ["25 + (200 * (skill _leader))","(30 + (random 60)) + ((skill _leader) * 50)"];
    _setSupport = [1,2];
    _setJoin    = 2;
    _setEmpVeh  = [[100,50],[100,80]];
};
```

Here is a DAC scriptcall which feeds the units from the zone **z1** with **behaviour configuration 1**:

["z1",[1,0,0,300,200],[5,3,30,8],[ ],[ ],[ ],[0,0,1,0]]exec "DAC\Scripts\DAC_Init_Zone.sqs"

The settings are as following:

_setSkill	[0.3,0.9] This array determines the range of the skill value with which the units in the zone will be generated. **1**st Entry = the minimal skill of the units in this zone. **2**nd Entry = the maximum skill of the units in this zone.
_setCombat	["white","yellow"] This array can be used with the possible combat mode values and needs to be filled with at least one entry. The units then switch to a combat mode per random at any of their waypoints. If only one entry is given, only this value is used.
_setBehav	["aware",“combat“] This array can be used with the possible behaviour mode values and needs to be filled with at least one entry. The units then switch to a behaviour mode per random at any of their waypoints. If only one entry is given, only this value is used.
_setSpeed	["normal","full"] This array can be used with the possible speed mode values and needs to be filled with at least one entry. The units then switch to a speed mode per random at any of their waypoints. If only one entry is given, only this value is used.

_setForm	["line","vee","column"] This array can be used with the possible formation values and needs to be filled with at least one entry. The units then switch to a formation per random at any of their waypoints. If only one entry is given, only this value is used.
_setFleeing	[0,50] These parameters determine the fleeing behavior and the search for cover of the units. **1**st entry = fleeing behavior. The value must be between 0 and 1. **2**nd entry = coverage search. The value of the radius in which the units will search for cover (0 = disabled).
_setHeliVal	[45,100,0.7,0] This array determines the altitude and the takeoff behavior of helicopter units depending on the weather. It is also possible that an "HeliHEmpty" can be created at any helicopter starting position. This means that the helicopters will land exactly at their starting point. Otherwise they look for a suitable landing position, mostly in the vicinity of its starting position.. **1**st entry = minimum altitude for helicopters in the zone **2**nd entry = maximum altitude for helicopters in the zone **3**rd entry = maximum weather value when helicopters will start, otherwise they land prematurely **4**th entry = generate "HeliHEmpty" OFF (0) / ON (1)
_setPause	[[3,5],[3,5],[5,10],[20,30,5,5],[3,5],[1,30]] In this array you can define a pause time for the various units they will wait at each of their waypoints. **1**st array = waypoint pause for **infantry** [minumum + random] **2**nd array = waypoint pause for **wheeled vehicles** [minumum + random] **3**rd array = waypoint pause for **tracked vehicles** [minumum + random] **4**th array = waypoint pause for **helicopters** and (if available) for the **cargo group** [minumum + random] for **helicopters** on the ground [random] for **helicopters** in the air [random] for the **cargo group** (when on patrol) **5**th array = waypoint pause for the camp group [minumum + random] **6**th array = delay for all units in the zone [minumum + random] If you want the units from a zone not to move out directly to their waypoints you can set up a delay time. For example, [1,30] means that each group from the zone moves out with a delay of 1 to 31 seconds. [10,0] would mean that each group moves out after 10 seconds.
_setBldgBeh	[2,50,120,600] Here you determine how much infantry groups can occupy buildings. If you don't want to use this feature, you need to set the **1**st value to **0**. **1**st entry = maximum number of units from the group, which can occupy a building **2**nd entry = radius of the building detection **3**rd entry = time how long the units stay in a building **4**th entry = time before a building can be entered again a second time The example above would also mean that 2 units of a group occupy a building, if a suitable building in a radius of 50 meters was found. The units hold it for 120 seconds. Then the building is not occupied for 600 seconds.. Caution, there are not many buildings / objects which work correctly. Buildings which cause problems can be excluded (see issue **Tips + Tricks**)

_setPatrol	["50","60"] The units of the category **Wheeled** will disembark at their waypoints to patrol a certain area for a certain amount of time. Here you can determine the radius and the time. **1**st entry = max radius for the patrol depending from the vehicle of the leader **2**nd entry = time a patrol will last, until the group will board their vehicle again to continue their waypoints Attention, the values in the array must be entered as a string. The example above means: the group will patrol for 60 seconds in a radius of 50m and then board their vehicle again. 2 more examples: ["50 + random 50","60 + random 30"] ["30 * DAC_AI_Level","60 * DAC_AI_Level"] As a special you can access the leader of a group by the variable **_leader**. Therefore the array could have the following entries (example): ["25 + (30 * (skill _leader))","(30 + (random 30)) + ((skill _leader) * 50)"] or ["25 + (10 * (count units (group _leader)))","(20 * DAC_AI_Level) + ((skill _leader) * 50)"]
_setSearch	["100","120"] Units from the categories **infantry** + **wheeled vehicles** will search the area for enemy units when they lost contact with them. The handling of the parameters is exactly the same as for **_setPatrol**. **1**st entry = max. search radius for the group **2**nd entry = time a search will last, before the group will continue their normal waypoints
_setSupport	[1,2] This setting controls the behavior of the support units on the one hand, and on the other the DAC artillery request is activated / deactivated here. If you want to make sure that the zones' units which support units in other zones don't leave, you have to set the first value to **0**. This ensures that a zone doesn't run empty because all groups are on a support mission ;-) **1**st entry = zone groups are free to support (0 = no, 1 = yes) **2**nd entry = activates the artillery request in the zone (0 = disabled) To activate artillery requests by infantry groups in the zone the **2**nd value has to be a valid number from the **DAC_Config_Arti**.
_setJoin	2 This parameter determines, from which groupsize a group dissolves and joins the nearest group. This behavior can lead to a respawn of a new group when a camp is available.
_setEmpVeh	[[150,100],[100,50]] These arrays enable the boarding of empty vehicles and guns. If you don't want to use one of these characteristics in the zone, you have to set the **1**st value to **0**, otherwise as follows: **1**st Array = radius of the detection of empty vehicles to board; probability % **2**nd Array = radius of the detection of empty guns in order to board; probability % Which vehicle types can be boarded, will be determined by the setting up **DAC_VehAllowed** (page 22). Which guns can be boarded, will be determined by setting up **DAC_GunAllowed** (page 22).

[DAC_Config_Waypoints]

The **DAC_Config_Waypoints** holds all the parameters, which are needed to generate waypoints.
