Dynamic-AI-Creator [DAC] by Silola

OFP 1.75* required as well as the addon : *cti_marker.pbo

Please don´t be intimidated by the extensiveness of this document as it contains an exhaustive explanation of all of the DAC's properties. The first three to five topics contain the core principles. Those who desire an extra thorough understanding of how the DAC works need only to proceed studying the remainder of the topics.

The summary shall help for an immediate start.

Content of the Zip:
DAC_Preview.pdf
DAC_Readme.pdf
DAC_Summary.pdf (short readme)
DAC_Editor_Examples 1-11 on Everon
DAC_folder (to copy to your missionroot)
DAC_MP_Demo on Everon
cti_marker.pbo

Topic overview:

-tips for DAC [**1**]

-what does DAC need in/out of the editor [**2**]

-first steps with demo1 (example with one zone) [**3-6**]

-DAC control marker [**6-7**]

-choice of side, the unit-addon for one zone,the AI-Behaviour [**7-8**]

-creating more than one zone (copy zones) [**8-9**]

-linking zones [**9-10**]

-creating waypoint zones [**10**]

-linking userdefined waypoints [**11**]

-de/activating zones [**12**]

-saving unit-categories [**13**]

-choosing AI-Addon/AI-Level [**13**]

-activating AI-Spawn [**14**]

-reducing of infantry-groups and caching of units [**15**]

-the config-files [**15-25**]

- multiplayer [**25**]

-tips & tricks [**26-27**]

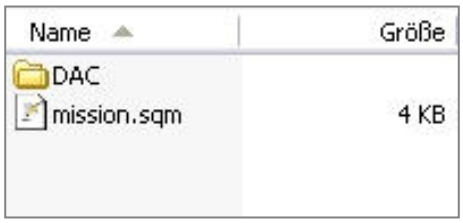
-thanks [**28**]

Tips for DAC

Those who've never heard about the DAC, should first take a look at the **DAC_Preview.pdf** to get a basic understanding of the core mechanics.

Please take your time reading the **DAC_Readme.pdf**. Thoughtful consideration of this document will enable a thorough understanding of the DAC system in all of its rich complexity. Most of the topics found in this pdf will be explained by added examples.

What the DAC needs in and out of the editor



All one needs to do is copy the DAC folder into your mission root of OFP and a "Description" or "Init.sqs" isn't required.

