

Warning, DAC has lots of functions, the documentation is accordingly comprehensive.
If you had a look at the 30-part demonstration 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 SectorFight Settings** (explanation of the parameters)
- DAC Demonstration** (contains 30 different DAC demonstration missions.)
- DAC_Source.pbo + DAC_Sound.pbo** (the standard DAC-Addons)
- DAC folder** (with all the config files, if you use the “DAC_extern” logic)
- DAC MP Missions** (contains 4 different MP mission examples)
- DAC Script** (the script version of DAC, incl. a example mission)

Theme overview:

02	Advice to the DAC + prepare the DAC	39	DAC_Config_Units
03	The DAC script requestion for a zone	40	DAC_Config_Behaviour
07	The selection of the side, the unit-, behaviour- and camp configuration	43	DAC_Config_Waypoints
09	Creating several DAC zones (copy)	44	DAC_Config_Camp
10	Linking DAC zones	47	DAC_Config_Arti
11	Create waypoint zones	49	DAC_Config_Events
12	Merge user defined waypoints	51	DAC_Config_Marker
14	Assigning logics to DAC zones	53	DAC_Config_Objects
15	Activate / deactivate DAC zones	54	DAC_Config_Sound
16	Reduce infantry groups	55	DAC_Config_Weapons
17	activate and configurate AI respawn	56	Thanksgivings
18	connect camps with zones	57	Attachment
19	adjust zone values (size, position and configuration)		
21	Creating DAC zones mid mission		
22	Deleting DAC zones mid mission		
23	The DAC Artillery		
24	The DAC Ground-support		
25	Integrate editor-groups		
26	Release DAC-groups		
27	DAC-object generator		
33	The configuration files		
34	DAC_Config_Creator		

A hint to the DAC

This version of DAC 3.1 is still a Beta version. So keep in mind that errors might occur considering the beta status. Additionally, DAC may not behave as intended.

It's pretty much important though to always make sure to feed DAC with **valid** datas only. Those ones are mostly stored in the different **configuration-files**. If problems occur so you should feed the zone constellation within your mission with default datas (DAC_intern), at first.

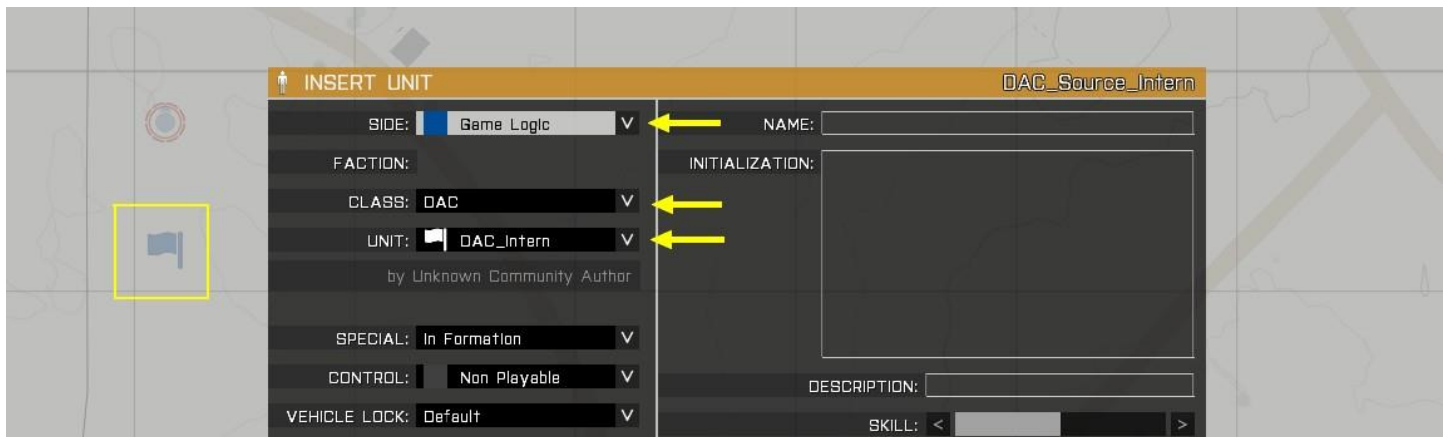
It also might be helpful to deactivate critical zones just for a test. You can do that by defining **false** in the Conditions line of each zone. The respective Zone is no more active then.

Preparing the DAC

Because the DAC in the Version 3.1 is more an Addon than a script resource, you should handle the pbo file as any other addon/Mod file. Create a Mod folder or use an already existing one (that's pretty much up to you, but I recommend to create an own DAC related Mod folder).

To get the DAC started or even initiated, you have to place a player on the map and name it "S1". Then the DAC Logic also needs to be placed on the map (insert units - F1).

Insert units (F1) >>> Game logic >>> DAC >>> DAC_intern



The Game Logic **DAC_intern** is using, as the name already explains itself, intern configuration files. The default behaviour and default units are defined right in there. This variant is always the first choice if you want to learn the DAC or even for a quick Test.

But if you are a professional already and want the total control about the configuration-files, which are providing you a high grade of Adjustment, you've got to use the **DAC_extern** Logic. But this logic requires the **DAC Folder** itself to be placed right into your Missions folder (DAC Folder is part of the Zip-file). In this case DAC is expecting all configuration files right in this folder. Which settings are adjustable will be explained in the sub item [**The DAC-Configuration_files**]. You will also learn there that there's a further, third variant to store configuration-files individually outside the main DAC resource.

OK, if you've already got a DAC_Logic placed on the map, so DAC will get initiated automatically once the mission starts. The last thing which is still missing to be complete is a DAC Zone, the area where units are about to get generated, this is the basic Job of DAC.

I just can recommend you to **check the 30 parts of the Demo**. It will be much easier then to understand, about what I am trying to explain here :).

A DAC Zone will be created just by a simple trigger. The area which is defined by the trigger is simultaneously the same area where the units will be generated. You can adjust the size, the shape or even the turning pretty much up to your needs. The picture below is representing a typical DAC Zone, as it is required.

EDIT TRIGGER

SHAPE

Ellipse **Rectangle**

AXIS A 200

AXIS B 200

ANGLE 0

TIMER

Countdown Timeout

MIN: 0

MID: 0

MAX: 0

NAME: z3

TEXT: local master zone east

TYPE: None

ACTIVATION: Game Logic

Once **Repeatedly**

Present Not present

Detected by BLUFOR Detected by OPFOR

Detected by Independent Detected by Civilians

CONDITION: time > 1

ON ACT.: fun=["z3",[3,0,0],[5,3,20,8],[],[],[0,0,0,0]] spawn DAC_Zone

ON DEA.:

CANCEL **EFFECTS** **OK**

I recommend to adjust the parameters as marked within the yellow options above. The green marked Parameters are variable and need to be adjusted up to your needs.

Name: The name of the DAC zone

Condition: This option should be the same for all DAC-zones (recommendation)
(time > 1 is regularly the best choice)

OnActivation.: The script-call, which is initiating the DAC Zone.

Notification:

If you want the DAC Zone to be initiated by a delay of e.g. 10 seconds, because you may have several other scripts to be initiated or a cut movie is preset, so define following syntax in all DAC Zones: **time > 10**.

If you'd like to become a certain DAC Zone deactivated for special reasons (may be a Test), so enter following logical value in the Condition line: **false**

This provides the advantage that a certain Zone doesn't has to be deleted if it is not needed in any case. But you're nevertheless able to reactivate it again.

Attention

It is strictly required that all player-units will receive a name, to enable DAC to react in correct manner with this units. The names **s1** till **s10** are predefined already.

That means that the player character used in a single player mission will get named **s1** while all playable units within a Multiplayer Mission will get named **s1, s2, s3,...** and so on.

It is of course possible to change these names. But to do this the configuration-file called "DAC_Config_Creator" needs to be adjusted. The Names are defined right in this file.

But how exactly that works and which changings are also possible within this file can be read in the section [**The DAC Configuration Files**].

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],[ ],[ ],[ ],[ ],[1,1,1,1]] spawn DAC_Zone
```

First we have a look at the 4 empty arrays which I have coloured.

There the amount of groups 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],[8,2,50,8],[2,2,20,6],[5,1,30,8],[2,2,3],[1,1,1,1]] spawn DAC_Zone
```

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],[5,2,50,6],[1],[1],[1],[0,0,0,0]] spawn DAC_Zone

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],[1],[3,3,30,5],[1],[1],[0,0,0,0]] spawn DAC_Zone

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.

Regardless of the group size, DAC can only fill a maximum of one vehicle. This is dependent on the maximum crew size of the created vehicle.

The ability to produce more vehicles per group I disabled because the AI is not able to control several vehicles within one group ☹

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],[],[],[3,3,25,5],[],[0,0,0,0]] spawn DAC_Zone

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)

3. 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],[],[],[],[2,2,4],[0,0,0,0]] spawn DAC_Zone

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.

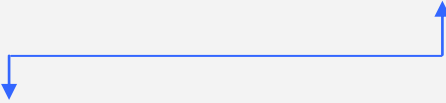
The group size is only responsed, if the heli has cargo capacities. A AH64 e.g.won't take more units. Just the pilots will be generated.

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.

The selection of the Side, the unit-, behaviour- and Camp-configuration

These characteristics are merged in here, because they're standing together in the Array of the script call, anyway. This Array is located right at the end of the script call (see blue marker).

```
["z1",[1,0,0,0],[8,2,50,8],[ ],[ ],[ ],[0,0,0,0,0]] spawn DAC_Zone
```



Parameter 1 = the selection of the Side for the unit within this Zone

Parameter 2 = the selection of the unit configuration for this Zone

Parameter 3 = the selection of the behaviour-configuration for this Zone

Parameter 4 = the selection of the Camp-configuration for this Zone

Parameter 5 = the selection of the Waypoint-configuration for this Zone (optional)

To select the Side

```
["z1",[1,0,0,0],[8,2,50,8],[ ],[ ],[ ],[0,0,0,0,0]] spawn DAC_Zone
```

There are only 4 adjustment settings for the selection of the Side (apparently, cause there are only 4 Parties...;)

0 = East | **1** = West | **2** = Resistance | **3** = Civilian

So eastern units will be generated in the example above, as the parameters were defined with **0**.

The choice of the unit configuration

```
["z1",[1,0,0,0],[8,2,50,8],[ ],[ ],[ ],[0,0,0,0,0]] spawn DAC_Zone
```

This parameter defines the kind of unit which are about to get generated in the Zone. If you're using the **DAC_intern** Logic, so following unit configuration are selectable (standard units):

0	=	East (CSAT)		2	=	Independent (AAF)
1	=	West (NATO)		3	=	Civilian (Civilians)

Attention

There must always be the right resp. most fitting Site selected for any unit configuration!!!

With other words: **Please don't create West units on Eastern Side a.s.o.**

That might cause uncontrollable (but funny) AI reactions.

The unit configurations are defined in the **DAC_config_Units** file.

Its basically possible to load as many configurations as wanted right in this configuration file, while each other DAC Zone could handle total different configuration files as well.

How exactly that works can be read in the Section [**The DAC-Configuration-files**]

The choice of the behaviour Configuration

```
["z1",[1,0,0,0],[8,2,50,8],[ ],[ ],[ ],[1,1,1,1,0]] spawn DAC_Zone
```

This parameter defines which behaviour configuration will be loaded for which Zone.
The behaviour configurations are located in the **DAC_Cnfig_Behaviour** file.
The parameters which are definable in a behaviour configuration is again a special section which can be found in the Theme [**The DAC_Configuration-files**].

If you're using the Game Logic **DAC_intern**, so following behaviour configurations are selectable:

0	=	Basic behaviour for Side East
1	=	Basic behaviour for Side West

2	=	Basic behaviour for Side Resistance
3	=	Basic behaviour for Side Civilian

It's again possible to add as many entries as one wants in the behaviour configuration.
So it's possible to define a different configuration for each Zone individually.

The choice of Camp configurations

```
["z1",[1,0,0,0],[8,2,50,8],[ ],[ ],[ ],[1,1,1,1,0]] spawn DAC_Zone
```

The last parameter is responsible for the Camp configuration. So basically it is only important if minimum one Camp is present in the Zone (or about to be generated), apart from that this parameter keeps unconsidered.

As same as for the behaviour configuration so a number needs to be entered here again, which will load the respective configuration out of the **DAC_config_Camps**. The pretty much complex structure is defined in the camp-configuration.

The choice of the Waypoint configuration

```
["z1",[1,0,0,0],[8,2,50,8],[ ],[ ],[ ],[1,1,1,1,0]] spawn DAC_Zone
```

This parameter is optional and don't has to be defined coercively. It's setting up the waypoint configuration for each Zone. if this parameter is not set, so the value 0 will be set as default.

The parameter „NoReturn“

```
["z1",[1,0,0,0],[8,2,50,8],[ ],[ ],[ ],[1,1,1,1,0]] spawn DAC_Zone
```

This parameter is optional and causes the groups to leave their master zone and then they will never return back.

However, this parameter requires at least one waypoint zone, which must be linked to the master zone, so that the units have a reason to leave their master zone.

If you enter a value > **0** (max. 100), then this value corresponds to a probability of the groups can have waypoints in their master zone.

If you apply this parameter to a heli zone, then he causes that the helicopters only return back to their master, to land there.

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 script call
The script call from the zones then could look in such a way:

```
["z1",[1,0,0],[3,2,20,5],[ ],[ ],[ ],[0,0,0,0]] spawn DAC_Zone  
["z2",[1,0,0],[3,2,20,5],[ ],[ ],[ ],[0,0,0,0]] spawn DAC_Zone  
["z3",[1,0,0],[3,2,20,5],[ ],[ ],[ ],[0,0,0,0]] spawn DAC_Zone
```



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],[3,2,20,5],[ ],[ ],[ ],[0,0,0,0]] spawn DAC_Zone  
["z2",[2,0,0],[3,2,20,5],[ ],[ ],[ ],[0,0,0,0]] spawn DAC_Zone  
["z3",[3,0,0],[3,2,20,5],[ ],[ ],[ ],[0,0,0,0]] spawn DAC_Zone
```



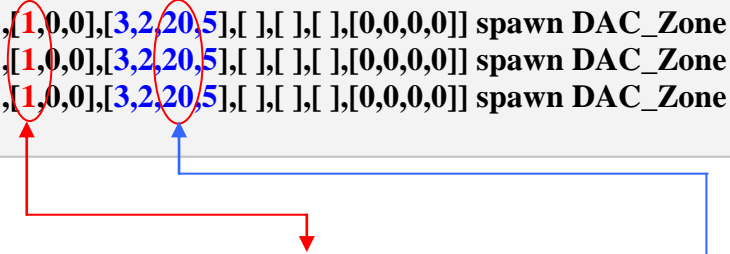
That is the zones' **ID-Nr**

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],[3,2,20,5],[ ],[ ],[ ],[0,0,0,0]] spawn DAC_Zone  
["z2",[1,0,0],[3,2,20,5],[ ],[ ],[ ],[0,0,0,0]] spawn DAC_Zone  
["z3",[1,0,0],[3,2,20,5],[ ],[ ],[ ],[0,0,0,0]] spawn DAC_Zone
```



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

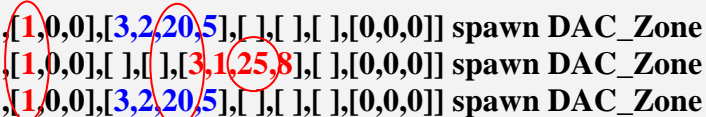
That does mean each of those 9 **infantry** groups has access to **60** waypoints. Which waypoints will be used by groups is chosen randomly.