Here again any number of "blocks" is possible. Each block contains all the parameters for all types of waypoints.

Default is always the configuration **0** (case 0). If necessary you can equip any zone with its own configuration (see issue **The choice of configuration** on page 7).

The picture below shows you a "block" from the DAC_Config_Waypoints:

```
case 0: {
            //------------------ #Sol----#Veh----#Tan----#Air----#Camp----;
            _CheckRadius1    = [ 5,      10,     10,     20,     20      ];
            _CheckRadius2    = [ 10,     30,     30,     40,     60      ];
            _checkAreaH      = [ 20,     15,     15,     10,     10      ];
            _checkMaxH       = [ 500,    500,    500,    500,    500     ];
            _checkMinH       = [ 5,      5,      5,      5,      5       ];
            _checkNear       = [ 0,      0,      0,      100,    200     ];
            _checkObjH1      = [ 0.5,    0.5,    0.5,    0.2,    0.2     ];
            _checkObjH2      = [ 30,     20,     10,     5,      4       ];
            _checkCount      = [ 200,    200,    200,    500,    1500    ];
            _checkResol      = [ 45,     36,     36,     12,     12      ];

            _TempWPArray  = call compile format["DAC_WP_Pool_%1",(_DACTemp select _DAC_WP_Typ)];
        };
```

Example: For zone **z6** the waypoint configuration **2** has been choosen (this parameter is optional)

["z6",[1,0,0,300,200],[8,2,80,12],[ ],[ ],[ ],[0,0,0,0,2]]exec"DAC\Scripts\DAC_Init_Zone.sqs"

I will only address the parameters roughly. Some are self explainatory.

_checkRadius1	The critical radius, in which no or only very small objects may exist
_checkRadius2	The uncritical radius, in which only objects of a certain size / height may exist
_checkAreaH	The maximum difference in altitude within _checkRadius2
_checkMaxH	The maximum height where waypoints can be generated
_checkMinH	The minimum altitude where waypoints can be generated
_checkNear	The minimum distance for waypoints
_checkObjH1	The maximum size of objects for the critical area (_checkRadius1)
_checkObjH2	The maximum size of objects for the uncritical area (_checkRadius2)
_checkCount	The maximum number of attempts to find a waypoint before DAC bogs
_checkResol	The value for the raster-scan resolution at the height values (smaller value = higher load)

[DAC_Config_Camp]

On the first look the **DAC_Config_Camp** looks very complicated - and it is indeed ;-)
After all, all the DAC camps are defined there in their 7 possible stages of expansion.

For example the picture below shows a camp configuration which uses all stages of expansion, that's why it looks somewhat chaotic. If you have a look at the script **DAC_Config_Camp.sqf** you will find far more uncomplicated camp configurations, too. Apart from the array **_campObjInit** the arrays correspond to exactly one stage of expansion.

["z6",[1,0,0,300,200],[8,2,80,12],[ ],[ ],[ ],[0,0,0,6,2]]exec"DAC\Scripts\DAC_Init_Zone.sqs"

```
case 6:
{
    _campBasic     = ["FlagCarrierSouth",["Fire",5,10,0],["CampEmpty",10,0,0],["Logic",0,15,0],0];
    _campAmmo      = [["AmmoBoxWest",16,10,0],["WeaponBoxWest",16,5,0],["SpecialBoxWest",16,0,0]];
    _campStatic    = [["M2StaticMG",7,42,0,"Soldierwb"],["M2StaticMG",23,42,0,"Soldierwb"]];
    _campAddUnit   = ["Soldierwsniper","Soldierwsniper","Soldierwsniper","Soldierwsniper"];
    _campUserObj   = [["Logic",20,55,315],["ShedBig",25,20,90],["HMMWV",0,32,90],["Truck5tOpen",25,5,0]];
    _campRandomObj = [[["AAR0038",4,80],["AAR0041",1,80],["AAPL068",1,80],["AAPL076",2,80]],50,1,30,15];
    _campWall      = ["AAH0393",[-3,40],[40,45,0],[3,0,0,3],[1,0.8],[0,0]];
    _campObjInit   = [[],[],[],[],[],[],["[_x]execVM ""scripttest.sqf"""]];
};
```

_campBasic = ["FlagCarrierNorth",["Fire",5,10,0],["CampEast",7,5,0],["Logic",0,15,0],0];

This array is the basis and must not be extended or reduced.
The present parameters may be changed however.

The first entry is always the flag. It is the reference object for all other objects which will be generated in the camp.
It will be placed exactly on the positions that DAC has found for the camp.
Then there are 3 arrays which contain 4 parameters each.

Important: The last object should be of type "Logic" (like you can see above).
It is the position for respawns where new units will be generated in the camp.

The 4 parameters have the following meaning:

["CampEast ",7,5,0]

- The object type that will be generated
- The x-variation from the reference object (7 = 7m to the east/ -7 = 7m west of the flag)
- The y-variation from the reference object (5 = 5m to the north/ -7 = 5m south of the flag)
- The direction of the object (for example: 0 = random direction, 90 = east, 180 = south, 270 = west, 360 = north)

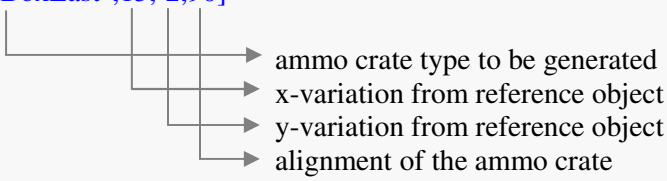
_campBasic = ["FlagCarrierNorth",["Fire",5,10,0],["CampEast",7,0,0],["Logic",0,15,0],0];

The last parameter in the array _campBasic controls the "self-maintenance" (applies to only 3 basis objects)
0 = self-maintenance deactivated, >0 = seconds until objects are repaired again.
Info: it happens that respawned vehicles destroy these basis objects. Therefore this parameter

_campAmmo = [["AmmoBoxEast",15,-2,90],["WeaponBoxEast",20,-2,90],["SpecialBoxEast",30,30,0]];

This array is for creating one or more ammo crates inside the camp.
Each array consist of 4 parameters again which work similar to the basis objects:

["AmmoBoxEast",15,-2,90]



_campStatic = [["D30",-3,18,270,"Soldiereb"],["D30",5,38,0,"Soldiereb"],["D30",38,25,90,"Soldiereb"]];

This array defines on demand any number of static weapons.
That can be for example a MG-nest or an AA- or artillery gun. Precondition for the statics is that there has to be a gunner position. You can also use vehicles with gunner positions; the driver slot will then intentionally not be manned.