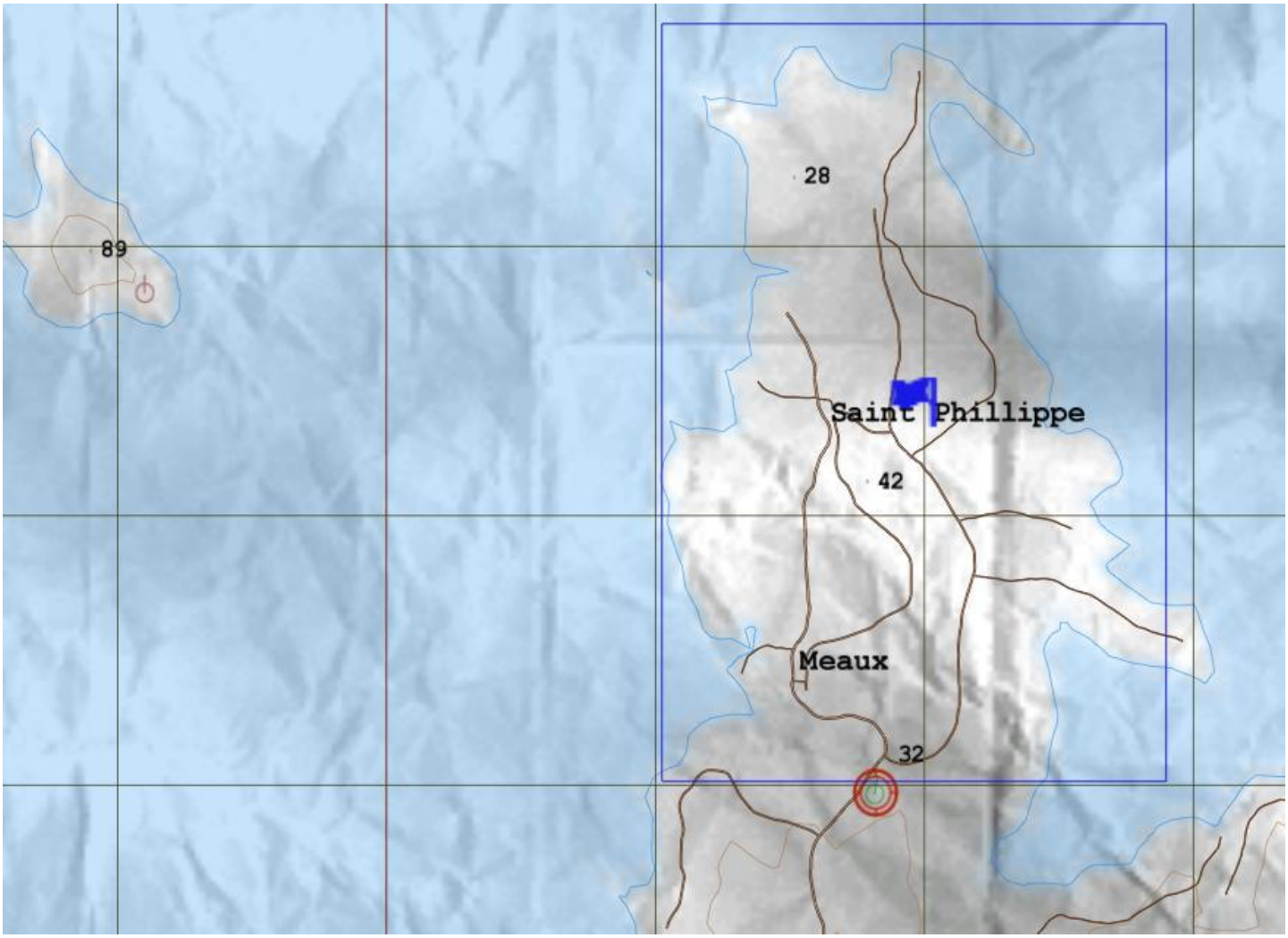
There is a small Addon required for DAC to work (**cti_marker.pbo**), and it is possible to use other third party Addons if these are installed and loaded.

Pic 1 explains the first situation in the demo **DAC_01_OneZone**. Pictured below is what is needed in the editor.

- Player named "s1"
- a dummy unit (on left top)
- at least one trigger for one zone

This is all that's needed to get started ☺

(Pic 1)



hint **player**

The player must be named so that the AI is able to react to the player or to his group. The name "s1" is the generalised name for the player in DAC. Other units of your side must be named in ...s2,s3,s4...etc.

If you need to change names (see example provided for in the existing mission), you have to register these names in the **DAC_Players** array, and they are found in the **DAC\DAC_Config_Creator.sqs.**

This array cleans itself, so as many entries as desired can be registered in this array (even individual ones); e.g. **DAC_Players = [s1,s2,s3,s5,ap1,ap2,ap3.....]** As soon as the mission starts, DAC reduces this array to used entries.

This only concerns units placed in the editor and on your side.

hint **dummy unit**

A dummy unit is always needed for each side, and there is a maximum of four dummy units that can be generated. For generating only east units, place just one east dummy unit. To generate east and west units one needs to place a dummy unit for each of the sides.

Attention: how the dummy units are named:

east unit = **E_Dummy** | west unit = **W_Dummy** | resistance unit = **R_Dummy** | civil unit = **C_Dummy**

First steps with the DAC Demo 1

The demo mission **DAC_01_OneZone** is just a demonstration of one zone, it's not a mission in itself.

An AI-Zone (trigger), like shown in the demo, needs the following adjustments...