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],[3,2,20,5],[ ],[ ],[ ],[0,0,0]] spawn DAC_Zone  
["z2",[1,0,0],[ ],[ ],[3,1,25,8],[ ],[ ],[0,0,0]] spawn DAC_Zone  
["z3",[1,0,0],[3,2,20,5],[ ],[ ],[ ],[0,0,0]] spawn DAC_Zone
```



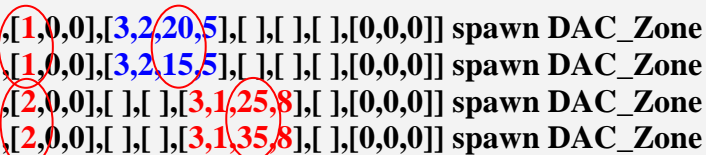
This example also shows 3 linked zones, however in the zone **z2** **tracked vehicles** are provided.

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.

Basically you don't need to link zone **z1** with zone **z2**.

```
["z1",[1,0,0],[3,2,20,5],[ ],[ ],[ ],[0,0,0]] spawn DAC_Zone  
["z2",[1,0,0],[3,2,15,5],[ ],[ ],[ ],[0,0,0]] spawn DAC_Zone  
["z3",[2,0,0],[ ],[ ],[3,1,25,8],[ ],[ ],[0,0,0]] spawn DAC_Zone  
["z4",[2,0,0],[ ],[ ],[3,1,35,8],[ ],[ ],[0,0,0]] spawn DAC_Zone
```



4 Zones named **z1** to **z4** are placed on the map. Above you see their script calls.

Zones **z1** and **z2** are linked by **ID1** → **35** waypoints at disposal

Zones **z3** and **z4** are linked by **ID2** → **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],[15,2,10,15],[ ],[ ],[ ],[0,0,0,0]] spawn DAC_Zone  
["z2",[1,0,0],[20],[ ],[ ],[ ],[0,0,0,0]] spawn DAC_Zone  
["z3",[1,0,0],[20],[ ],[ ],[ ],[0,0,0,0]] spawn DAC_Zone  
["z4",[1,0,0],[20],[ ],[ ],[ ],[0,0,0,0]] spawn DAC_Zone
```

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],[15,2,10,15],[ ],[ ],[ ],[0,0,0,0]] spawn DAC_Zone  
["z2",[1,0,0],[ ],[ ],[3,1,25,8],[ ],[0,0,0,0]] spawn DAC_Zone
```

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],[15,2,10,15],[ ],[20],[ ],[0,0,0,0]] spawn DAC_Zone  
["z2",[1,0,0],[20],[ ],[3,1,25,8],[ ],[0,0,0,0]] spawn DAC_Zone
```

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 useful, 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

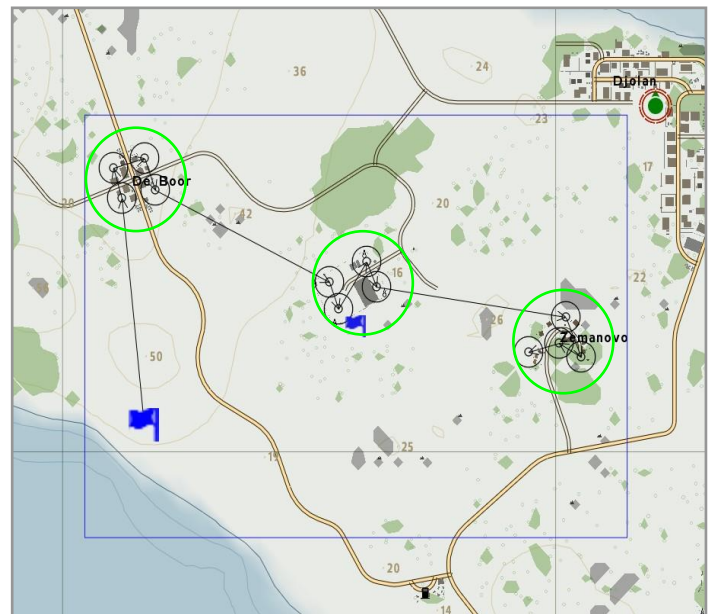
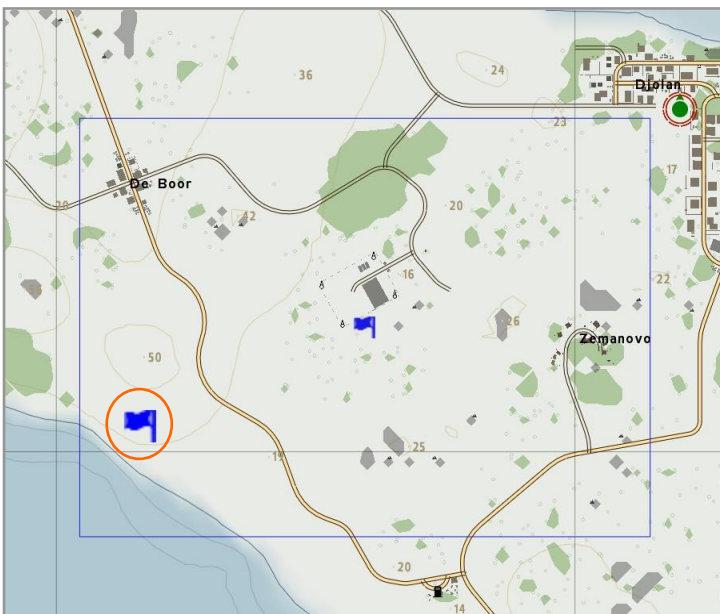
It is quite easy to build-in user defined Waypoints:

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.

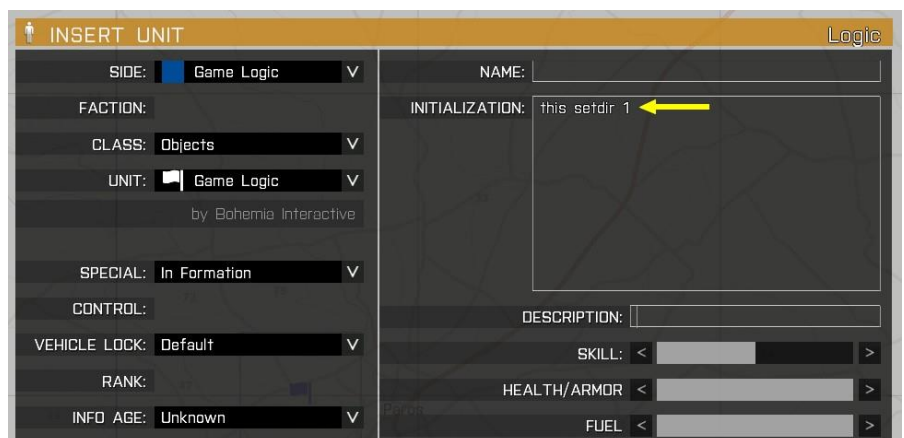
The waypoints have to be in the zone.



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**



It is also possible to **define custom waypoints in advance**.

This means, that you can place as many game logics as you want, which are not located within a DAC-zone and thus have no real purpose.

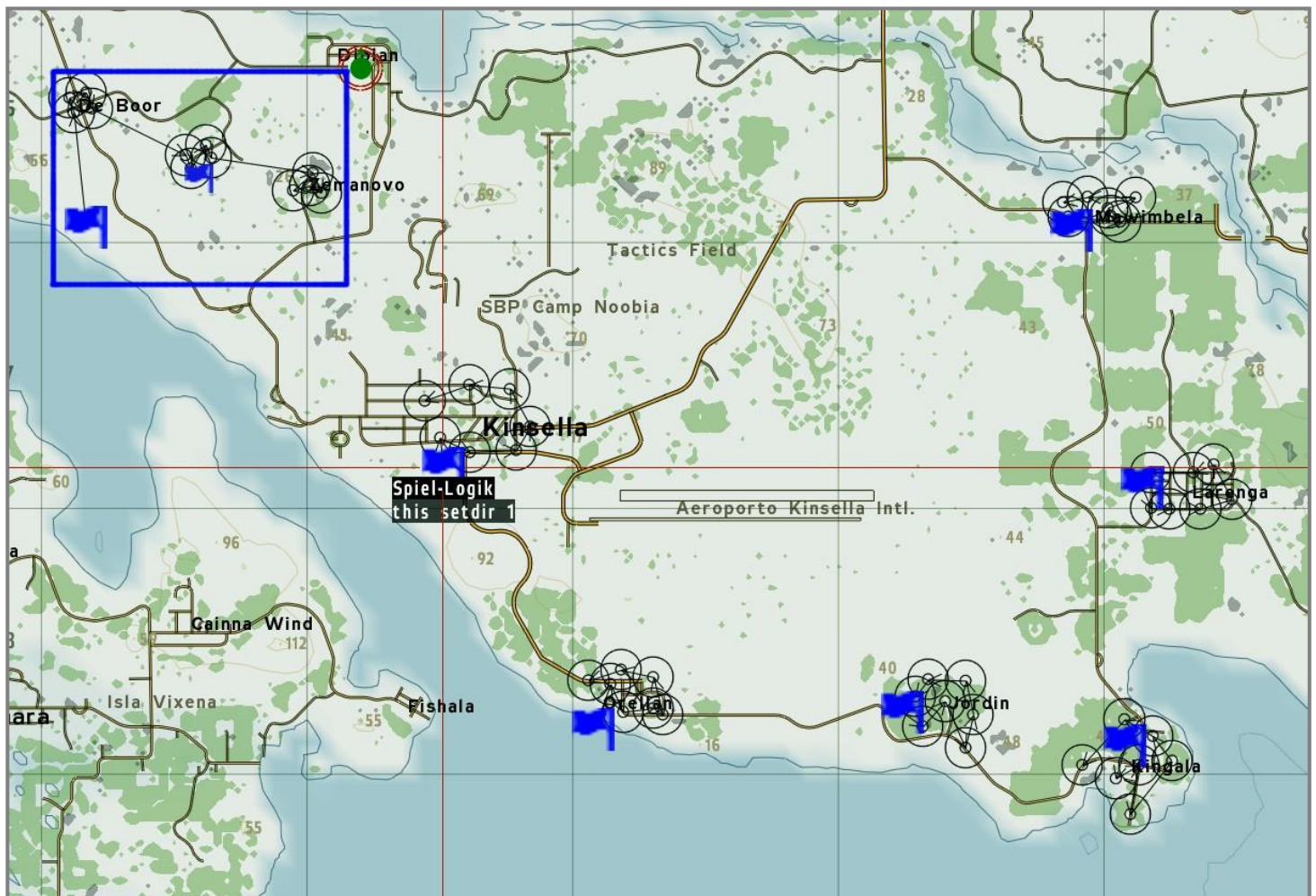
But when one of these logics come into the catchment area of a DAC-zone, they are taken into account in the waypoint- generation. It does not matter if that Zone is a pre-placed or a newly generated DAC-zone.

Hint:

Each logic, with its predefined waypoints, must have the correct unit-catagory entry:

For example > [this setdir 1](#) (Waypoints for Infantry)

This picture below shows a DAC-Zone with custom waypoints and several game logics with pre-defined waypoints at certain spots.



If you want to assign predefined waypoints to a certain DAC-zone, you'll have to give that logic a certain variable. This option ensures that a DAC-zone really recognizes only the custom waypoints which have been allocated through the logic.

Please note, that DAC-Zones can only input one logic per unit-category.

With other words: If you assign two logics with waypoints for the infantry, DAC will only be able to process one of them. The other one will be neglected.

How does “assign predefined waypoints” works, you can find out on the next page.

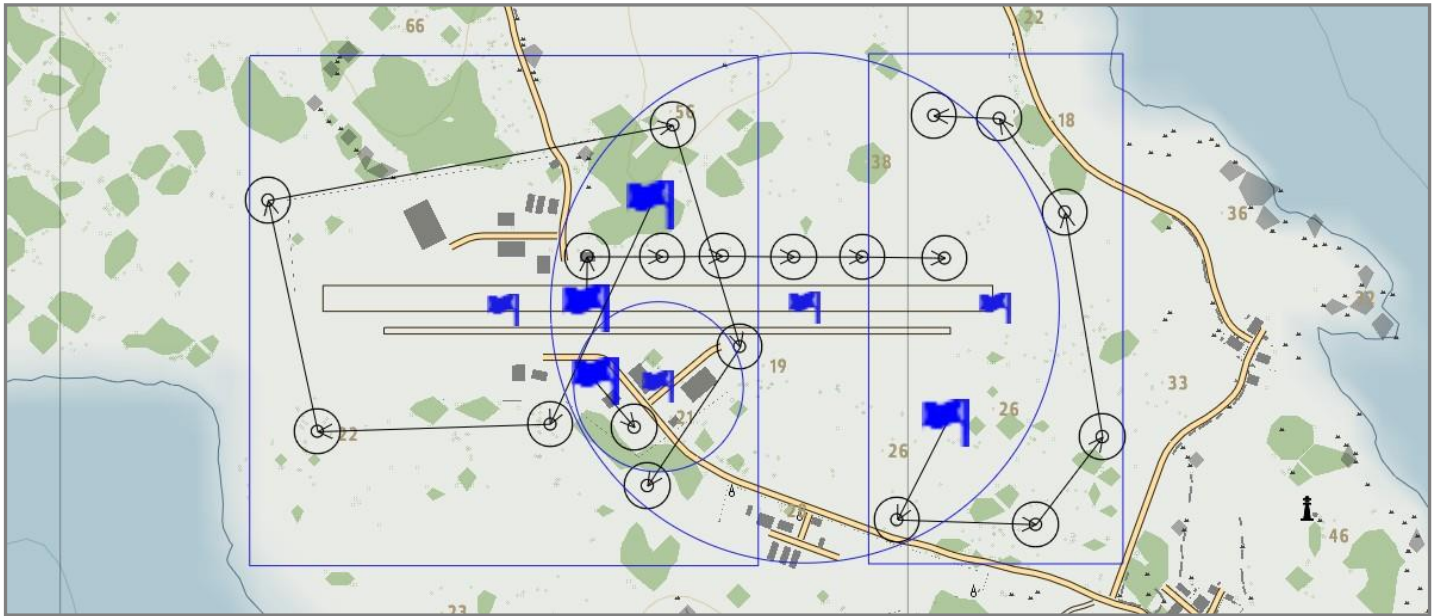
Assigning Logics to a DAC-Zone.

Logics with waypoints can be used differently, depending on the entry in the init section of the Logic:

this setdir (1-5)	The waypoints of the logic definig Custom Waypoints .
this setdir 90	The waypoints of the logic definig Custom object-positions .
this setdir 180	The waypoints of the logic definig a Polygon as a DAC-Zone .

These entries define how DAC should use the waypoints of a Logic.

There is also the possibility to assign a Logic to a DAC-Zone, which would come in handy when multiple zones overlap with each other. The following picture should give you a better understanding of this:



As you can see, the DAC zones overlap und use their own set of logics, which define a Polygon with their waypoints or mark several custom waypoints.