The arrays have a further parameter. This value defines the unit type for the gunner slot.
The rest is structured analogue to the others.

_campAddUnit = ["Soldiereaa","Soldiereaa","Soldieresniper"];

Here you can define additional infantry units which will be generated in the camp group.
Thereby you can assure that for example enough AA units are present inside a camp, because the camp group as all the others is generated randomly and there is no warranty for certain unit types. Please be careful to use appropriate units (with respect to the side).

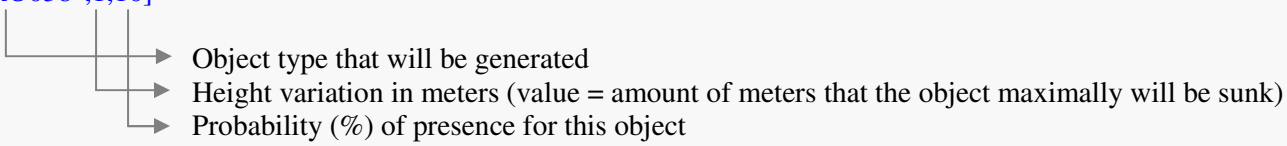
_campUserObj = [["UAZ",0,32,90],["UAZ",0,25,90],["Ural",30,0,270]];

This array contains any amount of custom objects of any type. That can be empty vehicles or certain buildings or facilities like tents or guard towers. The array's structure is as supplied before: object type, x-pos, y-pos, azimuth

_campRandomObj = [[["AAPL048",1,60],["AARO038",1,10],["AARO041",1,10],["AAPL068",3,20]],50,1,50,10];

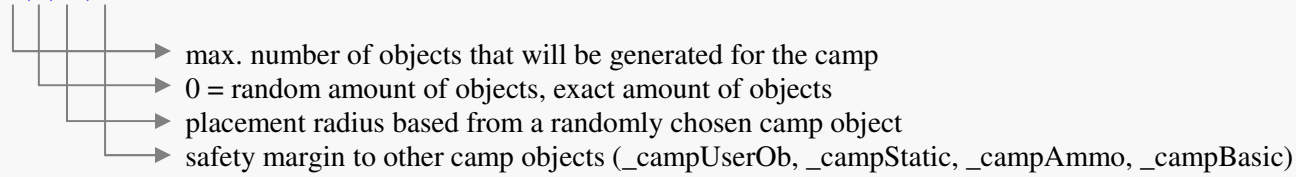
If you want to provide some natural cover for the camp and the units you can take care of it here.
Objects like trees, bushes, rocks and so on are concerned.
The array can store multiple object arrays. Each of these object arrays needs 3 values:

["AARO038",1,10]



_campRandomObj = [[["AAPL048",1,60],["AARO038",1,10],["AARO041",1,10],["AAPL068",3,20]],50,1,50,10];
The 4 values at the end of the array do the following:

50,1,50,10



_campWall = ["FenceWood",[-2,35],[40,40,1],[7,0,0,4],[1,0.1],[1,90]];

This defines the surrounding for the camp: walls and sandbags that will be placed around the camp.

_campWall = ["**FenceWood**",[-2,35],[40,40,1],[7,0,0,4],[1,0.1],[1,90]];

The first value determines the object type that will be used for the surrounding.

_campWall = ["FenceWood",[**-2,35**],[40,40,1],[7,0,0,4],[1,0.1],[1,90]];

These values determine the starting position [x,y] from the flag.
The objects will be generated clock wise from there. In this example 2 m west and 35 m north of the flag.

_campWall = ["FenceWood",[-2,35],[**40,40,1**],[7,0,0,4],[1,0.1],[1,90]];

This determines the size for the surrounding. The first value defines the x-dimension to the east, the second value defines the y-dimension to the south (meters). The third value determines which object axis DAC will measure:
0 = x-axis; **1** = y-axis.. This will calculate the exact object size. This is important for DAC to be able to place the objects right. Just try it out, there are only these 2 possibilities ;-)

_campWall = ["FenceWood",[-2,35],[40,40,1],[**7,0,0,4**],[1,0.1],[1,90]];

Here you can define exits. The camp surrounding always has 4 sides; the north-, east-, south- and west-side. These 4 sides are represented in this array. Each value defines how many segments shall be left out at the respective side (always from the middle). Value 0 means that the surrounding will be closed on this side.

_campWall = ["FenceWood",[-2,35],[40,40,1],[7,0,0,4],[**1,0.1**],[1,90]];

Here you can define how much the objects shall be sunk into the ground.
It can be a static value or a random value from a certain interval.
The first parameter controls the sinking: 0 = random value; 1 = exact value
The second parameter defines how much the object will be sunk down maximally.

_campWall = ["FenceWood",[-2,35],[40,40,1],[7,0,0,4],[1,0.1],[**1,90**]];

Only the alignment for the objects you can influence with this array lacks:
Value 1 defines whether the objects have a static facing or if their facing shall vary randomly inside a given interval:
0 = static direction; > 0 = value for random variation.

Value 2 determines the basic direction. Objects will basically be aligned to the north when they are generated.
Some objects face north with their “wide” side, others with the narrow side.
To be able to take this issue into account a customisation is needed.

Because the surrounding is built up in right angels only the following values make sense here: 0, 90, 180, 270.
(There are some described pictures in the attachment to understand this better)

_campObjInit = [[],[],[],[],[],[],[]];

Here you can assign a state to each object or start a script. Each empty array refers to a certain object array:

[_campBasic, _campAmmo, _campStatic, _campAddUnit, _campUserObj, _campRandomObj, _campWall]

If you want to lock the vehicles you have defined in the object array _campUserObj for example, it would look like this:

_campObjInit = [[],[],[],[],[„_x lock true“],[],[]];

The entry is in the 5th array because the _campUserObj array is the 5 th array.
Furthermore the entries have to be strings. Objects are referred to with the placeholder _x.

[DAC_Config_Arti]

The **DAC_Config_Arti** contains all the relevant parameters for an artillery request as well as the 3 conditions.
In order to make the artillery useable for the AI there has to be a zone that has a valid entry in its **DAC_Config_Behaviour** from the **DAC_Config_Arti**.

This zone for example loads the behaviour configuration **1** from the **DAC_Config_Behaviour**:.

```
["z10",[10,0,0,1000,500],[5,3,30,8],[ ],[ ],[ ],[0,0,1,0]]exec "DAC\Scripts\DAC_Init_Zone.sqs"
```

This configuration has the array **_setSupport = [1,3]** which contains the parameter for the artillery configuration.