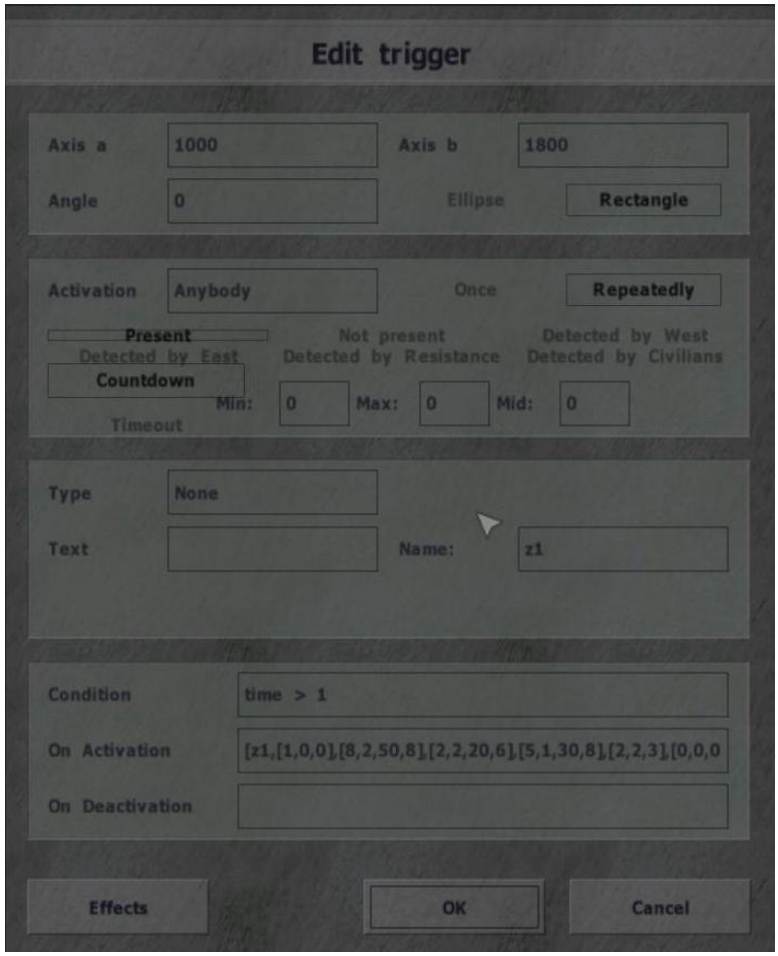
postion : **any**
axis a/b : **any**
angle : **0**
form : **rectangle**
activation : **everybody/multiple/present**
name : **name of the zone**
condition : **time > 1**

"**on activation**" → script call, that calls DAC

For deactivating a zone you can change the condition to : time > 100000000 (for testing)

The advantage is that all preferences of the zone stay the same and you can activate the zone as soon as you want to again.

You can change the triggers´size on the demo, or place it at another point on the map.



These changes are recognized *automatically* by the DAC and you just need to klick "preview" to generate the new changes.

The most challenging DAC feature to master is the zone configuration. The *script call* offers many variations and possibilities. Half of this readme.pdf is therefore concerned with this script call. But there's no need to panic as it is a very easy feature to manage ; -).

Each zone is generated by a script call:

[z1,[1,0,0],[ ],[ ],[ ],[ ],[0,0,0]]exec "DAC\Scripts\DAC_Init_Zone.sqs"

Let's now look at the coloured empty *arrays*. With these arrays the following unit categories will be created:

1.empty array generates infantry
2.empty array generates unarmed vehicles
3.empty array generates armed vehicles
4.empty array generates helicopters or spawncamps

These arrays need following parameters for DAC.....

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 (this is *not* an *exact* count of wp, as the DAC counts +/-1. For example: specifying "5" would give each group 4-6 waypoints)

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

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

Unarmed 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 (this is *not* an *exact* count of wp, as the DAC counts +/-1. For example: specifying "5" would give each group 4-6 waypoints)

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

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

Why no number for vehicles? → It is because this number is *dynamicly* generated. For example...