Such constellations of zones often lead to problems, since DAC sometimes can't figure out which logic to use. To avoid this problem, logics will be „linked“ to their respective zones. To do this, you'll have to make an entry in the init section of the Logic:

this setdir 1; this setVariable["Link", z1]	→	the logic has been linked with the DAC-Zone z1 .
--	---	---

As I said before, this measure is only necessary in critical zone constellations, and only if at least one logic with waypoints are used.

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],[15,2,10,15],[],[],[],[0,0,0,0]] spawn DAC_Zone

↑
→ The zone is **active** from the beginning

["z1",[1,1,0],[15,2,10,15],[],[],[],[0,0,0,0]] spawn DAC_Zone

↑
→ 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 scripts 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**
- **damage 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:

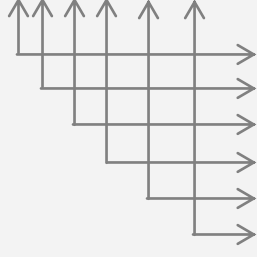
```
["z1",[1,0,0],[ ],[ ],[ ],[1,2,50,0,100,10],[0,0,0,3]] spawn DAC_Zone
```

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** = **infantry 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 than 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**

Connect 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 being 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],[ ],[ ],[ ],[ 1,2,50,0,100,10,[z2]],[0,0,0,3]] spawn DAC_Zone
```



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],[ ],[ ],[ ],[ 1,2,50,0,100,10,[z4]],[0,0,0,3]]exec spawn DAC_Zone
```



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],[ ],[ ],[ ],[ 1,2,50,0,100,10,[z4,z5]],[0,0,0,3]] spawn DAC_Zone
```

```
["z2",[2,0,0],[ ],[ ],[ ],[ 1,2,50,0,100,10,[z6,z7]],[0,0,0,3]] spawn DAC_Zone
```

```
["z3",[3,0,0],[ ],[ ],[ ],[ 2,2,50,0,100,10,[z5,z7]],[0,0,0,3]] spawn DAC_Zone
```

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, direction 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 unique 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&direction, 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 & direction | for a possible change of the size & direction[x,y,d] |
| 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],0,0,0] 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],0,0,0] 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-activation 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 !

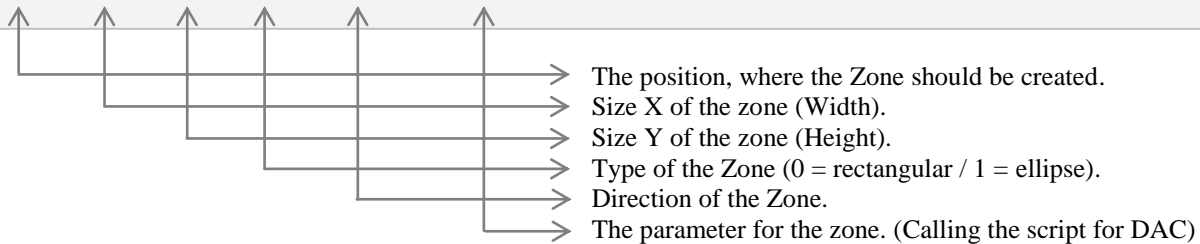
Creating DAC-Zones (mid-mission)

A new feature of DAC V3.0 is to create DAC zones „on the fly“, means during a running mission. This will provide many new possibilities and will help you in making missions much more dynamic.

If this function is called, DAC will create a new Trigger in the background and process the parameters. This newly created DAC-Zone will be fully integrated into the present DAC system.

This function can be called like this:

[position,sizeX,sizeY,type,direction,parameter] call DAC_fNewZone



Example:

[(position player),250,150,0,0,parameter] call DAC_fNewZone

A rectangular zone will be created at the position of the player. The Size of the zone is 250 x 150 m and the direction is 0 (North)

The last parameter complies to the script, which makes the created trigger to a new DAC-Zone.

It is basically the script call that is usually entered in the init line of the trigger.

It would make sense to store these parameters in a local variable and implementing it afterwards in the function:

Example:

```
_values = ["z1",[1,0,0],[5,2,50,8],[[],[],[1,1,1,1]];  
[(position player),250,150,0,0,_values] call DAC_fNewZone
```

I won't describe the parameters of the variable **_values**, since i did this before. But, as you can see, a zone with the name "z1" will be created, which will generate 5 BLUFOR Infantry-groups.

There is a little limitation: You will not be able to create multiple zones at the same time, which means that you must wait until the zone and their groups are generated before creating another zone. To check if a DAC-Zone has been completed, you can call the variable **DAC_NewZone**:

Example:

```
waituntil{DAC_NewZone == 0};  
_values = ["z1",[1,0,0],[5,2,50,8],[[],[],[1,1,1,1]];  
[(position player),250,150,0,0,_values] call DAC_fNewZone
```

Deleting a DAC-Zone (mid-mission)

It is everytime possible to delete a DAC-Zone. And you can not only delete zones that have been created during the Mission, but also zones that were already placed in the editor.

The deletion of DAC-Zones works in the same way as deactivating them:

All units, which are generated by their respective zones, will be deleted.

Furthermore, all linked variables, arrays and waypoints will be deleted. It is not possible to activate the deleted zone afterwards, since DAC does not recognize the zone anymore.

The function call to delete a zone is as follows:

```
["zone"] call DAC_fDeleteZone
```

If a group from another Zone has waypoints which belong to the deleted DAC-Zone, DAC will automatically reduce these waypoints of the group.

Note that you can't delete multiple zones by using the function several times. In order to check if the zone has been deleted you can use the same variable I mentioned before:

Example:

```
waituntil{DAC_NewZone == 0};  
["zone"] call DAC_fDeleteZone
```

However, you can delete multiple zones by just passing the zones to the function like this:

Example:

```
waituntil{DAC_NewZone == 0};  
["zone1","zone2","zone3","zone4"] call DAC_fDeleteZone
```

As you can see, the zones which have to be deleted must be specified as "strings".

The DAC artillery (Requires a logic with the name "DAC_Support_Logic")

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 separate 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. That's 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-Group** [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
 - 12** = **Artillery-Unit** outside the maximum range

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

DAC-Ground Support (Requires a logic with the name "DAC_Support_Logic")

The DAC-Ground Support system is handled automatically by the AI. If AI groups get under fire, they will try to call in support.

This feature can now be used also by a Player in the new DAC-Version, which means, that players can call in AI-support if needed.

Ground Support can be called at all times. This does not mean however, that support groups will be on their way automatically. There is no difference between a player calling in support or an AI.

The following function will request DAC-Ground Support (max. 1 group per request):

```
[player,position] spawn DAC_fCallHelp
```

The first parameter must be a valid player unit.

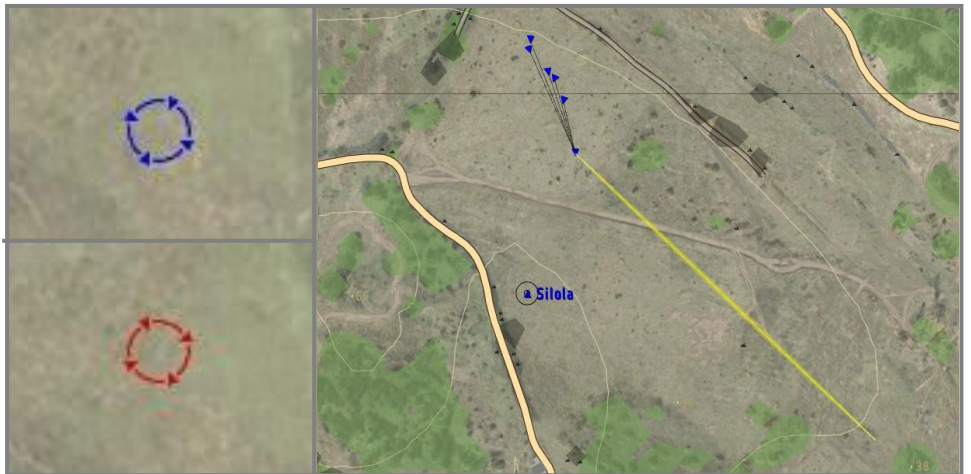
DAC will ignore the request if the first para refers to an AI unit.

The second parameter must be a valid position within a certain radius of the player unit.

The **DAC_Config_Creator** section of this readme file will tell you how to define the radius or other parameters. The standard distance is defined as 1000m.

If the DAC-Markers are activated, you should be able to see a rotating blue marker if the request was successful.

If the position is out of reach, you should see a rotating red marker.



A group, which is on a support-mission, is indicated by an **animated yellow** marker.

DAC will usually try to „contact“ the nearest available group, whatever unit category the group comes from. As I said before, a successful support-request does not mean that a group will automatically come in for support. There could be many causes, which would not allow the group to come in for support: They could be under fire or are already supporting another group et cetera.

You can use this function, in order to see which groups are defined as „supporting groups“:
DAC_Support_Logic getVariable "**support_groups**" = (**only by the player requested groups**)

Note:

Only groups, which are generated by the DAC-System are affected by this.
Other groups are not supported by DAC.

Integrate Editor-groups

It's basically possible to integrate groups, which have been placed in the Editor, right into the DAC System. You can either get this started right when the mission starts or even later as well.

An integrated group can get "released" by the way out of the DAC System at any time.

If the group has waypoints assigned, so DAC will save them if the group will get released out of the DAC System later again, to reassign these waypoints to the group.

Following categories will be supported with this

Infantry, **wheeled vehicles**, **tracked vehicles**

As long a group was integrated in the DAC-system, following DAC specials will be assigned:

- The group is using DAC-waypoints resp. the corresponding movement routines.
- The DAC group will be reduced.
- Killed units of the group will be deleted (corresponding to the settings of the deleting routines).
- The group is enabled to call in reinforcements and Artillery support.
- The group can be called for reinforcement.
- The group can make use of the DAC_events.
- The group will use a DAC_behaviour configuration.

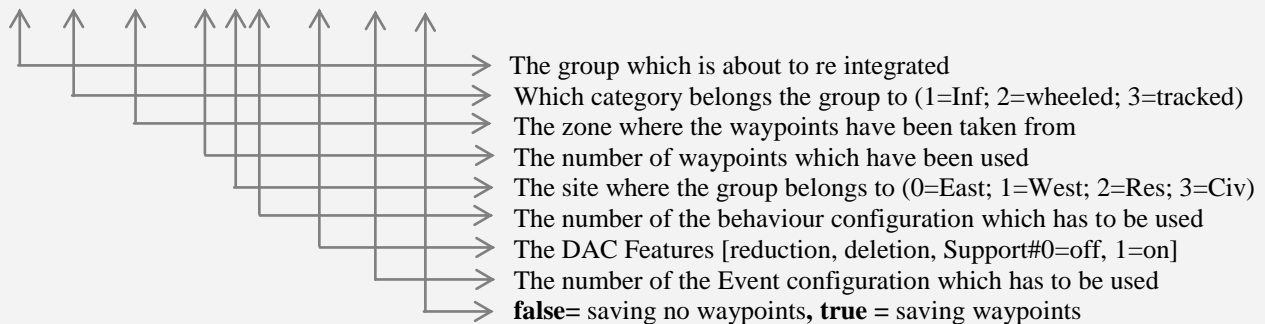
But there's none the less a limitation:

- An integrated group can not get respawned again after it has been killed

The integration of an editor-group will be done by using following script-call:

Example:

`[group,1,[z1,z2],10,1,1,[1,1,1],0,false] spawn DAC_fInsertGroup`



Use following Function just to release a group again out of the DAC system:

`[group] spawn DAC_fReleaseGroup`

If waypoints were saved for this group so this function will care about the waypoints to be restored again for this group.

Release DAC groups

It's recently even possible to become DAC groups either released or completely removed out of the DAC system. This function might cause the question, "why to do that?"

Well, the reason is pretty simple: There are quite good other AI Applications which I would like to get supported by this ☺

Such a constellation could be look like this:

1. DAC is generating the units automatically within the placed DAC zones.
2. All generated groups will be removed out of the DAC System after initialization.
3. Another AI enhancement takes over control of the group.

Not all DAC zones need to be removed of course. So it's as well possible to remove DAC groups only which belongs to certain zones within the DAC system.

Example:

You're generating 4 Zones at all with the DAC. Two of these Zones keep being untouched, so that DAC is taken control about these groups. The groups of the other two zones will be removed out of the DAC system and the groups control will be taken over by another AI system.

It's of course possible to remove groups out of the DAC System individually. The function script call is the same as mentioned above already.

```
[group] spawn DAC_fReleaseGroup
```

Because the released DAC groups don't have any waypoints anymore, so they'll just keep standing once the Function call has been executed. There're a few optional parameters around to keep the group moving.

```
[group, [z1],1,8] spawn DAC_fReleaseGroup
```

- | | | |
|------|---|---|
| [z1] | = | one (or even more) valid DAC zone(s), where waypoints will be taken from. |
| 1 | = | The units Category (1= Infantry; 2= wheeled; 3= tracked) |
| 8 | = | The number of waypoints, which will randomly be assigned to the groups out of the DAC Zone(s) |

A simple DAC independent Move script is making the groups running along their waypoints.

Because I appreciate the „**Group Link4**“ pretty much, so I cared about to support this AI System by default. Means that on a need released DAC groups will automatically be recognized and integrated by GL4.

SNKMAN and me, we have appointed a special group-variable, which will clearly recognize whether a group has been generated by DAC and if yes whether this group is under control by DAC or whether she's out of command by DAC.

Explanation:

Each group which has been generated by DAC gets following variable assigned:

```
group setVariable ["DAC_Excluded", False]
```

The Variable will be replaced for each group which is leaving the DAC-System:

```
group setVariable ["DAC_Excluded", True]
```

DAC-object-generation

Since version 3.0 it's possible to create objects within DAC-zones. Objects are created strictly random. But there are some options to manipulate this randomness.

The DAC-zones for object-generation can have the same shapes as the DAC-zones for generating units: **circle, ellipsis, rectangle and polygon**.

Hint:

Generated objects must have their own zone. It's not possible to create units and objects in the same zone.

The generation of objects in a zone is triggered by a unique script-load. The procedure is similar with generating waypoints, except the fact, that additional objects are created.

DAC knows 2 modes of generating objects which are triggered via the script-load.

To successfully create an object within DAC, it has to know a valid position within the zone where it places the object. The two different variants are:

The standard variant	:	Positions are checked against the same criterias as waypoints.
The turbo variant	:	No checks. Any random position is used.

It's up to you which mode you choose to match your requirements.

To save time, you can force DAC to split the generation of objects into two processes which are processed parallel. If you have many objects it can save you a lot of time if the conditions are good.

The following pictures show the two variants of processing (left = **single** / right = **dual**)



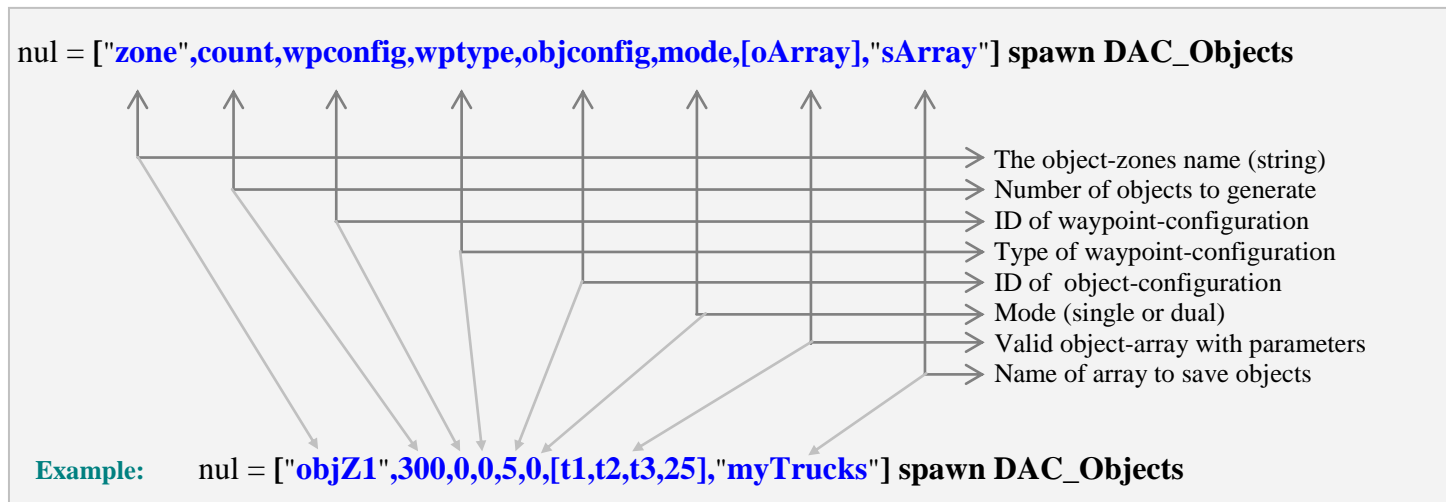
If the DAC markers are enabled, DAC will draw for each object its own marker. The size of each marker will be drawn exactly in the size of the object. (for very small objects, it may be that you recognize no more markers).

DAC-object-markers are black rectangles by default, but it's possible to configure form and color.

A counter (or two if dual-processing is enabled) shows the process of object-generation. If DAC-system-messages are enabled, additional output is displayed in the sidechat:

```
1-1-A 1 (Silola): "ZONE '05' : ALL 250 OBJECTS WERE GENERATED."  
1-1-A 1 (Silola): "ZONE '05' : ALL 250 OBJECTS WERE GENERATED."  
1-1-A 1 (Silola): "ZONE '04' : ALL 500 OBJECTS WERE GENERATED."
```


The following diagram shows the script-call to create objects within a DAC-zone:



“zone” The name of the object zone. It’s mandantory to give the zone (trigger) a name wich has to be used in the script-call as a string (same principle as the unit-zones).

count This parameter configures the number of objects to create. Please keep in mind that too many objects do have a negative influence on the performance.

wpconfig This parameter has allways to be used in connection with **wptype**.
If the value is greater than **0** a valid configuration from **DAC_Config_Waypoints** is loaded and requires a value between **0** and **4** in the parameter **wptype**.
The generation of objects will match the same criterias as the generation of waypoints.
This is the **standard-variant** wich is mentioned above

If both parameters are **0** you have enabled the **turbo-variant** wich does not check any positions and is thereby faster.

wptype This parameter is used in conjunction with **wpconfig**. Possible values are **0** to **4** wich indicates the unit categorie.

Examples:	wpconfig	wptype		
	1	4	standard	- positions of type camp
	3	0	standard	- positions of type infantry
	3	5	invalid	- wptype 5 is an invalid option
	2	2	standard	- positions of type heavy vehicles (tanks)
	0	0	turbo	- no checks
	0	1	invalid	- wptype 1 requires a wpconfig value >0

Standard

With standard-variant objects will be placed by the criterias of waypoint-generation. Vegetation and terrain will be considered.



Turbo

With turbo-variant objects will be placed without any checks of positions. Terrain and objects are **not** considered.



- objconfig** This parameter loads a valid configuration from **DAC_Config_Objects**. It has a ruleset with objects which will be placed in the zone. You will learn how to build such an object-configuration and which additional parameters you have in the chapter **DAC_Config_Objects**
- mode** The type of generation-mode: **0** = **single-mode** / **1** = **dual-mode**
- [oArray]** You can use this array to determine which objects or positions should have a special focus on clearance distance. In other words, you can set a safe-distance around the objects in this array.
- There are several ways to fill the array with data, but only one way to set the safe-distance: It's **the last value (numeric)** in that array.
- Possible variants:
- Objects** - **[obj1,obj2,obj3,obj4,25]**
Objects, separated with comma. The last value is the safe-distance around the objects. In the given example DAC will not create any other objects within **25** m of objects, given in the array.
 - Array** - **[Array,25]**
Instead of using a lot of objects you can also use an (existing) array. The consideration of these objects is done according to the same principle as in the variant **objects**.
 - Logic** - **[Logic,25]**
Instead of using real objects, you can use logic-objects. Just place it on the map, name it and place as many waypoints as you like. Those waypoints will have a safe-distance around them, like real objects in the example above. Fill in the name of the logic instead of object- or array names.
 - Zone** - **[Zone,25]**
The last variant takes account of objects, generated by another object-zone. Here are a small example:
You have a zone which generates buildings. Now you want to create a zone which creates trees in the same area. You can now give the buildings from the first zone a safe-zone around them, where no trees will be placed.
- "sArray"** The last parameter creates a new array which contains the newly generated objects. This parameter can have any name but has to be a string. This array is then available to you for further processing.

Hint:

If you just need random positions in a zone without DAC creating objects, use a logic unit as an object. DAC will then save the positions in the array **"sArray"**. You may want to have that to generate random mission-markers or anything else. ☺

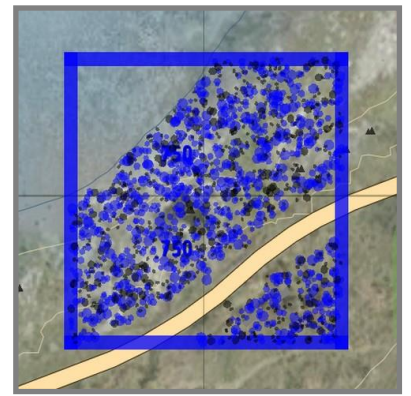
Some examples on object-generation:

`["oz1",500,0,0,3,0,[],""]` spawn DAC_Objects

This zone was filled in turbo-mode. Not checking positions does not mean that you'll have objects everywhere. DAC takes care of critical positions like water and roads, like shown in the picture.

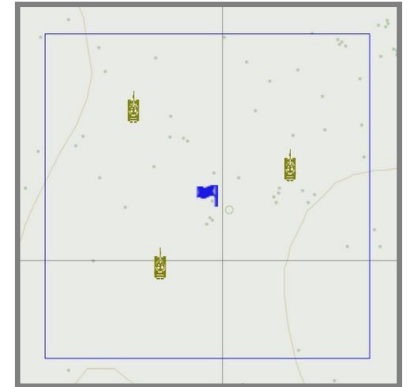
You can change that behaviour everytime by editing the file:

[DAC_Config_Objects](#).

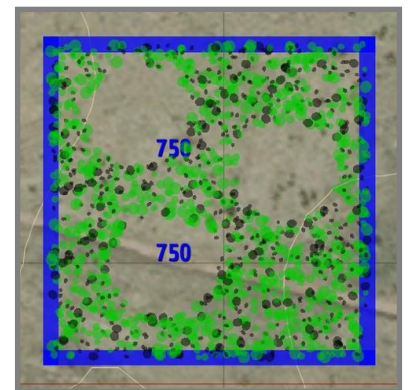


`["oz1",500,0,0,3,0,[t1,t2,t3,25],""]` spawn DAC_Objects

This zone contains three tanks, which have a safe-zone of 25 m. The tanks are named **t1**, **t2**, **t3** as you can see in the script-call.



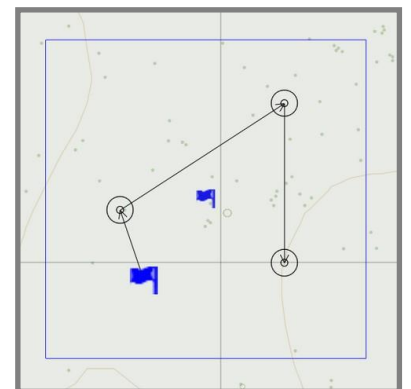
As you can see in the picture, DAC has not placed any objects within 25 m of those tanks.



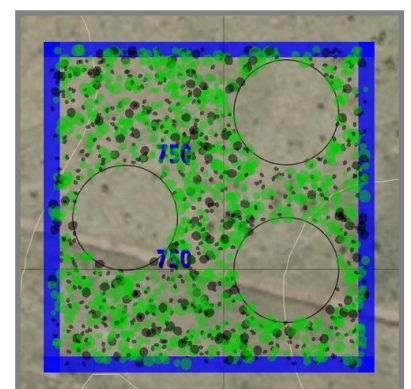
`["oz1",500,0,0,3,0,[Log1,25],""]` spawn DAC_Objects

Same stuff as the tanks, but without real objects.

Here I used a **Logic** with three waypoints which will have a safe-zone around them.



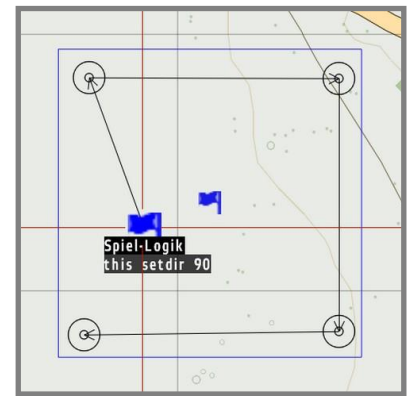
Result is the same as above but without tanks.



`["oz1",15,4,4,6,0,[],""]` spawn DAC_Objects

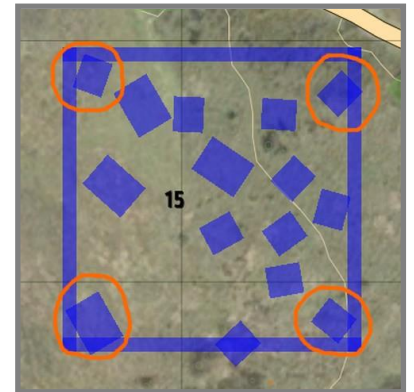
You can use logic-waypoints the other way around.

If you place waypoints for a logic, objects will be created at those waypoints.



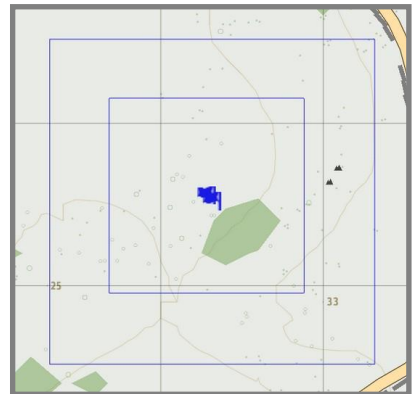
Place **this setdir 90** in the init of the logic.
This entry tells DAC what to do with the logic.

In this example 15 buildings were created. 4 on the given waypoints and the rest at random positions.

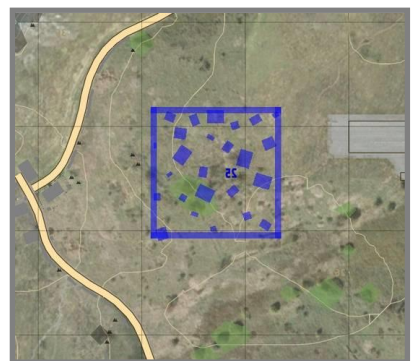


The last example shows you the variant **zone**.
So it is possible to leave on an object-zone until another zone has generated all objects. These objects can then be considered.

In this example we have two zones. The smaller one should have 25 buildings generated in it, the bigger one will contain 400 trees and bushes.
The picture to the right shows the situation in the beginning.

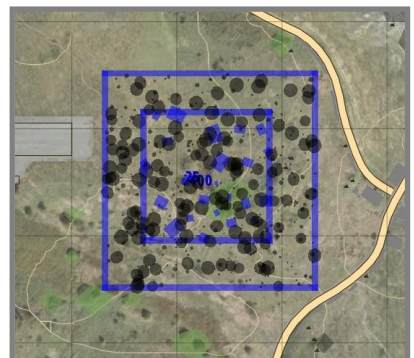


The two pictures on the right show you the problem we have to solve.
If the object-generation is not depending on each other, buildings will be created with a nice space between each other.



The trees and bushes will be placed everywhere in the bigger zone even in the buildings. The resulting chaos is nothing we want to have.

To resolve this chaos, you can tell the zone with the trees and bushes to wait until all buildings are created, give them a safe-zone and then create the vegetation.



If you look at this picture you'll see that there are no trees and bushes in or near buildings. The safe-zone is 8 m.

To be able to do so, the bigger zone needs the positions of those buildings.

If you remember what you read earlier in this document, you have the ability to store all objects from an object-zone into a custom array.

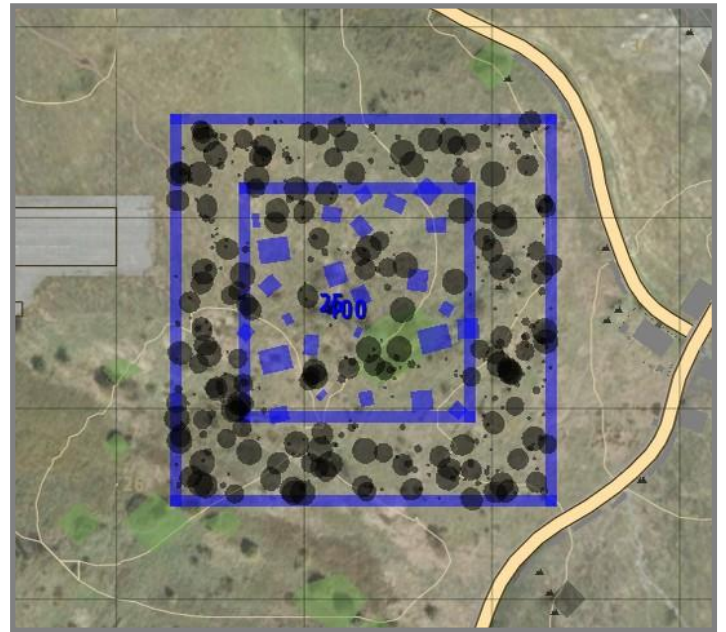
You can use this array then to give all objects in that array a safe-zone.

Suppose that the zones look like this:

U1 is the zone with buildings in it

U2 is the zone with vegetation

A3 is the array where all objects from zone **U1** are stored.



```
["U1",25,4,4,6,0,[],"A3"] spawn DAC_Objects
```

U1 generates 25 buildings and stores the objects in the array **A3**

```
["U2",400,0,0,3,0,[U1,8],"" ] spawn DAC_Objects
```

U2 will generate 400 trees and bushes. But it'll wait until **U1** has generated it's buildings. The vegetation will be placed outside the 8 m safe-zone around the buildings from **U1**.

It could happen that DAC is not able to create all desired objects in the given zone. If you have enabled the DAC-system messages, you'll get a message about it in sidechat.