With other words: the units from this zone are able to request artillery.
When they do this the artillery configuration 3 will be used (each zone can use a different configuration).

```
case 3:
{
    _set0 = [100,1,0,3];
    _set1 = [0,0,100,[],3,30];
    _set2 = [100,1,100,100,4,1,0];
    _set3 = [["M1Abrams",["Sh_105_HE"]]];
    _set4 = [2,5,0];
    _set5 = [20,50,0];
    _set6 = [ 5,15,0];
    _set7 = [ 1, 5,0];
    _set8 = [];
    _set9 = ["((player distance TestLogic) < 200)","Not possible"];
};
```

The above picture shows you a complete “block” of the **DAC_Config_Arti**. Each of the first 3 arrays (**marked in red**) define one condition. The rest of the 7 arrays (**marked in green**) contain the parameters for the DAC-Arti.

_set0 = [20,2,0,30] — Condition level 1

- Max. speed of the **Target-Group** for the artillery strike to be executed.
- Min. number of units in the **Target-Group** for the artillery strike to be executed.
- Min. number of vehicles in the **Target-Group** for the artillery strike to be executed.
- Min. amount of time until the next artillery strike can be called upon the **Target-Group**.

_set1 = [2,0,100,[ ],2,30] — Condition level 2

- Min. number of units in the **Call-Group** for the artillery strike to be able to be called.
- Min. average skill of the **Call-Group** for the artillery strike to be able to be called.
- Min. distance of the **Call-Group** to the **Target-Group** for the artillery strike…
- Unit types that have to be present in the **Call-Group** for the artillery strike …..
- Number of attempts to call artillery if a condition (e.g. distance) is not met.
- Time span between the attempts to call in artillery.

_set2 = [50,100,10,100,4,1,0] — Condition level 3

- Over all probability for an artillery strike
- Min. distance of the **Arti** -unit to the **Target-Group**
- Min. distance of friendly troops to the **Target-Group**
- Max. over all number of **Arti** -strikes
- Max. number of **Arti** -units per Arti-strike
- Max. number of rounds per **Arti** -unit
- Moveable **Arti** -Units ON (1) / OFF (0)

_set3 = [["T72",["Sh_105_HE"]]]

The unit type which will be used as Arti-unit
The ammunition type which will be used for an artillery strike.

Here there are multiple entries possible, e.g.:
[["T72",["Sh_105_HE"]]**,**["D30",["Sh_105_HE"]]**]**
Again for the ammunition type for fusing smoke grenades for example:
[["T72",["Sh_105_HE","Smokeshell"]]**]**

_set4 = [2,5,1];

Min. time span from artillery request to artillery execution.
Max time span from artillery request to artillery execution.
0 = skill dependency deactivated, 1 = leader skill of the group will be considered

Example: Skill = 1.0 => time frame = 2 – 5 seconds
Skill = 0.5 => time frame = 4 – 10 seconds
Skill = 0.2 => time frame = 10 – 25 seconds

_set5 = [3,10,1];

Min. variation for position declaration with **Call-Group**
Max. variation for position declaration with **Call-Group**
0 = skill dependency deactivated, 1 = leader skill of the group will be considered

Example: Skill = 1.0 => Min. variation = 3m | Max. variation = 10m
Skill = 0.5 => Min. variation = 6m | Max. variation = 20m
Skill = 0.2 => Min. variation = 15m | Max. variation = 50m

_set6 = [5,20,1];

Min. variation for position calculation of **Arti**-unit
Max. variation for position calculation of **Arti**-unit
0 = skill dependency deactivated, 1 = leader skill of the group will be considered

Example: Skill = 1.0 => Min. variation = 5m | Max. variation = 20m
Skill = 0.5 => Min. variation = 10m | Max. variation = 40m
Skill = 0.2 => Min. variation = 25m | Max. variation = 100m

_set7 = [0.1,0.5,1];

Min. break between volleys
Max break between volleys
0 = skill dependency deactivated, 1 = leader skill of the group will be considered

Example: Skill = 1.0 => time frame = 0.1 – 0.5 seconds
Skill = 0.5 => time frame = 0.2 – 1.0 seconds
Skill = 0.2 => time frame = 0.5 – 2.5 seconds

_set8 = [ ];

Name of the editor placed unit which shall be used as artillery.
Multiple entries are possible.

_set9 = [ ];

Global condition which has to be met in order to execute an artillery strike.
Please provide this condition as string.
Example: ["(({alive _x} count [unitName1, unitName2, unitName3]) > 0)"]

[DAC_Config_Events]

The **DAC_Config_Events** allows you to interfere with the DAC at certain points. Therefore the DAC provides some events, where you can start your own script for example, or query values of a group / unit. As with the other DAC configuratios, you can create any number of configurations. In principle, you can equip each zone with a different configuration.

Event		*Variable*		*Description*
Create	>	**_group**	>	After a group has been generated
ReachWP	>	**_group**	>	When a group has reached one of their waypoints
NotAliveGroup	>	**_group**	>	When a group is down
NotAliveUnit	>	**_unit**	>	When a unit is down
BeforeReduce	>	**_unit**	>	Before a group is reduced
AfterBuildUp	>	**_unit**	>	After a reduced group has been re-generated
InitVehicle	>	**_vehc**	>	After a vehicle has been generated

For every unit category (see below) actions can be specified to the corresponding events.

_Init_Unit_S	=	Category **Infantry**
_Init_Unit_V	=	Category **Wheeled**
_Init_Unit_T	=	Category **Tracked**
_Init_Unit_A	=	Category **Helicopter**
_Init_Unit_C	=	Category **Camp-Group**
_Init_Unit_H	=	Category **Helicopter-Group**
_Init_Unit_V	=	All empty vehicles + helicopter, generated by DAC

In each category you will find several arrays, which correspond to one certain event.