A group with 9 units is generated. Only after this procedure is final are the vehicles actually produced. E.g. only if a jeep is generated, there are only 4 units that will fit properly. Now the next vehicle is generated. If the remaining 5 units **fill up** the transport then no further vehicles are generated. However e.g. again if a jeep that holds 4 people was generated, then that means one person does not find a place. This situation then leads to a further vehicle being generated.

Armed 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 (this is *not* an *exact* count of wp, as the DAC counts +/-1. For example: specifying "5" would give each group 4-6 waypoints)

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

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

Helicopter

1. number of generating groups

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

(Only meaningful if the DAC AI was *activated*. Then an additional group is generated for the helicopter.)

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

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

In this zone 2 helicopters will be created, the group size is 2-6 units (at called upon conditions), a heli gets 3-5 waypoints during flight; after the waypoints have been completed, the helicopter will *land* at an *optimal* landing position, that is *not* necessarily the same position as where the helicopter started.

The group size is only responsed, if the DAC AI is active and the heli has cargo capacities. For example, other than the generated pilots, a AH64 won't carry any more personnel.

Take a look at the script call in **DAC_01_OneZone**:

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

8 groups with **infanterie**
(group size 2, whole 50 WP, 8 WP /group)

2 group with **unarmed vehicles**
(group size 2, whole 20 WP, 6 WP /group)

5 groups with **armed vehicles**
(group size 1, whole 30 WP, 8 WP / group)

2 groups **helos**
(group size 2, 3 WP / group)

If one has grasped these 4 arrays, then one has already come to understand the heart of the DAC system. The parameters are actually all identical, except when dealing with helicopters. If one desires to not provide a category to certain units, one then simply leaves the appropriate array *empty*. Practice with the demo **DAC_01_OneZone** to see how the groups, numbers, categories and waypoints can change.

DAC control marker

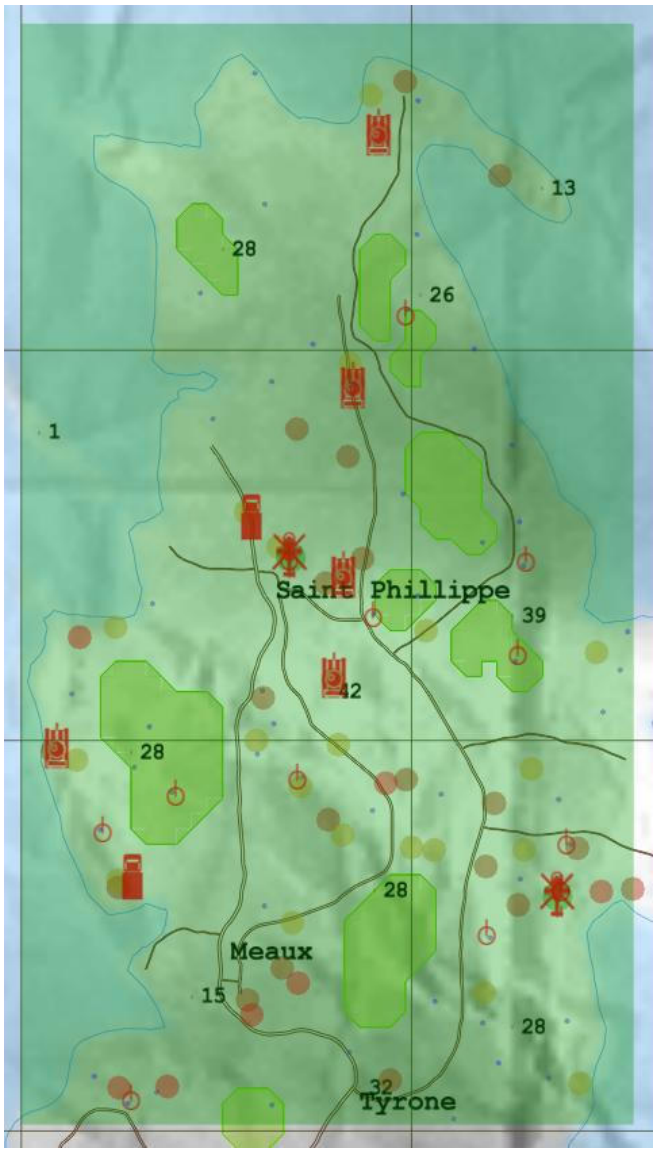
In almost every DAC Demo the *control markers* are activated. That means that you are able to watch the visual changes on the map that show how the DAC is working.