```
1-1-A 1 (Silola): "ZONE '04' : ONLY 74 OF 150 OBJECTS COULD BE GENERATED."
```

The configuration-files

The configuration files can be seen as a kind of data pool for DAC. The file **DAC_config_creator** takes a special part in this, because all the global variables and the basic parameters are defined in there. Those ones are adjustable up to the needs of the user.

A further important file is the **DAC_config_sound** where all the sound files are defined which are used by DAC in certain Situations. This file is not fully adjustable.

All the other config files are free adjustable up to the users needs. The structure and the calling of these configuration-files are always the same.

These configuration files are containing so called "data-blocks", which are separated by clear ID's. If DAC has to use a special data-block, so the respective ID is defined only. That makes this parameter block known to the DAC System. How and where this directly works, you'll know in the description of each configuration-file.

To initialize DAC respective to start DAC, one of the both given DAC-logics need to be placed on the map. First there is the "**DAC-intern**" and the other one is the "**DAC-extern**" logic.

The DAC-Logic "**DAC-intern**" is working with the already provided intern configuration files, which are located inside the pbo. These files are containing e.g. all default unit settings and even the default behaviour settings and so on. So everything you need to initialize a test environment for your DAC Mission.

Additionally to this the "**DAC-intern**" file offers you the possibility to outsource single configuration-files right inside your Mission directory. That enables you to make changing's on the files or expand the already existing ones with your own files. To outsource files individually, just define following Syntax right inside the init-line of the DAC-Logic:

```
DAC_Single_Config = ["Units"]
```

In this case this means that DAC is expecting the file „DAC_Config_Units.sqf“ in your missions directory. If you want to outsource further files, so you just need to declare further files:

```
DAC_Single_Config = ["Units","Behaviour","Arti"]
```

This example represents the case that DAC is expecting now the files DAC_Config_Units.sqf, DAC_Config_Behaviour.sqf and DAC_Config_Arti.sqf in your missions directory.

This looks a bit different for the Logic "**DAC-extern**". DAC is now expecting a DAC folder right in your missions directory, where all configurations-files are stored in. The DAC folder can be found in the zip file by the way. These configuration-files are adjustable and can be expanded pretty much up to your needs.

There are following configuration-files existing:

DAC_Config_Creator	The basic settings of DAC
DAC_Config_Units	Defines the unit types for all 4 parties
DAC_Config_Behaviour	Defines the behaviour of created DAC groups
DAC_Config_Waypoints	Defines the settings for the waypoint generator
DAC_Config_Camp	Defines the look and the settings of DAC camps
DAC_Config_Arti	Defines the conditions and settings for the DAC Arti
DAC_Config_Events	Enables the access on certain parts of the DAC
DAC_Config_Objects	Defines the objects for the object generator
DAC_Config_Weapons	Enables the definition of own weapon compositions
DAC_Config_Sound	Defines language and radio sounds within the DAC
DAC_Config_Marker	Defines the look and the behaviour of the DAC marker

[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],[5,2,50,6],[],[],[],[0,0,0,0]] spawn DAC_Zone

 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] <<<< **Use this only in SinglePlayer mode** >>>

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],0,360,100,0]

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.
6. Behaviour of the Respawn-camps when no more respawn available (to hold arti-support for example).
0 = The camp group will give up the camp, **1** = The camp group will continue to guard the camp.

DAC_Delete_Value = [[120,150],[140,150],300]

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

1. Time (in sec.) + minimum distance to all player units before dead units get deleted
2. Time (in sec.) + minimum distance to all player units before destroyed vehicles get deleted
3. Time in seconds, until an empty vehicle without a membership is destroyed (so that it can be deleted later)

DAC_Del_PlayerBody = [10,150]

If needed you can activate the removal of dead player units (in MP) here.
Occasionally 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 units before a dead player unit gets deleted

DAC_Com_Values = [1,2,0,0]

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 as well 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
4. DAC Monitor: 0 = Monitor de-activated, 1 = Monitor activated for all, "Name" = Only "Name" has Monitor

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 scripts.

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 representing 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 **Independent** are fighting for.
You must (!) apply the same settings that are given in the editor.

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

Attention! With this setting independent 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 considered usable by the AI.

DAC_Marker = 1

This parameter 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_Join_Action = false

This option is currently just a test option and not fully supported. It's also available for SP Mode only.
If this option has been activated (true), so you'll receive an Action menu entry once you close up a friendly DAC-unit.
This menu entry would enable you to become that unit part of your group.

DAC_Fast_Init = false

This option is available in Singleplayer Mode only.
If this option has been activated (true), so while the DAC is about to get initiated the view distance will be reduced to the most minimum value and the fog will be set on a maximum value.

This would cause a further CPU performance and would further help DAC to calculate the massive amount of operations without any massive lags. This would decrease the time of initialization on up to 40%.

If the initialization process has been finished, the view distance and the fog value would be reset to the actual values.

DAC_Player_Marker = false

This options creates a marker for each player individually. This is for single player as well as for multiplayer Missions.

DAC_Direct_Start = false

DAC is expecting one DAC zone per default to become initiated. But if you will activate this option (true), so you can make the DAC initializing by the DAC Logic only.

This option is needed only if you want to become DAC zones created while a running mission, or if you want to generate objects only.

DAC_Activate_Sound = false

This parameter is activating / deactivating DAC sounds, which are played in certain Situations.

DAC_Auto_UnitCount = [8,10]

You can use this option to adjust the amount of DAC groups on the amount of player units dynamically. With other words: So more human players are present on the map (coop) so more DAC groups will be generated and conversely.

1. The amount of players which is set for the mission (default setting for your mission).
2. The rate the DAC groups will increase or decrease in percent (for each player which is more or less participating the mission).

That doesn't happen automatically, it's furthermore possible to directly define the Zones resp. the unit types, which are supposed to be part at automatism.

May be you have a DAC-Zone with following script call:

```
["z1",[1,0,0],[18,2,75,10],[ ],[ ],[ ],[1,1,1,1]] spawn DAC_Zone
```

That means, that exactly 18 infantry groups will be created right at missions start, whereas it makes no difference how many players are participating the mission.

To dynamically reduce or increase the amount of units of these 18 DAC-groups, related to the number of players, just set this number in quotes:

```
["z1",[1,0,0],[18",2,75,10],[ ],[ ],[ ],[1,1,1,1]] spawn DAC_Zone
```

You can use these settings individually in each zone for respective unit-categories.

Below you find some calculation examples about how 18 DAC groups would behave, outgoing from the settings above [8,10]

If 8 players are participating the mission, DAC will create 18 groups (defined as default)

If 6 players are participating the mission, DAC will create 14 groups ($18 - (2 \times 10\%)$)

If 4 players are participating the mission, DAC will create 10 groups ($18 - (4 \times 10\%)$)

If 9 players are participating the mission, DAC will create 20 groups ($18 + (1 \times 10\%)$)

If 11 players are participating the mission, DAC will create 23 groups ($18 + (3 \times 10\%)$)

DAC_Player_Support = [10,[10,2000,3,1000]]

This option activates and configured the ground- and Artillery support for the Player.

To use this, a DAC Logic with the name "„**DAC_Support_Logic**“ must be present.

1. Maximum amount of Artillery-requests.
2. Maximum mount of ground support requests.
3. Maximum range where a player is enable to request ground support (outgoing of the position of the player)
4. Maximum amount of supporting groups which are providing ground support simultaneously
5. Maximum range where the System is searching for support groups.

Further details about respective kind of support can be found on the pages:

23 = The DAC-Artillery

24 = The DAC-ground support

DAC_SaveDistance = [500,["DAC_Save_Pos"]]

Here you can define positions on the map, where no DAC-Waypoints and no units are supposed to be generated in these areas (during the start-phase).

May be you want to start a Mission right within a huge DAC-Zone, so you can use following settings to make sure that no DAC units will be created near your position.

1. The distance to given Objects, where no DAC waypoints have to be generated
2. The list of Objects (as string), which shall be considered while the distance measuring (e.g. a Logic)

It's possible by the way to define several Objects: **DAC_SaveDistance** = [500,["savePos1","savePos2","savePos3"]]

DAC_GunNotAllowed = ["B_Mortar_01_F","O_Mortar_01_F"]

DAC_VehNotAllowed = ["O_MBT_02_arty_F","I_APC_Wheeled_03_cannon_F"]

This line declares the types of (empty) static weapons and vehicles that may not be acquired by infantry units.

DAC_Locked_Veh = [player_car_1,player_car_2]

Here you can set editor placed vehicles, which may not be used by the AI.

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

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.

DAC_SP_Soldiers = ["B_soldier_AR_F","B_G_soldier_AR_F","O_soldier_AR_F"];

The unit types which will provide suppressed fire support, will be defined here.

Most of the settings of the file **DAC_Config_Creator** can access 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.

[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 = [
        "RU_Soldier_Crew","RU_Soldier_Pilot","RU_Soldier_SL","RU_Soldier","RU_Soldier_GL",
        "RU_Soldier_MG","RU_Soldier2","RU_Soldier_Medic","RU_Soldier_AT",
        "RU_Soldier_Sniper","RU_Soldier_AR","RU_Soldier_Marksman",
        "RUS_Soldier_Sab","RUS_Soldier_Marksman"
    ];
    _Unit_Pool_V = [ "UAZMG","UralOpen_INS","UAZ_RU","UAZ_AGS30_RU","BRDM2_INS","UAZ_MG_INS"];
    _Unit_Pool_T = [ "T72_RU","ZSU_INS","BMP3","2S6M_Tunguska","T90","BMP3","BTR90"];
    _Unit_Pool_A = [ "Mi17_rockets_RU","Ka52","Mi24_V","Mi24_P"];
};
```

Figure based on Arma2

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

```
["z1",[1,0,0],[5,3,30,8],[ ],[ ],[ ],[0,0,3,4]] spawn DAC_Zone
```

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 separately. "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      = [0.2,0.7];
    _setCombat     = ["green","white","yellow"];
    _setBehav      = ["careless","safe","aware"];
    _setSpeed      = ["limited","normal","full"];
    _setForm       = ["line","vee","column","wedge","stag column","ech left","ech right","file","diamond"];
    _setFleeing    = [0,100];
    _setHeliVal    = [45,100,0.7,1];
    _setPause      = [[5,10],[5,10],[5,10],[20,30,5,5],[1,3],[0,0]];
    _setBldgBeh    = [0,50,120,600,1];
    _setPatrol     = ["45 + (20 * (skill _leader))","(60 + (random 60)) + ((skill _leader) * 50)"];
    _setSearch     = ["40 + ((skill _leader) * 150)","50 + ((skill _leader) * 50)"];
    _setSupport    = [1,2];
    _setJoin       = 2;
    _setEmpVeh     = [[0,100],[0,100]];
    _setSupTime    = ["5 + ((skill _unit) * (5 * DAC_AI_Level))",2,5];
    _setHidTime    = ["((10 * DAC_AI_Level) + ((skill _leader) * 50)) / ((count units _group) + 1)"];
};
```

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

```
["z1",[1,0,0],[5,3,30,8],[ ],[ ],[ ],[0,0,1,0]] spawn DAC_Zone
```

The settings are as following:

_setSkill	<p>[0.3,0.9] or alternatively [0.2,0.8],[0.2,0.8],[0.2,0.8],[0.2,0.8],[0.2,0.8],[0.2,0.8],[0.2,0.8],[0.2,0.8],[0.2,0.8],[0.2,0.8]</p> <p>This array determines the range of the skill value with which the units in the zone will be generated.</p> <p>1st Entry = the minimal skill of the units in this zone. 2nd Entry = the maximum skill of the units in this zone. In the alternative version, the Min / Max values for each sub-skill can be entered.</p>
_setCombat	<p>["white","yellow"]</p> <p>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.</p>
_setBehav	<p>["aware","combat"]</p> <p>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.</p>
_setSpeed	<p>["normal","full"]</p> <p>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.</p>

_setForm	<p>["line","vee","column"]</p> <p>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.</p>
_setFleeing	<p>[0,50]</p> <p>These parameters determine the fleeing behavior and the search for cover of the units.</p> <p>1st entry = fleeing behavior. The value must be between 0 and 1.</p> <p>2nd entry = this value determined at what distance to the leader, a unit will leave his place in a stationary gun, to rejoin the group again. If for example the guns in principle should be remain occupied, then a great value should be entered there.</p>
_setHeliVal	<p>[45,100,0.7,0]</p> <p>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 pos, mostly in the vicinity of its starting position..</p> <p>1st entry = minimum altitude for helicopters in the zone</p> <p>2nd entry = maximum altitude for helicopters in the zone</p> <p>3rd entry = maximum weather value when helicopters will start, otherwise they land premat.</p> <p>4th entry = generate "HeliHEmpty" OFF (0) / ON (1)</p>
_setPause	<p>[[3,5],[3,5],[5,10],[20,30,5,5],[3,5],[1,30]]</p> <p>In this array you can define a pause time for the various units they will wait at each of their waypoints.</p> <p>1st array = waypoint pause for infantry [minumum + random]</p> <p>2nd array = waypoint pause for wheeled vehicles [minumum + random]</p> <p>3rd array = waypoint pause for tracked vehicles [minumum + random]</p> <p>4th 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)</p> <p>5th array = waypoint pause for the camp group [minumum + random]</p> <p>6th array = delay for all units in the zone [minumum + random]</p> <p>If you want the units from a zone not to move out directly to their waypoints you can set up a delay time.</p> <p>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.</p>
_setBldgBeh	<p>[2,50,120,600,1]</p> <p>Here you determine how much infantry groups can occupy buildings.</p> <p>If you don't want to use this feature, you need to set the 1st value to 0.</p> <p>1st entry = maximum number of units from the group, which can occupy a building</p> <p>2nd entry = radius of the building detection</p> <p>3rd entry = time how long the units stay in a building</p> <p>4th entry = time before a building can be entered again a second time</p> <p>5th entry = number of positions, which a building must have at least</p> <p>The example above would also mean that 2 units of a group occupy a building, if a suitable building with at least one position in a radius of 50 meters was found. The units hold it for 120 seconds. Then the building is not occupied for 600 seconds..</p> <p>Caution, there are not many buildings / objects which work correctly.</p> <p>Buildings which cause problems can be excluded (see issue Tips + Tricks)</p>

<p>_setPatrol</p>	<p>["50","60"]</p> <p>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.</p> <p>1st entry = max radius for the patrol depending from the vehicle of the leader</p> <p>2nd entry = time a patrol will last, until the group will board their vehicle again to continue their waypoints.</p> <p>The example above means: the group will patrol for 60 seconds in a radius of 50m and then board their vehicle again.</p> <p>2 more examples: ["50 + random 50","60 + random 30"] ["30 * DAC_AI_Level","60 * DAC_AI_Level"]</p> <p>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)"]</p>
<p>_setSearch</p>	<p>["100","120"]</p> <p>Units from the categories infantry + wheeled vehicles will search the area for enemy units when they lost contact with them.</p> <p>The handling of the parameters is exactly the same as for _setPatrol.</p> <p>1st entry = max. search radius for the group</p> <p>2nd entry = time a search will last, before the group will continue their normal waypoints</p>
<p>_setSupport</p>	<p>[1,2]</p> <p>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.</p> <p>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 ;-)</p> <p>1st entry = zone groups are free to support (0 = no, 1 = yes)</p> <p>2nd entry = activates the artillery request in the zone (0 = disabled)</p> <p>To activate artillery requests by infantry groups in the zone the 2nd value has to be a valid number from the DAC_Config_Arti.</p>
<p>_setJoin</p>	<p>2</p> <p>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.</p>
<p>_setEmpVeh</p>	<p>[[150,100],[100,50]]</p> <p>These arrays enable the boarding of empty vehicles and guns.</p> <p>If you don't want to use one of these characteristics in the zone, you have to set the 1st value to 0, otherwise as follows:</p> <p>1st Array = radius of the detection of empty vehicles to board; probability %</p> <p>2nd Array = radius of the detection of empty guns in order to board; probability %</p> <p>Which vehicle types can be boarded, will be determined by the setting up DAC_VehAllowed (page 22).</p> <p>Which guns can be boarded, will be determined by setting up DAC_GunAllowed (page 22).</p>
<p>_setSupTime</p>	<p>["5 + ((skill _unit) * (5 * DAC_AI_Level))",2,5]</p> <ol style="list-style-type: none"> 1. The time window in which valid units provide suppressed fire 2. How many times in a row, these units provide suppressed fire 3. How many seconds pause is placed between the suppressed fire phases.
<p>_setHidTime</p>	<p>["(((10 * DAC_AI_Level) + ((skill _leader) * 50)) / ((count units _group) + 1))"]</p> <p>The time, a group go for cover and do not advance further, when they has enemy contact.</p>

[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 = [ 10,    10,    10,    20,    20    ];  
    _checkRadius2 = [ 15,    20,    20,    40,    40    ];  
    _checkAreaH   = [ 40,    15,    20,    10,    10    ];  
    _checkMaxH    = [ 500,   500,   500,   500,   500   ];  
    _checkMinH    = [ 5,     5,     5,     5,     5     ];  
    _checkNear    = [ 0,     0,     0,    100,   200    ];  
    _checkObjH1   = [ 1.5,   0.5,   0.5,   0.2,   0.2    ];  
    _checkObjH2   = [ 30,    15,    15,    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 chosen (this parameter is optional)

["z6",[1,0,0],[8,2,80,12],[],[],[],[0,0,0,2]] spawn DAC_Zone

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

_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],[8,2,80,12],[],[],[],[0,0,0,2,2]] spawn DAC_Zone