For example the category **Infantry**:

```
_Init_Unit_S =  [
                    [],     <=      for event Create
                    [],     <=      for event ReachWP
                    [],     <=      for event NotAliveGroup
                    [],     <=      for event NotAliveUnit
                    [],     <=      for event BeforeReduce *for infantry, camps and wheeled vehicles only*
                    [],     <=      for event AfterBuildUp *for infantry, camps and wheeled vehicles only*
                ];
```

Important: <u>All event entries have to be specified as string **!!**</u>

Here are some examples, of how such an entry should look like (possible variables are printed **bold**):

Event Create :
“{_x Setskill 1} foreach units **_group** “
“{if(format[“”%1””,typeof**_x]** == “”Soldierwsniper””) then {_x Setskill 1}} foreach units **_group** “
“{if(_x != vehicle _x) then {if(!((vehicle _x) in VehArray)) then { VehArray = VehArray + [vehicle _x]}}} foreach units **_group** “
“{[_x] execVM “”MyUnitScript.sqf””} foreach units **_group**”
“{_x addeventhandler [“”hit””,{_this spawn MyHitScript}]} foreach units **_group** “

Event ReachWP :
“{if((getdammage _x) > 0.8) then {_x commandmove (position HealthBuilding)}} foreach units **_group**“
“{if((_x distance (leader (group _x))) > 50) then {_x commandmove (position (leader (group _x)))}} foreach units **_group**“
“[**_group**] call MyGroupFunction”

Event NotAliveGroup :
“MyGroupArray = MyGroupArray - [**_group**]”

Event NotAliveUnit :
“MyUnitArray = MyUnitArray - [**_unit**]”
“MyUnitCount = MyUnitCount - 1“

Event BeforeReduce :
“MyUnitArray = MyUnitArray - [**_unit**]”
“MyUnitCount = MyUnitCount - 1“

Event AfterBuildUp :
“MyUnitArray = MyUnitArray + [**_unit**]”
“**_unit** addeventhandler [“”hit””,{_this spawn MyHitScript}]“
“MyUnitCount = MyUnitCount + 1“

If you want to start a script at each waypoint for infantry groups, which gives you some data from the groups of a zone, you must first enter the configuration number from the DAC_Config_Events in the scriptcall of the zone and the corresponding entry must be present in the configuration (as a string).

This would be a scriptcall, which loads the configuration no. **1** for zone “z1” from the **DAC_Config_Events**:
(By default only number 1 is present. Additional numbers can be created by you)

```
["z1",[1,0,1,400,200],[5,3,50,8],[],[],[],[1,1,0,6]]exec "DAC\Scripts\DAC_Init_Zone.sqs"
```

Among this number in the **DAC_Config_Events.sqf** you have to enter the corresponding entry for the scriptcall in the array for infantry:

```
case 1:
	{
		_Init_Unit_S =	[
					[],
					[“[_group] execVM “”NameOfYourScript.sqf”””],
					[],
					[],
					[],
					[],
				];
		_Init_Unit_V =	[
					[],
					[],
					[],
					[],
					[],
					[],
				];
		_Init_Unit_T =	[
					[],
					[],
					[],
					[],
				];
		_Init_Unit_A =	[
					[],
					[],
					[],
					[],
				];
		_Init_Unit_C =	[
					[“{_x Setskill 1} foreach units _group “],
					[],
					[],
					[],
					[],
					[],
				];
		_Init_Unit_H =	[
					[],
					[],
					[],
					[],
				];
		_Init_Vehicle =	[
					[],
					[“[_vehc] execVM “”NameOfYourScript.sqf”””],
					[]
				];
	};
```

Infantry
event Create
event ReachWP
event NotAliveGroup
event NotAliveUnit
event BeforeReduce
event AfterBuildUp

Wheeled vehicles
event Create
event ReachWP
event NotAliveGroup
event NotAliveUnit
event BeforeReduce
event AfterBuildUp

Tracked vehicles
event Create
event ReachWP
event NotAliveGroup
event NotAliveUnit

Helicopter crew
event Create
event ReachWP
event NotAliveGroup
event NotAliveUnit

Campgroup
event Create
event ReachWP
event NotAliveGroup
event NotAliveUnit
event BeforeReduce
event AfterBuildUp

Helicoptergroup
event Create
event ReachWP
event NotAliveGroup
event NotAliveUnit

Wheel., Track., Heli.
event InitVehicle
event InitVehicle
event InitVehicle

[DAC_Config_Marker]

The **DAC_Config_Marker** contains all parameters which are needed to generate the DAC-markers. Those markers can be activated or even deactivated as often as one wants. To activate the DAC-marker, you only need to enter a valid number out of the **DAC_Config_Marker** into the **DAC_Config_Creator** :

DAC_Marker = 2 would load the marker configuration 2.

You only need to tell the DAC to deactivate the DAC-marker by using following order: **DAC_Marker = 0**
You can do this by using a script or even a trigger.

A block in the **DAC_Config_Marker** is built as follows:

```
case 2: {
            _setShowZones       = 2;
            _setShowWPs         = 1;
            _setShowUnit        = [1,1,1,1];
            _setGroupType       = 1;
            _setMarkerText      = ["format[""<--- %1 (%2)"",_group,count units _group]"];
            _setMarkerDel       = 0;
            _setMarkerRefresh   = [1,1];
            _setSizeWpLine      = [0.5,3];
            _setSizeLeaderLine  = 0.5;
            _setSizeZoneLine    = 0;
            _setSizeCampLine    = 2;
            _setSizeZoneBorder  = 6;
            _setArtiMarker      = 1;
            _setCampMarker      = 2;
            _setSideColor       = [
                                    "ColorRed",
                                    "ColorBlue",
                                    "ColorYellow",
                                    "ColorGreen",
                                    "ColorWhite",
                                    "ColorBlack","ColorGreen"
                                  ];
```

I only will describe the single entries here shortly:

_setShowZones	zones marker: **0** = deactivated / **1** = only while initialization / **2** = display all the time
_setShowWPs	waypoint marker: **0** = deactivated / **1** = only while initialization / **2** = display all the time
_setShowUnit	units Marker: **0** = deactivated / **1** = activated for side [West,east,RACS,civilian]
_setGroupTyp	**0** = one marker per group / **1** = marker for each unit per group
_setMarkerText	Marker Text: formated Text as String (as shown above). Variable **_group** useable
_setMarkerDel	Unit-Marker: **0** = do not delete / **1**= delete if unit got killed / **2** = delete if units got deleted
_setMarkerRefresh	The Time which a marker need to update itself [unit Marker, other Marker]