The control markers are a great means to study how the DAC is working during the initialisation.

By starting the Demo **DAC_01_OneZone** you can see how DAC....

- calcuates zones
- generates zones
- creates units

In the neighbouring image you can see the units actually following their waypoints.



We get 5 different *types* of *map* markers:

blue - waypoints for infantry
orange - waypoints for unarmed vehicles
red - waypoints for armed vehicles
green - start/landing zones for helicopters
grey - places for camps

Each unit's map markers are also colour coded for *allegiances*: **blue** (west) and **red** (east).

Please note that the map markers for units reflect just the group and *not* all the units in a group, thus the map marking system is limited in regards to displaying every unit in the zones.

To disable this characteristic of DAC, set the variable **DAC_Marker** to **false**. The variable will be found in **DAC\DAC_Config_Creator.sqs**

If you would like to use these map markers in your mission, you only have to copy the markers from the editor examples into your map. Without these markers one has to guess where the units could be. This means that *every* mission starts a *new* scenario!

Choice of allegiance/side, the unit-addons for one zone, and AI behaviour

The characteristics for AI allegiance, unit-addons and AI behaviour are summarized together since they are grouped together in the script call as an array. These characteristics are listed near the end of the script call and are denoted below in a light blue text.

From the first demo...

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

These parameter mean:

parameter1 = choice of allegiance/side for the units in this zone
parameter2 = choice of unit addon
parameter3 = choice of AI behaviour in this zone

choice of allegiance/side

There are four options for assigning allegiance to AI: **0** = east | **1** = west | **2** = resistance | **3** = civil

As can be seen in our example the parameter has been set to 0 which denotes that the allegiance has been assigned to the east.

Please remember that east units need the E_Dummy!

choice of addon units

The choice of units in a zone is regulated by numbers. Each number is one entry in the script... **DAC\DAC_Config_Units.sqs.**

In this script you will find many ready to place addon units. Every addon can be identified by its corresponding number.

This script can be extended at will. Each addon has its own number.

```
;\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\
#Unit_Config_0
;/////////////////////////////////////////

_Unit_Pool_S = ["SoldierECrew","SoldierEPilot","officerE","Soldiereb","Soldier
_Unit_Pool_V = ["Ural","UAZ","BRDM"]
_Unit_Pool_T = ["BMP","T72","T80","BMP","T72","ZSU","BMP","ZSU","BMP2"]
_Unit_Pool_A = ["MI17","MI24"]
_Unit_Pool_C = ["rus_vlajka.pac","CampEast","Shed"]

goto "SetPool"


;\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\
#Unit_Config_1
;/////////////////////////////////////////

_Unit_Pool_S = ["SoldierWCrew","SoldierWPilot","officerw","Soldierwlaw","Sold
_Unit_Pool_V = ["Jeep","Truck5t","Truck5tOpen"]
_Unit_Pool_T = ["M1Abrams","Bradley","M60","Vulcan","JeepMG"]
_Unit_Pool_A = ["Cobra","AH64","Ch47D","UH60"]
_Unit_Pool_C = ["usa_vlajka.pac","CampEmpty","Shed"]

goto "SetPool"
```

The pic reflects: **#Unit_Config_0** and **#Unit_Config_1.**

In order to deploy specific units in a zone, one must enter that unit's config number into the script call. One only needs to make certain that one has also installed and/or loaded the selected units.

Standard units are assigned to the following variables:

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

choice of basic behaviour

The last parameter of this array determines the basic behavior of the units in a zone. Basic behavior means how the generated units behave without any influence from enemy units.

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

Each zone can be configured with an individual basic behavior. Here's again the script call from demo1:

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

East side
Standard **east units**
Behaviour **config 0**

A detailed description on how to configure a unit's basic behaviour along with other adjustable options can be found in **the config files** topic.