```
case 2:
{
    _campBasic      = ["FlagCarrierCDF",["Land_Campfire_burning",8,5,0],["Camp",5,0,0],["Logic",10,15,0],0];
    _campAmmo       = [["GuerillaCacheBox",10,2,0],["LocalBasicAmmunitionBox",10,0,0],["LocalBasicWeaponsBox",10,0,0]];
    _campStatic     = [["DSHKM_Gue",-7,25,0,"GUE_Soldier_1"],["DSHKM_Gue",25,25,0,"GUE_Soldier_1"],["DSHKM_Gue",25,25,0,"GUE_Soldier_1"]];
    _campAddUnit    = ["GUE_Soldier_AT","GUE_Soldier_AT","GUE_Soldier_AA","GUE_Soldier_Sniper"];
    _campUserObj    = [["Land_Antenna",10,-15,45],["Logic",17,35,0],["Logic",17,20,0],["Logic",47,0,0],["Logic",47,0,0]];
    _campRandomObj  = [];
    _campWall       = ["Land_BagFenceLong",[-10,30],[40,56,0],[5,5,5,5],[1,0.2],[0,0]];
    _campObjInit    = [[],[ ],[ ],[ ],[ ],[ ],[ ],[ ]];
};
```

Note: The object types in the following pages are originally from Arma2

_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/ -5 = 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];

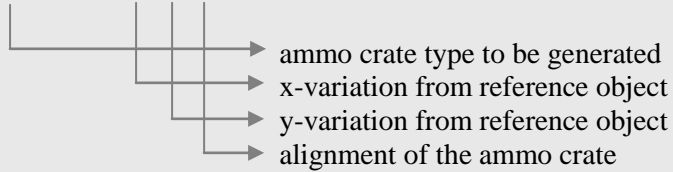
The last parameter in the array _campBasic controls the "self-maintenance" (applies to only 3 basis objects)
0 = self-maintenance deactivated, >0 = seconds until objects are repaired again.
Info: it happens that respawned vehicles destroy these basis objects. Therefore this parameter.

```
_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

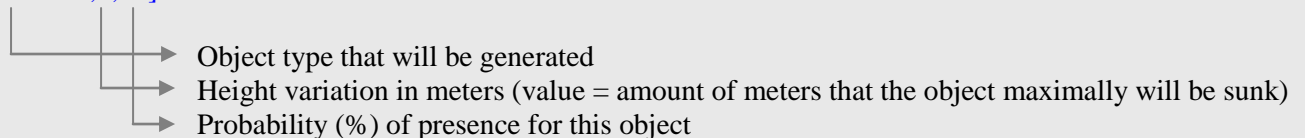
```
_campRandomObj = [{"AAPL048",1,60},{"AARO038",1,10},{"AARO041",1,10},{"AAPL068",3,20}],50,1,50,10];
```

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:

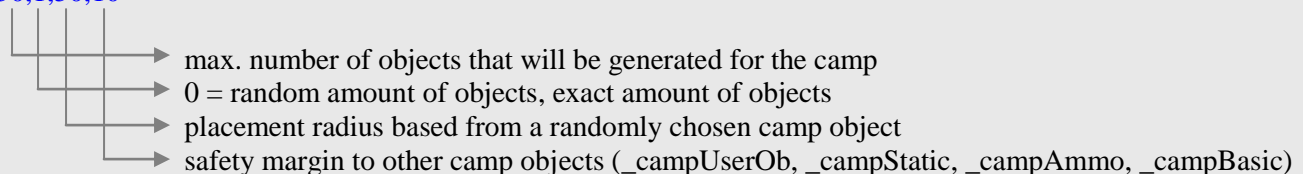
```
["AARO038",1,10]
```



```
_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 angles 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 5th 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],[5,3,30,8],[],[],[],[0,0,**1**,0]] spawn DAC_Zone

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 = [10,2,0,30];
  _set1 = [0,0,100,[ ],2,30];
  _set2 = [100,100,100,100,4,1,0];
  _set3 = [ ["D30_RU",["Sh_122_HE"]],["T72_INS",["Sh_125_SABOT"]]];
  _set4 = [2,5,1];
  _set5 = [10,30,1];
  _set6 = [10,30,1];
  _set7 = [0.1,0.5,1];
  _set8 = [ ];
  _set9 = [ ];
  _set10 = 2000;
};
```

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"]`

`_set10 = 2000;`

- The maximum range of the arti units.

[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 configurations, you can create any number of configurations. In principle, you can equip each zone with a different configuration.

<i>Event</i>		<i>Variable</i>		<i>Description</i>
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: All event entries have to be specified as string !!

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((getdamage _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],[5,3,50,8],[],[],[1,1,0,6]] spawn DAC_Zone`

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"""],
        []
    ];
};
```

Heli.

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.,

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 = 3 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 3:
{
    _setShowZones           = 2;
    _setShowWPs             = 1;
    _setShowUnit            = [1,1,1,1];
    _setGroupType           = 0;
    _setMarkerText          = ["if(isPlayer _unit) then {format[""%1"",_unit]}"];
    _setMarkerDel           = 1;
    _setMarkerRefresh       = [0.2,0.2];
    _setSizeWpLine          = [0.7,8];
    _setSizeLeaderLine      = 0.4;
    _setSizeZoneLine        = 2;
    _setSizeCampLine        = 2.5;
    _setSizeZoneBorder      = 4;
    _setArtiMarker          = 1;
    _setCampMarker          = 2;
    _setSideColor           = [
        "ColorRed",
        "ColorBlue",
        "ColorYellow",
        "ColorGreen",
        "ColorWhite",
        "ColorBlack",
        "ColorGreen",
        "ColorBlack"
    ];
};
```

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]
_setGroupType	0 = one marker per group / 1 = marker for each unit per group
_setMarkerText	Marker Text: formatted 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", [0.5,0.5], 1],
],
[
    [ "MAN" "mil_triangle", [0.6,0.6], 1],
],
[
    [ "CAR" "mil_box", [0.5,0.8], 1],
    [ "TRUCK" "mil_box", [0.5,0.9], 1],
],
[
    [ "TANK" "mil_box", [0.6,1.2], 1],
    [ "APC" "mil_box", [0.6,1.2], 1],
],
[
    [ "MOTORCYCLE" "Dot", [0.3,0.7], 1],
],
[
    [ "AIR" "mil_triangle", [0.7,1.5], 1],
    [ "HELICOPTER" "mil_triangle", [0.7,1.5], 1],
    [ "PLANE" "mil_triangle", [0.9,1.5], 1],
    [ "ParachuteBase" "mil_triangle", [0.9,0.3], 1],
],
[
    [ "SHIP" "Dot", [0.9,1.3], 1],
],
[
    [ "OTHER" "Dot", [0.7,0.7], 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)]

[DAC_Config_Objects]

The objects which can be generated by using the script **DAC_Objects** are defined in the config file **DAC_Config_Objects**. The respective Object definitions are located in the data blocks.

And again, it's possible as well to add as much further data blocks as you want. It's only important to always keep in mind that each data block needs a clear ID to be assigned (case no.)

While using such an own data block always make sure to define the respective ID in the script call:

nul = ["objZ1",300,0,0,**13**,0,[]," "] spawn DAC_Objects

Figure based on Arma2

```
case 13:
{
    _Object_Pool = [
        [0,1],
        ["MAP_R2_Boulder1",5,0,0.05,0,0,"",[0,"ColorBlack"]],
        ["MAP_t_picea1s",2,0,0.05,0,0,"",[0,"ColorGreen"]],
        ["MAP_t_picea2s",1,0,0.05,0,0,"",[0,"ColorGreen"]],
        ["MAP_R2_Boulder2",5,0,0.05,0,0,"",[0,"ColorBlack"]],
        ["MAP_R2_Stone",5,0,0.1,0,0,"",[0,"ColorBlack"]],
        ["MAP_b_betulaHumilis",6,0,0,0,0,"",[0,"ColorGreen"]],
        ["MAP_b_canina2s",6,0,0,0,0,"",[0,"ColorGreen"]]
    ];
};
```

It's possible to define as many objects as you wish in each data block, the only think you should do is keeping your fingers away from the Name **"_object_Pool"**. (do not change that name!)
The objects will be generated then in the respective zone.

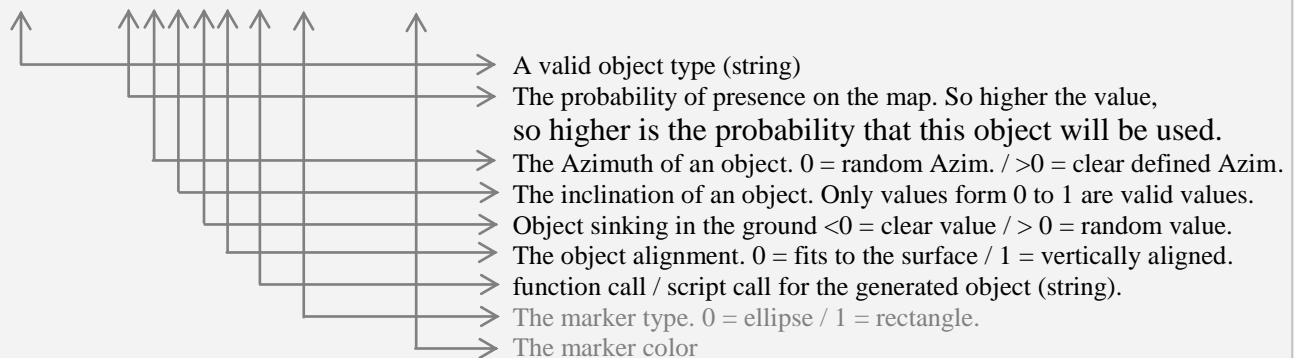
Please keep in mind that the first entry always has to be an Array with 2 logical values (numbers).
That's because these values are defining following options for each data block:

1. The minimum range (in meters) to street segments
2. The minimum height (meters) above sea level

Each single data line which is following then will contain following parameters:

Example:

["MAP_c_fern",1,0,0,0,0,"",[0,"ColorGreen"]]



- Please keep in mind that not all objects can be adjusted to the surface properties.
- To handover a generated Object to a script by using the parameters **function call / script call**, you can use the variable **_obj** as reference: " [_obj] execVM "myObjFunc.sqf" ... for example.
- The marker settings (last array) are optional and mustn't be defined.

[DAC_Config_Sound]