_setSizeWpLine	waypoint-marker: [thickness of lines, size of waypoints], thickness of lines = 0 deactivate the wp marker
_setSizeLeaderLine	Leader-connection-lines: thickness of lines, thickness of lines = 0 deactivate the Leader-connection-lines
_setSizeZoneLine	Zones-connection-lines: thickness of lines, thickness of lines = 0 deactivate the zones-connection-lines
_setSizeCampLine	Camp-connection-lines: thickness of lines, thickness of lines = 0 deactivate the Camp-connection-lines
_setSizeZoneBorder	Thickness of lines Zone borderlines
_setArtiMarker	Artillery-Marker : **0** = deactivated / **1** = activated
_setCampMarker	Camp-Marker : **0** = deactivated / **1** = minimum (Camp-Objects only) / **2** = maximum (all Objects)
_setSideColor	The marker colors: [East, West, RACS, Civil, neutral, waypoint-lines, waypoints]]

The definition of the actual marker which have to be used are shown in the example script below: **_setMarkerClass**

```
_setMarkerClass     = [
                        [
                            ["STATICWEAPON","Dot",[0.4,0.4],0]
                        ],
                        [
                            ["Man","ARMA_MARKER_Man",[0.4,0.4],1],
                            ["SquadLeaderE","ARMA_MARKER_Officer",[0.4,0.4],1],
                            ["SquadLeaderW","ARMA_MARKER_Officer",[0.4,0.4],1],
                            ["SquadLeaderG","ARMA_MARKER_Officer",[0.4,0.4],1],
                            ["OfficerE","ARMA_MARKER_Officer",[0.4,0.4],1],
                            ["OfficerW","ARMA_MARKER_Officer",[0.4,0.4],1],
                            ["OfficerG","ARMA_MARKER_Officer",[0.4,0.4],1],
                            ["SoldierESniper","ARMA_MARKER_SpecOp",[0.4,0.4],1],
                            ["SoldierWSniper","ARMA_MARKER_SpecOp",[0.4,0.4],1],
                            ["SoldierGSniper","ARMA_MARKER_SpecOp",[0.4,0.4],1],
                            ["SoldierEMedic","ARMA_MARKER_Medic",[0.4,0.4],1],
                            ["SoldierWMedic","ARMA_MARKER_Medic",[0.4,0.4],1],
                            ["SoldierGMedic","ARMA_MARKER_Medic",[0.4,0.4],1]
                        ],
                        [
                            ["Car","ARMA_MARKER_UAZ",[0.5,0.5],1],
                            ["Truck","ARMA_MARKER_TruckMG",[0.5,0.5],1]
                        ],
                        [
                            ["Tank","ARMA_MARKER_T72",[0.5,0.5],1],
                            ["APC","ARMA_MARKER_M113",[0.5,0.5],1]
                        ],
                        [
                            ["Motorcycle","Dot",[0.5,0.5],1]
                        ],
                        [
                            ["Air","Dot",[0.5,0.5],1],
                            ["Helicopter","ARMA_MARKER_Cobra",[0.5,0.5],1],
                            ["Plane","ARMA_MARKER_Harrier",[0.5,0.5],1],
                            ["ParachuteBase","Dot",[0.6,0.2],1]
                        ],
                        [
                            ["Ship","ARMA_MARKER_Gunboat",[0.5,0.5],1]
                        ],
                        [
                            ["Other","Dot",[0.6,0.6],1]
                        ]
                    ];
```

The image above explains you the 8 paramounted unit classes (which are marked in **orange**), which have to be exist basically. It's also possible to define further classes below that paramounted unit class. These new classes would be generated automatically if one of these units matches with one of the given classes
(as one can see above).

If it's not possible to classify that unit inside their class, so the paramounted class will be used automatically.
If it's not possible to classify a unit with one of the paramounted ones, so the DAC will use the settings which are defined in "**Other**".

Every single entry in the **DAC_Config_Marker** is corresponding with an Array with 4 entries:
["unit Class","Marker Class",[Size of Markers x,y],Marker turning (0 = OFF / 1 = ON)]

Tips, tricks and rules

Please make always sure that the correct basic settings have been adjusted for a DAC zone (Trigger):
I.E.: **Square, repeatedly, present, Game logic** (check out Page 2)

The Trigger conditions needs to bet the same for all zones, without **true** and **false**.
The Trigger condition true is enough generally. But it's also possible to defer the DAC initialization for all zones by using following entry: **Time > 15.**
DAC will run up only when the mission has been started 15 seconds ago. Another variant would be to run up the DAC if a special condition has been set on true. But that or these condition(s) need to be entered in all used DAC zones.

It is meaningfull to run own scripts (and also intensive ones) right when the initialization of the DAC has been finished. To check out when the DAC finished you only need to ask following Variable:
`waituntil{DAC_Basic_Value == 1}` inside of sq**f**-scripts, or
`@(DAC_Basic_Value == 1)` inside of sq**s**-scripts

The **DAC_Config_Camp** contains the Configuration **8,** which enables the AI-respawn even without Camp or Camp groups. DAC is generating only one position which will basically used for one respawn only.
You can define the exact position by using a user defined waypoint.

There are actually much more tips and tricks available but I am currently very out of time to explain more :-/

User advices & expression of thanks

DAC is an **unofficial** script package. Please do not complain at BIS or their Publisher.

Attention: The using of the DAC V2.0 (Beta) takes place on own danger!

I say thank you to the whole Mapfact-Team for their nice fully support in all sections, my special thank is dedicated to:

MCPXXL For offering an MP server and for lots of MP tests, where we all had much fun.

SKH-Team For offering an MP server and for lots of MP tests hours which shows us again and again how bad we actually are ;-).

Because we tested the DAC so many hours, we could locate and fix lots of mistakes. Thanks!

A **very special thanks** is dedicated to my translators as well. The release date would be around 6 month later minimum without their help. The translation was made by:

Zwobot, Rocko, Burns and MemphisBelle … many, many thanks !!