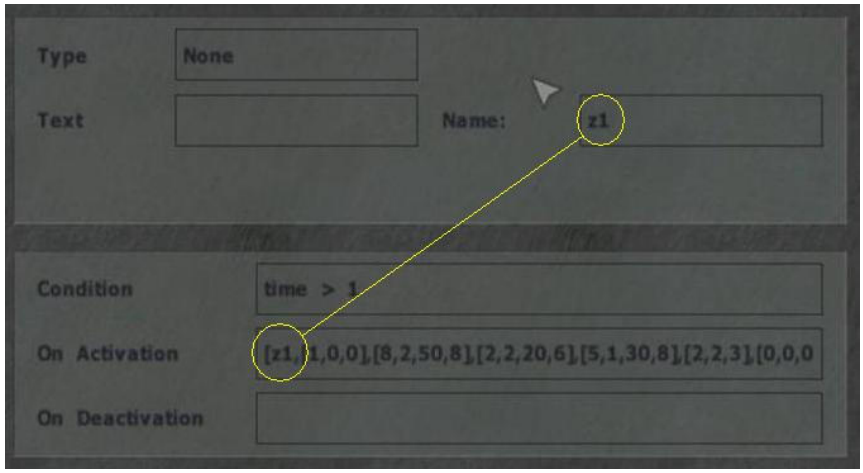
Create multiple zones

Providing more zones is a simple endeavour. One copies one zone and inserts it into another location.

It is *important* to enter the zones´ *name* in the script call →

If one copies a zone called "z1", this copy is *automatically* renamed to Ofp "z1_1".

One needs to insert this name to the **first** position in the script call!



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

If desired, the copied zone can now be changed in size, position and parameter. With this system, one can Create and configure zones in just a few minutes without having to leave the editor.

Suppose you have copied a zone 2 times and have already adapted the names of the zones and the scipt call, the script call from the zones could look like this:

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

These 3 zones (**z1, z2, z3**) are ones in which 3 infantry groups will be generated.

If you want the zones working locally you need to change one parameter.
I marked the parameter **red**; the red portion is the **ID** of the zone.

An example of what's stated immediately above:

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

In the demo **DAC_02_MoreZones** the principle of the local zones are shown.

Linking zones

Zones can be linked by folding up the waypoints of several zones. This means that the units will also have access to the waypoints of the linked zones. The **ID-Nr.** is used for linking zones. Immediately below are some examples:

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

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

Having the same waypoints mean that each of those 9 infantry groups has access to **60** waypoints. Which waypoints will be used by groups is choosen **randomly**. So now we have units exchanging between zones (e.g. **DAC_02_LinkedZones**).

More variations of linked zones:

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

This example also shows 3 linked zones, however in the zone **z2** armed vehicles are provided, and this zone is still linked with the two infantry zones. The armed vehicles won´t leave their zone, because in the zones the **z1** + **z2** waypoints are not at disposal. Keep in mind that it's *optional* how one decides to link and/or not link zones together.

another example:

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

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

Create waypoint zones

These waypoint zones have the purpose to give further waypoints to the units from the master zone.
In these waypoint zones no units are produced.You find an example in the demo **DAC_05_WPZones**.
By providing waypoint zones, the dynamics are increased. Here's an example:

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

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

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 there are 3 zones with **20** waypoints. Therefore, the generated groups from the master zone have access to a total of **70** waypoints (10+20+20+20).