All the soundfiles are defined in the **DAC_config_Sound**, which are located in the DAC_sound.pbo.

```
DAC_SayArrayE = [
    /* reach waypoint */ [100,"r01","r02","r03","r04","r05","r06","r07","r08","r09","r01_0","r01_1","r01_2","r01_3","r01_4","r01_5","r01_6","r01_7","r01_8","r01_9","r01_10","r01_11","r01_12","r01_13","r01_14","r01_15","r01_16","r01_17","r01_18","r01_19","r01_20","r01_21","r01_22","r01_23","r01_24","r01_25","r01_26","r01_27","r01_28","r01_29","r01_30","r01_31","r01_32","r01_33","r01_34","r01_35","r01_36","r01_37","r01_38","r01_39","r01_40","r01_41","r01_42","r01_43","r01_44","r01_45","r01_46","r01_47","r01_48","r01_49","r01_50","r01_51","r01_52","r01_53","r01_54","r01_55","r01_56","r01_57","r01_58","r01_59","r01_60","r01_61","r01_62","r01_63","r01_64","r01_65","r01_66","r01_67","r01_68","r01_69","r01_70","r01_71","r01_72","r01_73","r01_74","r01_75","r01_76","r01_77","r01_78","r01_79","r01_80","r01_81","r01_82","r01_83","r01_84","r01_85","r01_86","r01_87","r01_88","r01_89","r01_90","r01_91","r01_92","r01_93","r01_94","r01_95","r01_96","r01_97","r01_98","r01_99"],
    /* detect enemys */ [100,"r11","r12","r13","r14","r15","r16","r17","r18","r19","r20","r21","r22","r23","r24","r25","r26","r27","r28","r29","r30","r31","r32","r33","r34","r35","r36","r37","r38","r39","r40","r41","r42","r43","r44","r45","r46","r47","r48","r49","r50","r51","r52","r53","r54","r55","r56","r57","r58","r59","r60","r61","r62","r63","r64","r65","r66","r67","r68","r69","r70","r71","r72","r73","r74","r75","r76","r77","r78","r79","r80","r81","r82","r83","r84","r85","r86","r87","r88","r89","r90","r91","r92","r93","r94","r95","r96","r97","r98","r99"],
    /* attack enemys */ [100,"r21","r22","r23","r24","r25","r26","r27","r28","r29","r30","r31","r32","r33","r34","r35","r36","r37","r38","r39","r40","r41","r42","r43","r44","r45","r46","r47","r48","r49","r50","r51","r52","r53","r54","r55","r56","r57","r58","r59","r60","r61","r62","r63","r64","r65","r66","r67","r68","r69","r70","r71","r72","r73","r74","r75","r76","r77","r78","r79","r80","r81","r82","r83","r84","r85","r86","r87","r88","r89","r90","r91","r92","r93","r94","r95","r96","r97","r98","r99"],
    /* call for help */ [100,"r31","r32","r33","r34","r35","r36","r37","r38","r39","r40","r41","r42","r43","r44","r45","r46","r47","r48","r49","r50","r51","r52","r53","r54","r55","r56","r57","r58","r59","r60","r61","r62","r63","r64","r65","r66","r67","r68","r69","r70","r71","r72","r73","r74","r75","r76","r77","r78","r79","r80","r81","r82","r83","r84","r85","r86","r87","r88","r89","r90","r91","r92","r93","r94","r95","r96","r97","r98","r99"],
    /* lost contact */ [100,"r41","r42","r43","r44","r45","r46","r47","r48","r49","r50","r51","r52","r53","r54","r55","r56","r57","r58","r59","r60","r61","r62","r63","r64","r65","r66","r67","r68","r69","r70","r71","r72","r73","r74","r75","r76","r77","r78","r79","r80","r81","r82","r83","r84","r85","r86","r87","r88","r89","r90","r91","r92","r93","r94","r95","r96","r97","r98","r99"],
    /* call for arti */ [100,"r51","r52","r53","r54","r55","r56","r57","r58","r59","r60","r61","r62","r63","r64","r65","r66","r67","r68","r69","r70","r71","r72","r73","r74","r75","r76","r77","r78","r79","r80","r81","r82","r83","r84","r85","r86","r87","r88","r89","r90","r91","r92","r93","r94","r95","r96","r97","r98","r99"],
    /* help positiv */ [100,"r61","r62","r63","r64","r65","r66","r67","r68","r69","r70","r71","r72","r73","r74","r75","r76","r77","r78","r79","r80","r81","r82","r83","r84","r85","r86","r87","r88","r89","r90","r91","r92","r93","r94","r95","r96","r97","r98","r99"],
];

DAC_SayArrayW = [
    /* reach waypoint */ [100,"u11","u12","u13","u14","u15","u16","u17","u18","u19","u20","u21","u22","u23","u24","u25","u26","u27","u28","u29","u30","u31","u32","u33","u34","u35","u36","u37","u38","u39","u40","u41","u42","u43","u44","u45","u46","u47","u48","u49","u50","u51","u52","u53","u54","u55","u56","u57","u58","u59","u60","u61","u62","u63","u64","u65","u66","u67","u68","u69","u70","u71","u72","u73","u74","u75","u76","u77","u78","u79","u80","u81","u82","u83","u84","u85","u86","u87","u88","u89","u90","u91","u92","u93","u94","u95","u96","u97","u98","u99"],
    /* detect enemys */ [100,"u21","u22","u23","u24","u25","u26","u27","u28","u29","u30","u31","u32","u33","u34","u35","u36","u37","u38","u39","u40","u41","u42","u43","u44","u45","u46","u47","u48","u49","u50","u51","u52","u53","u54","u55","u56","u57","u58","u59","u60","u61","u62","u63","u64","u65","u66","u67","u68","u69","u70","u71","u72","u73","u74","u75","u76","u77","u78","u79","u80","u81","u82","u83","u84","u85","u86","u87","u88","u89","u90","u91","u92","u93","u94","u95","u96","u97","u98","u99"],
    /* attack enemys */ [100,"u31","u32","u33","u34","u35","u36","u37","u38","u39","u40","u41","u42","u43","u44","u45","u46","u47","u48","u49","u50","u51","u52","u53","u54","u55","u56","u57","u58","u59","u60","u61","u62","u63","u64","u65","u66","u67","u68","u69","u70","u71","u72","u73","u74","u75","u76","u77","u78","u79","u80","u81","u82","u83","u84","u85","u86","u87","u88","u89","u90","u91","u92","u93","u94","u95","u96","u97","u98","u99"],
    /* call for help */ [100,"u41","u42","u43","u44","u45","u46","u47","u48","u49","u50","u51","u52","u53","u54","u55","u56","u57","u58","u59","u60","u61","u62","u63","u64","u65","u66","u67","u68","u69","u70","u71","u72","u73","u74","u75","u76","u77","u78","u79","u80","u81","u82","u83","u84","u85","u86","u87","u88","u89","u90","u91","u92","u93","u94","u95","u96","u97","u98","u99"],
    /* lost contact */ [100,"u51","u52","u53","u54","u55","u56","u57","u58","u59","u60","u61","u62","u63","u64","u65","u66","u67","u68","u69","u70","u71","u72","u73","u74","u75","u76","u77","u78","u79","u80","u81","u82","u83","u84","u85","u86","u87","u88","u89","u90","u91","u92","u93","u94","u95","u96","u97","u98","u99"],
    /* call for arti */ [100,"u61","u62","u63","u64","u65","u66","u67","u68","u69","u70","u71","u72","u73","u74","u75","u76","u77","u78","u79","u80","u81","u82","u83","u84","u85","u86","u87","u88","u89","u90","u91","u92","u93","u94","u95","u96","u97","u98","u99"],
    /* help positiv */ [100,"u71","u72","u73","u74","u75","u76","u77","u78","u79","u80","u81","u82","u83","u84","u85","u86","u87","u88","u89","u90","u91","u92","u93","u94","u95","u96","u97","u98","u99"],
];

DAC_SayArrayD = /* soldier die */ [100,"d01","d02","d03","d04","d05","d06","d07","d08","d09","d10","d11","d12","d13","d14","d15","d16","d17","d18","d19","d20","d21","d22","d23","d24","d25","d26","d27","d28","d29","d30","d31","d32","d33","d34","d35","d36","d37","d38","d39","d40","d41","d42","d43","d44","d45","d46","d47","d48","d49","d50","d51","d52","d53","d54","d55","d56","d57","d58","d59","d60","d61","d62","d63","d64","d65","d66","d67","d68","d69","d70","d71","d72","d73","d74","d75","d76","d77","d78","d79","d80","d81","d82","d83","d84","d85","d86","d87","d88","d89","d90","d91","d92","d93","d94","d95","d96","d97","d98","d99"];
```

Sounds of East (**DAC_SayArrayE**) and sounds of West (**DAC_SayArrayW**) can be defined in there, which can be used by the group leader in certain situations.

DAC has 7 different situations already defined. That's why there are 7 arrays which can be used. You can get information about each single situation by reading the comments which are located right in front of each Array.

Furthermore there's the Array **DAC_SayArrayD**, which has definitions inside about sounds which are used by units who are about to die ;-)

```
DAC_RadioArrayW = [
    /* reach waypoint */ [100,"c103","c104","c105","c106","c107","c108","c109","c110","c111","c112","c113","c114","c115","c116","c117","c118","c119","c120","c121","c122","c123","c124","c125","c126","c127","c128","c129","c130","c131","c132","c133","c134","c135","c136","c137","c138","c139","c140","c141","c142","c143","c144","c145","c146","c147","c148","c149","c150","c151","c152","c153","c154","c155","c156","c157","c158","c159","c160","c161","c162","c163","c164","c165","c166","c167","c168","c169","c170","c171","c172","c173","c174","c175","c176","c177","c178","c179","c180","c181","c182","c183","c184","c185","c186","c187","c188","c189","c190","c191","c192","c193","c194","c195","c196","c197","c198","c199"],
    /* getin order */ [100,"c067","c068","c069","c070","c071","c072","c073","c074","c075","c076","c077","c078","c079","c080","c081","c082","c083","c084","c085","c086","c087","c088","c089","c090","c091","c092","c093","c094","c095","c096","c097","c098","c099"],
    /* getout order */ [100,"c072","c073","c074","c075","c076","c077","c078","c079","c080","c081","c082","c083","c084","c085","c086","c087","c088","c089","c090","c091","c092","c093","c094","c095","c096","c097","c098","c099"],
    /* detect infantry */ [100,"c041","c042","c043","c044","c045","c046","c047","c048","c049","c050","c051","c052","c053","c054","c055","c056","c057","c058","c059","c060","c061","c062","c063","c064","c065","c066","c067","c068","c069","c070","c071","c072","c073","c074","c075","c076","c077","c078","c079","c080","c081","c082","c083","c084","c085","c086","c087","c088","c089","c090","c091","c092","c093","c094","c095","c096","c097","c098","c099"],
    /* detect vehicle */ [100,"c057","c058","c059","c060","c061","c062","c063","c064","c065","c066","c067","c068","c069","c070","c071","c072","c073","c074","c075","c076","c077","c078","c079","c080","c081","c082","c083","c084","c085","c086","c087","c088","c089","c090","c091","c092","c093","c094","c095","c096","c097","c098","c099"],
    /* attack enemy */ [100,"c012","c013","c014","c015","c016","c017","c018","c019","c020","c021","c022","c023","c024","c025","c026","c027","c028","c029","c030","c031","c032","c033","c034","c035","c036","c037","c038","c039","c040","c041","c042","c043","c044","c045","c046","c047","c048","c049","c050","c051","c052","c053","c054","c055","c056","c057","c058","c059","c060","c061","c062","c063","c064","c065","c066","c067","c068","c069","c070","c071","c072","c073","c074","c075","c076","c077","c078","c079","c080","c081","c082","c083","c084","c085","c086","c087","c088","c089","c090","c091","c092","c093","c094","c095","c096","c097","c098","c099"],
    /* call for help */ [100,"c031","c032","c033","c034","c035","c036","c037","c038","c039","c040","c041","c042","c043","c044","c045","c046","c047","c048","c049","c050","c051","c052","c053","c054","c055","c056","c057","c058","c059","c060","c061","c062","c063","c064","c065","c066","c067","c068","c069","c070","c071","c072","c073","c074","c075","c076","c077","c078","c079","c080","c081","c082","c083","c084","c085","c086","c087","c088","c089","c090","c091","c092","c093","c094","c095","c096","c097","c098","c099"],
    /* call for arti */ [100,"c027","c028","c029","c030","c031","c032","c033","c034","c035","c036","c037","c038","c039","c040","c041","c042","c043","c044","c045","c046","c047","c048","c049","c050","c051","c052","c053","c054","c055","c056","c057","c058","c059","c060","c061","c062","c063","c064","c065","c066","c067","c068","c069","c070","c071","c072","c073","c074","c075","c076","c077","c078","c079","c080","c081","c082","c083","c084","c085","c086","c087","c088","c089","c090","c091","c092","c093","c094","c095","c096","c097","c098","c099"],
    /* help positive */ [100,"c085","c086","c087","c088","c089","c090","c091","c092","c093","c094","c095","c096","c097","c098","c099"],
    /* help negative */ [100,"c080","c081","c082","c083","c084","c085","c086","c087","c088","c089","c090","c091","c092","c093","c094","c095","c096","c097","c098","c099"],
    /* arti positive */ [100,"c001","c002","c003","c004","c005","c006","c007","c008","c009","c010","c011","c012","c013","c014","c015","c016","c017","c018","c019","c020","c021","c022","c023","c024","c025","c026","c027","c028","c029","c030","c031","c032","c033","c034","c035","c036","c037","c038","c039","c040","c041","c042","c043","c044","c045","c046","c047","c048","c049","c050","c051","c052","c053","c054","c055","c056","c057","c058","c059","c060","c061","c062","c063","c064","c065","c066","c067","c068","c069","c070","c071","c072","c073","c074","c075","c076","c077","c078","c079","c080","c081","c082","c083","c084","c085","c086","c087","c088","c089","c090","c091","c092","c093","c094","c095","c096","c097","c098","c099"],
    /* lost contact */ [100,"c092","c093","c094","c095","c096","c097","c098","c099"],
    /* start searching */ [100,"c126","c127","c128","c129","c130","c131","c132","c133","c134","c135","c136","c137","c138","c139","c140","c141","c142","c143","c144","c145","c146","c147","c148","c149","c150","c151","c152","c153","c154","c155","c156","c157","c158","c159","c160","c161","c162","c163","c164","c165","c166","c167","c168","c169","c170","c171","c172","c173","c174","c175","c176","c177","c178","c179","c180","c181","c182","c183","c184","c185","c186","c187","c188","c189","c190","c191","c192","c193","c194","c195","c196","c197","c198","c199"],
    /* back to zone */ [100,"c023","c024","c025","c026","c027","c028","c029","c030","c031","c032","c033","c034","c035","c036","c037","c038","c039","c040","c041","c042","c043","c044","c045","c046","c047","c048","c049","c050","c051","c052","c053","c054","c055","c056","c057","c058","c059","c060","c061","c062","c063","c064","c065","c066","c067","c068","c069","c070","c071","c072","c073","c074","c075","c076","c077","c078","c079","c080","c081","c082","c083","c084","c085","c086","c087","c088","c089","c090","c091","c092","c093","c094","c095","c096","c097","c098","c099"],
    /* use smoke genade */ [100,"c118","c119","c120","c121","c122","c123","c124","c125","c126","c127","c128","c129","c130","c131","c132","c133","c134","c135","c136","c137","c138","c139","c140","c141","c142","c143","c144","c145","c146","c147","c148","c149","c150","c151","c152","c153","c154","c155","c156","c157","c158","c159","c160","c161","c162","c163","c164","c165","c166","c167","c168","c169","c170","c171","c172","c173","c174","c175","c176","c177","c178","c179","c180","c181","c182","c183","c184","c185","c186","c187","c188","c189","c190","c191","c192","c193","c194","c195","c196","c197","c198","c199"],
    /* soldier down */ [100,"c094","c095","c096","c097","c098","c099"],
    /* arti negative */ [100,"c131","c132","c133","c134"],
];
```

There're 2 new Arrays available, the array **DAC_radioArrayW** (for West) and the Array **DAC_RadioArrayE** (for East) where radio sounds are defined for different Situations. If the group is getting in such a predefined situation so the group leader will use one of these radio messages, which one can then listen to.

The more entries are defined in an Array, the more variations will it have.

The first value within an Array is always a numerally value, which has an overall meaning. It defines the probability to become a sound get played. A given value of **100** is equal with 100%. That enables one furthermore to become a sound totally deactivated for certain situations. To do this just set the value **0**.