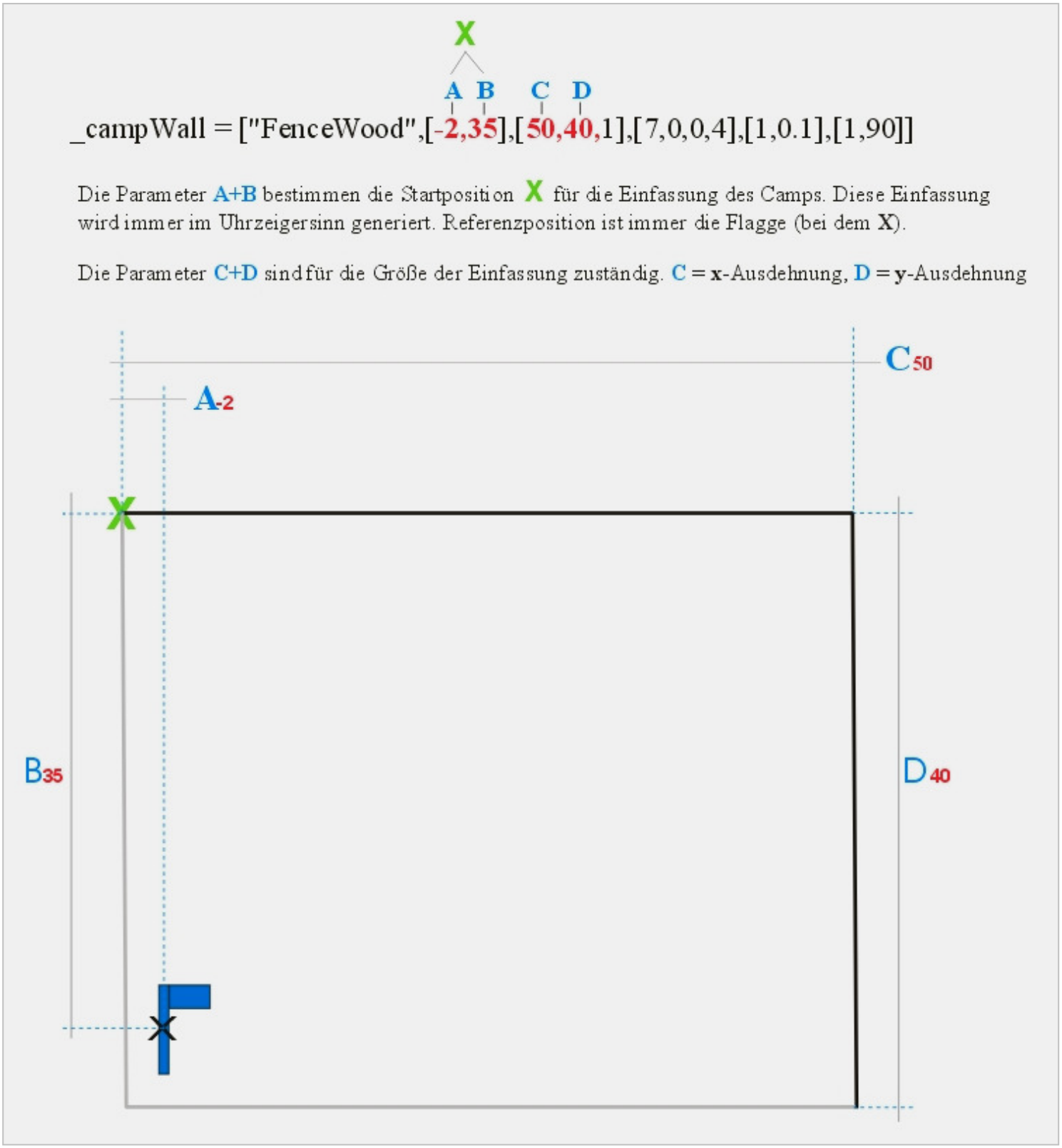
A further thank you goes to: MattRochelle, PitPines, Sickboy, Thomas Niedermeier

I apologise if I forgot someone.

Last but not least a great thank you to BIS who created a awesome game with ArmA!

Attachement:

Here the mentioned „ **figurative**“ description to the Array _campWall out of the script **DAC_Config_Camp**:



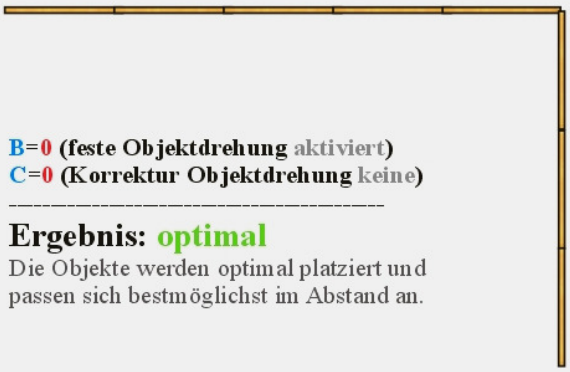
```
                                                A                   B C
_campWall = ["FenceWood",[-2,35],[50,40,0/1],[7,0,0,4],[1,0.1],[0,0]]
```

Das erste Beispiel zeigt ein Mauer-Segment,das mit der breiten Seite nach Norden zeigt. Dieser Zustand ist für uns optimal, da wir nur wenig anpassen müssen. Alle Parameter (A+B+C) können auf 0 stehen bleiben.
(Achtung, um festzustellen wie ein Objekt grundsätzlich ausgerichtet ist, mußt Du es nur einmal im Editor platzieren.)

Das Ergebnis siehst Du im linken Bild. Das rechte Bild zeigt das Ergebnis, wenn der Parameter A den Wert 1 enthält.
Die Objekte werden zwar richtig aufgebaut, jedoch mit einem viel zu geringen Abstand. Der Abstand entspricht dem y-Ausmaß.

A=0 (x-Ausmaß des Objekts wird benutzt = 2 m)



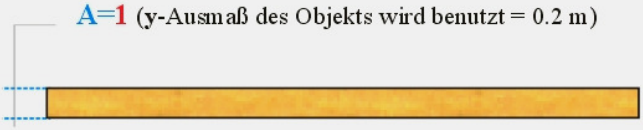


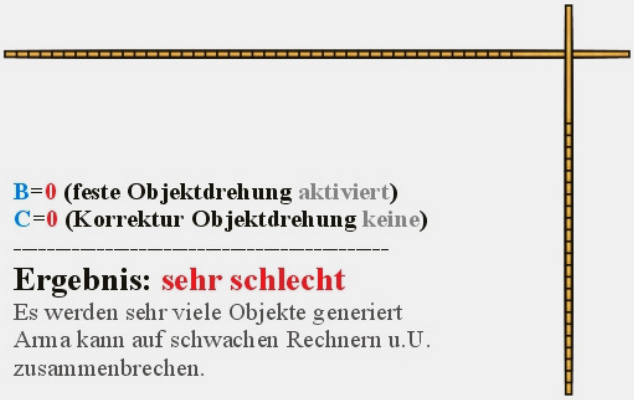
B=0 (feste Objektdrehung aktiviert)
C=0 (Korrektur Objektdrehung keine)

Ergebnis: optimal

Die Objekte werden optimal platziert und passen sich bestmöglichst im Abstand an.

A=1 (y-Ausmaß des Objekts wird benutzt = 0.2 m)





B=0 (feste Objektdrehung aktiviert)
C=0 (Korrektur Objektdrehung keine)

Ergebnis: sehr schlecht

Es werden sehr viele Objekte generiert Arma kann auf schwachen Rechnern u.U. zusammenbrechen.