Extra Emphasis: 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]]exec "DAC\Scripts\DAC_Init_Zone.sqs"
[z2,[1,0,0],[ ],[ ],[3,1,25,8],[ ],[0,0,0]]exec "DAC\Scripts\DAC_Init_Zone.sqs"
```

These zones are connected by the **ID** **1**, but the units have no possibility to come to the other zone, because there are no waypoints of the respective category that are shared.

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

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

Merge user defined waypoints

It is possible to integrate user-defined waypoints into the system.This function is always usefull, especially if one desires to merge firm positions, and from which the appropriate units can then be chosen.

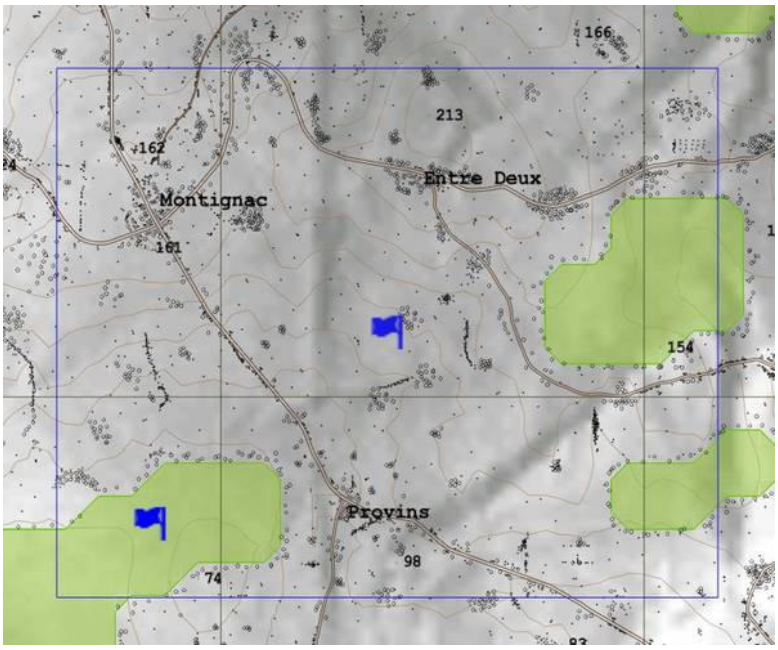
Such positions can be in a village (DAC has thereby some problems if buildings stand too closely together), or also at mission relevant positions.

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

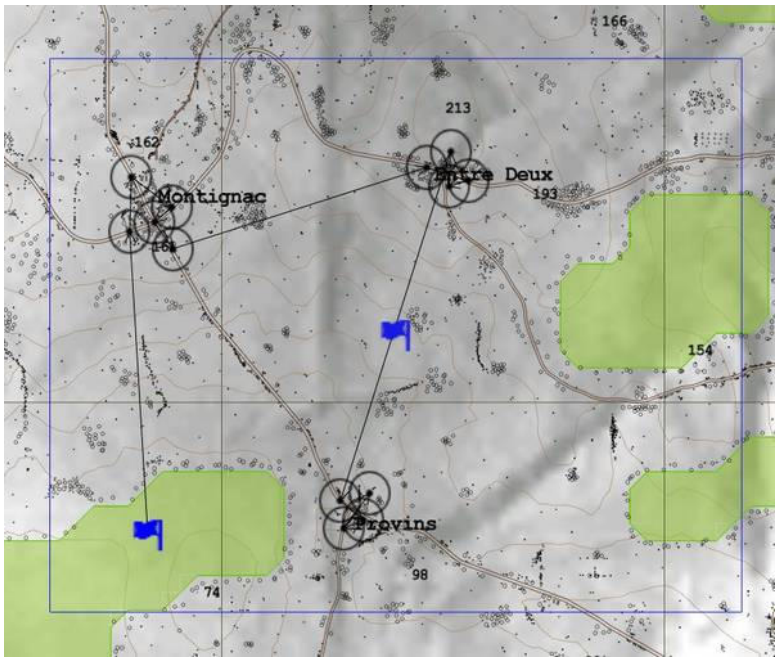
It is quite easy to construct user defined waypoints.

Place a logic-unit in the corresponding zone. →



It isn't critical to carefully define the sequence and the number of wayponts, as the DAC does all of this automatically. The only point to remember is that the waypoints only have to be **in** the zone.

→



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

Manage this by adjusting the settings of your logic. →

At "Azimut" you type a number between 1 and 5 (the variables are defined on the far right).



1. **infantry**
2. **unarmed vehicle**
3. **armed vehicle**
4. **helicopter**
5. **camps**

De/activate zones

Each zone can be activated and/or deactivated at a later time by again using a numeric code.
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 the 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". Doing so may save resources, thus boosting performance. The same goes for zones, which no longer have any meaning for the mission. Likewise zones can be deleted too for a performance boost.

The two **grey** marked parameters found below are examples of the de/activation process:

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

Parameter for the **activation**
Parameter for the **deactivation**

These two parameters are directly connected to the variable **DAC_Activate**. This variable is initialized with **0**. Therefore the two parameters were indicated as 0 up to now, so that the zones are generated directly at the beginning.

The units from the zone are only provided for the following example **if** the variable **DAC_Activate** gets the value **1**.You only have to ensure that this variable **DAC_Activate** gets the value **1** in every script and all zones will be automatically activated that get this value.

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

In regards to setting up parameters to delete the zones… Provided that the **DAC_Activate** = **1**, the units will be created in this zone, and with **DAC_Activate** = **2**, the units of this zone will be deleted....

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

And still another example: The zones **z1** + **z2** are generated directly at the beginning of the mission, because the value to activate is set to **0**. If the **DAC_Activate** gets the value **3** both zones are then deleted.
If the zone **z3** is activated, then **DAC_Activate** gets the value **1** and will not be deleted.
If the zone **z4** is activated, then **DAC_Activate** gets the value **2** and will not be deleted.