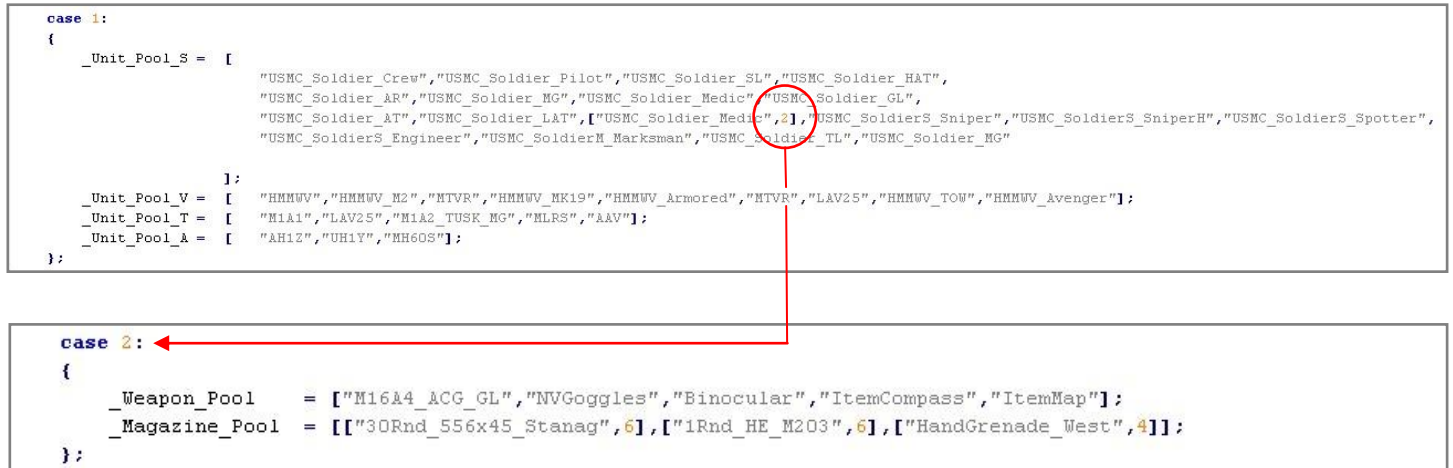
[DAC_Config_Weapons]

You can create as much weapon configurations as you want in the **DAC_Config_Weapons** to assign them to different unit types. The default weapons and ammunition configuration will be removed from the units as soon as they are created and will be replaced with the new configuration of your choice.

To assign a new weapon configuration to a special type of unit you have to make a small change in the **DAC_config_Units** file. That's why cause all the unit types are defined in there.

```
case 1:
{
    _Unit_Pool_S = [
        "USMC_Soldier_Crew","USMC_Soldier_Pilot","USMC_Soldier_SL","USMC_Soldier_HAT",
        "USMC_Soldier_AR","USMC_Soldier_MG","USMC_Soldier_Medic","USMC_Soldier_GL",
        "USMC_Soldier_AT","USMC_Soldier_LAT",["USMC_Soldier_Medic",2], "USMC_SoldierS_Sniper","USMC_SoldierS_SniperH","USMC_SoldierS_Spotter",
        "USMC_SoldierS_Engineer","USMC_SoldierM_Marksman","USMC_Soldier_TL","USMC_Soldier_MG"
    ];
    _Unit_Pool_V = [ "HMMWV","HMMWV_M2","MTVR","HMMWV_MK19","HMMWV_Armored","MTVR","LAV25","HMMWV_TOW","HMMWV_Avenger"];
    _Unit_Pool_T = [ "M1A1","LAV25","M1A2_TUSK_MG","MLRS","AAV"];
    _Unit_Pool_A = [ "AH1Z","UH1Y","MH60S"];
};

case 2:
{
    _Weapon_Pool = ["M16A4_ACG_GL","NVGoggles","Binocular","ItemCompass","ItemMap"];
    _Magazine_Pool = [{"3ORnd_556x45_Stanag",6},["1Rnd_HE_M203",6],["HandGrenade_West",4]];
};
```



The respective unit type has to be defined in the Array as shown in the image above. This Array furthermore needs to receive the additional **ID** out of the **DAC_config_Weapons**:

Default entry in the DAC_Config_Units : "USMC_Soldier_Medic"
Advanced entry in the DAC_Config_Units : ["USMC_Soldier_Medic",2]
With definition of the weaponconfiguration which has to be loaded

It's furthermore possible to assign a new weapon configuration, even for single units and for player units as well, just by using the following script call (Example):

[player,"2"] spawn DAC_Weapons

The Parameters:

1. The ID which has to receive a new weapon configuration
2. The ID out of the weapon configuration file (as string)

Attention,

this configuration file I could not adapt to arma3 until now. I can not say, whether the function is still running correctly. Use of this function is therefore at your own risk.

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 be 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 meaningful 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 sqf-scripts

The **DAC_Config_Camp** contains the Configuration **3**, which enables the AI-respawn even without Camp or Camp groups. DAC is generating only one position which will basically be used for one respawn only. You can define the exact position by using a user defined waypoint.

User advices & expression of thanks

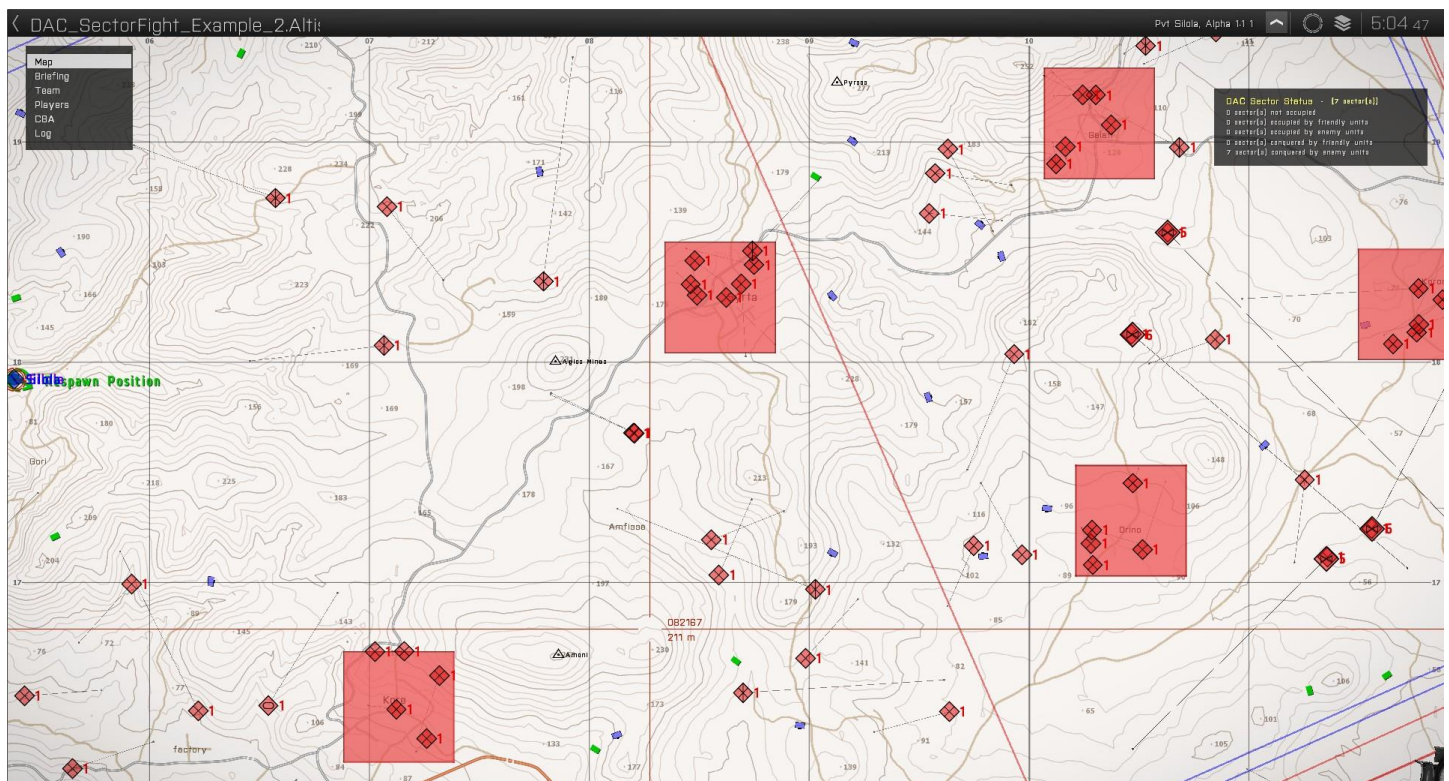
DAC is **not an official addon**. Please do not complain at BIS or their Publisher.

Attention: The using of the DAC V3.1 for Arma3 takes place on own danger!

At this point a **big thanks** to my testers and supporters, with them I have been tested DAC missions for many weeks and thus could solve many problems.

Thanks to: [MCPXXL](#) | [SKH|Flip](#) | [SKH|Cyborg](#) | [SKH|Hydra](#) | [t-800a](#) | [Lester](#)

Thanks also to Bohemia. They developed with Arma3 again a wonderful piece of software!



Attachement (sorry but german only):

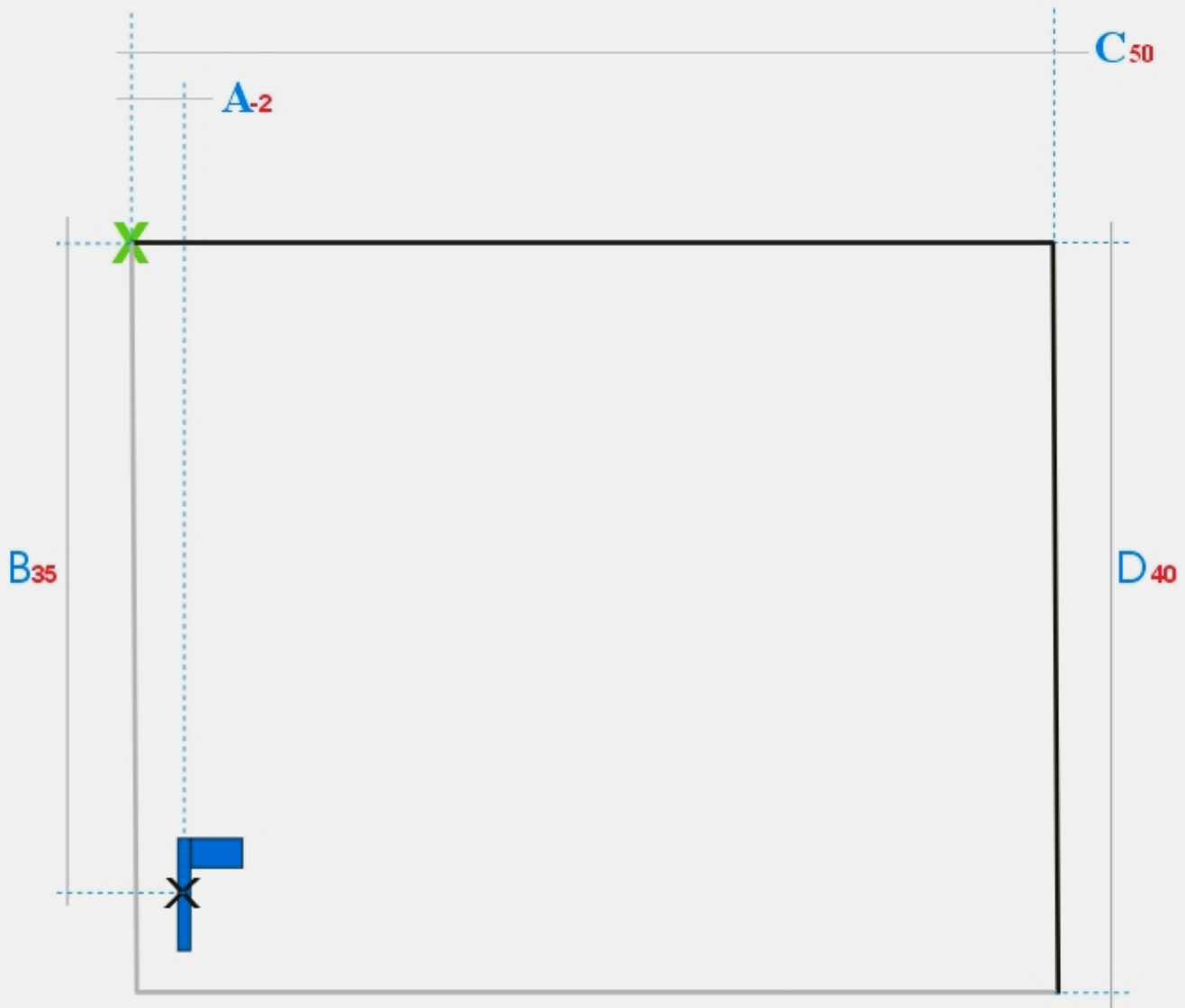
Here the mentioned „**figurative**“ description to the Array `_campWall` out of the script `DAC_Config_Camp`:

$\begin{matrix} & & \text{X} & & \\ & \swarrow & & \searrow & \\ \text{A} & & \text{B} & & \text{C} & & \text{D} \\ | & & | & & | & & | \end{matrix}$

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

Die Parameter **A+B** bestimmen die Startposition **X** für die Einfassung des Camps. Diese Einfassung wird immer im Uhrzeigersinn generiert. Referenzposition ist immer die Flagge (bei dem **X**).

Die Parameter **C+D** sind für die Größe der Einfassung zuständig. **C** = x-Ausdehnung, **D** = y-Ausdehnung



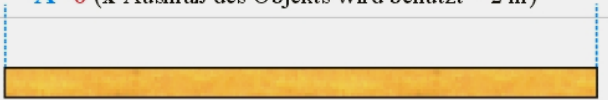
```
_campWall = ["FenceWood",[-2,35],[50,40,A0/1],[7,0,0,4],[1,0.1],[BC0,0]]
```

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.

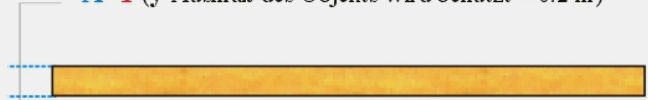
(Achtung, um festzustellen wie ein Objekt grundsätzlich ausgerichtet ist, mußt Du es nur einmal im Editor platzieren.)

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)



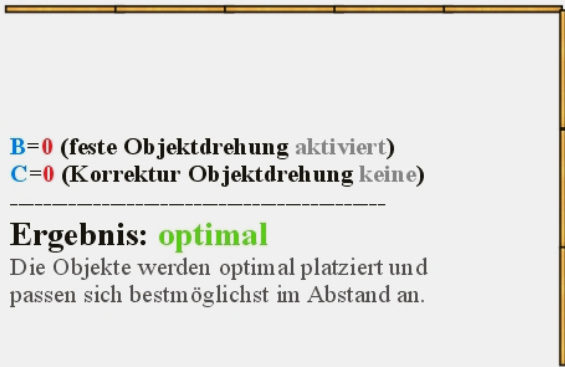
A=1 (y-Ausmaß des Objekts wird benutzt = 0.2 m)



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

Ergebnis: optimal

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



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

Ergebnis: sehr schlecht

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



```
_campWall = ["FenceWood",[-2,35],[50,40,A0B1],[7,0,0,4],[1,0.1],[B0C0]]
```

Dieses Beispiel zeigt genau die gleichen Einstellungen, 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.



`_campWall = ["FenceWood", [-2,35], [50,40,A1], [7,0,0,4], [1,0.1], [B0,C90]]`

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.

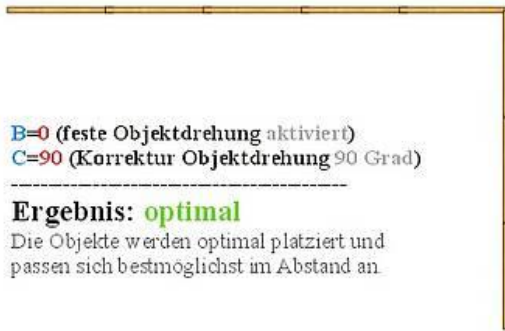
A=1 (y-Ausmaß des Objekts wird benutzt = 2 m)



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 entspricht der Anzahl der Segmente, die entfernt werden sollen.