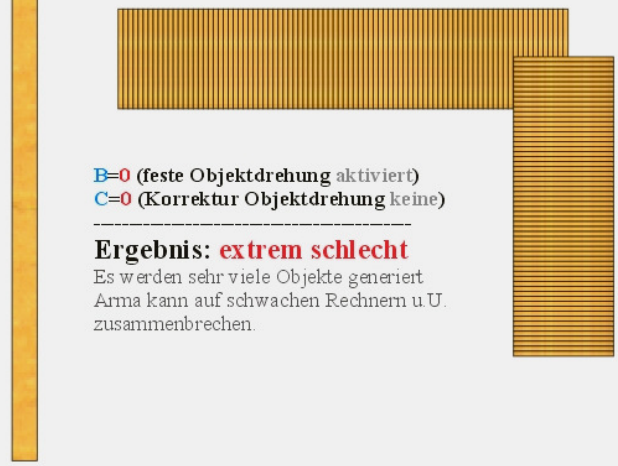
A B C

_campWall = ["FenceWood",[-2,35],[50,40,0/1],[7,0,0,4],[1,0.1],[0,0]]

Dieses Beispiel zeigt genau die gleichenEinstellungen, allerdings ist das Objekt nun mit der schmalen Seite nach Norden ausgerichtet.
Du kannst deutlich sehen, welche Auswirkungen ein solches Objekt auf das Generieren hat.

Im linken Bild werden die Objekte zwar exakt aneinander gereiht, allerdings stimmt die Objektausrichtung nicht.
Im rechten Bild ist im Prinzip der Abstand ok, nur fehlt auch hier die richtige Objektdrehung.

A=0 (**x**-Ausmaß des Objekts wird benutzt = 0.2 m)



B=0 (**feste Objektdrehung** aktiviert)
C=0 (**Korrektur Objektdrehung** keine)

Ergebnis: extrem schlecht

Es werden sehr viele Objekte generiert
Arma kann auf schwachen Rechnern u.U.
zusammenbrechen.

A=1 (**y**-Ausmaß des Objekts wird benutzt = 2 m)



B=0 (**feste Objektdrehung** aktiviert)
C=0 (**Korrektur Objektdrehung** keine)

Ergebnis: schlecht

Die Objekte werden im richtigen Abstand
generiert, aber die Objektdrehung passt nicht.

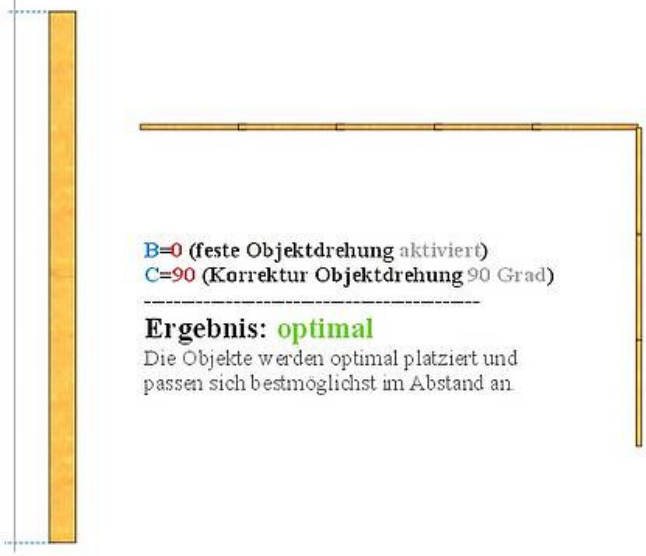
A B C

```
_campWall = ["FenceWood",[-2,35],[50,40,1],[7,0,0,4],[1,0.1],[0,90]]
```

Hier kommt nun der Parameter C zum Einsatz, damit wir auch diese Art Objekte verarbeiten können,
also Objekte, die mit der schmalen Seite nach Norden zeigen.

Das Objekt bekommt in diesem Beispiel eine permanente Drehung von 90 Grad verpasst.
Das Ergebnis siehst Du unten im linken Bild.Im rechten Bild wurde die zufällige Objektdrehung (B) aktiviert,
indem dort ein Wert von 3 Grad (gilt in beide Richtungen) eingestellt wurde.

A=1 (y-Ausmaß des Objekts wird benutzt = 2 m)

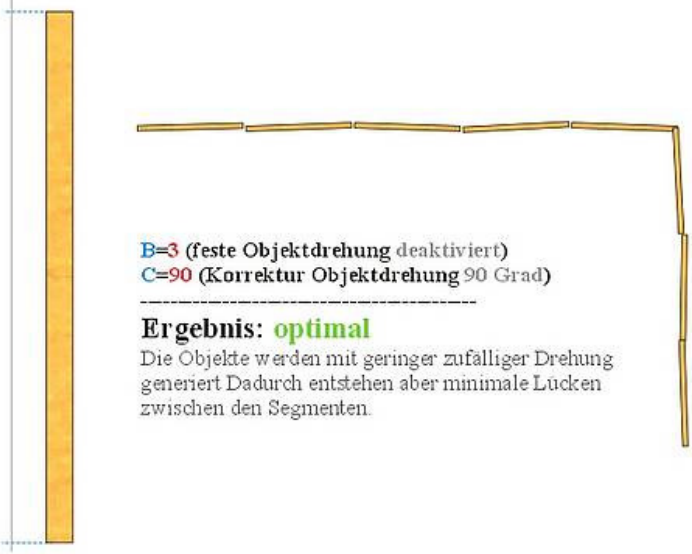


B=0 (feste Objektdrehung aktiviert)
C=90 (Korrektur Objektdrehung 90 Grad)

Ergebnis: optimal

Die Objekte werden optimal platziert und
passen sich bestmöglichst im Abstand an.

A=1 (y-Ausmaß des Objekts wird benutzt = 2 m)



B=3 (feste Objektdrehung deaktiviert)
C=90 (Korrektur Objektdrehung 90 Grad)

Ergebnis: optimal

Die Objekte werden mit geringer zufälliger Drehung
generiert Dadurch entstehen aber minimale Lücken
zwischen den Segmenten.

_campWall = ["FenceWood",[-2,35],[50,40,0],[0,0,0,0],[1,0.1],[0,0]]

Hier nochmal das Array für die Ausparungen in der Einfassung. Jeder Wert darin steht für einen Abschnitt: **Nord, Ost, Süd, West.** Der Wert selbst enspricht der Anzahl der Segmente, die entfernt werden sollen.