```
[z1,[1,0,3],[3,2,20,5],[ ],[ ],[ ],[0,0,0]]exec "DAC\Scripts\DAC_Init_Zone.sqs"
[z2,[1,0,3],[8,2,45,5],[ ],[ ],[ ],[0,0,0]]exec "DAC\Scripts\DAC_Init_Zone.sqs"
[z3,[2,1,0],[ ],[ ],[3,1,25,8],[ ],[0,0,0]]exec "DAC\Scripts\DAC_Init_Zone.sqs"
[z4,[2,2,0],[ ],[ ],[2,1,35,6],[ ],[0,0,0]]exec "DAC\Scripts\DAC_Init_Zone.sqs"
```

The **activation** and **deactivating** functions work only with **master** zones and has no influence on **pure** waypoint zones

Saving certain unit categories from one zone

It is possible to save the generated units of a zone and/or the generated user-defined array to save, to garantee later access to the groups or units.

This is possible with a "optional" parameter for each categorie. Here's an example:

In the zone **z1** 8 infantry groups are provided. In order to get access to the provided groups, further parameters are attached into the array for infantry.

```
[z1,[1,0,0],[8,2,50,5,''MyArray''],[ ],[ ],[ ],[0,0,0]]exec "DAC\Scripts\DAC_Init_Zone.sqs"
```

!!! Attention:The optional parameter needs to be in "quotation marks" !!!

From this "term" an array is produced, in which one then defines the groups of this category from this zone. The example above would therefore store the 8 **infantry** groups in the array titled "MyArray". It is important to designate these arrays meaningfully...e.g.. "Z1_inf" or "Z4_veh", only then is the affiliation recognizable by reading the name of the "array."

You can test this, by showing the contents of the array (ask only after the DAC has finished with the initialization): **hint format[,,%1",MyArray].**

It won't be any problem to determine the individual units and/or vehicles from the groups which are saved in the arrays.

Example for use:
You provide a zone with 3 groups of Shilkas with a group size of 1-2 vehicles. A mission goal could now be: **Destroy all Shilkas in this zone**. In order to determine, whether and when the Shilkas were destroyed you must test these groups or units for its condition. This is what the optional parameter is for.

Choosing KI-Addon/Choosing KI-Level

Unfortunately the AI Addon by KeyCat (GroupLink II) has not yet been implemented into the DAC yet, so we have at present only 2 adjustment possibilities which concern the AI:

DAC-KI-Addon → **ON** (watch demo mission DAC_10_Reaction)
DAC-KI-Addon → **OFF** (script performance is less,units behave like in editor placed ones)

You still have the possibility of adjusting the level of the DAC AI Addon in 4 stages. Details on these adjustment possibilities can be found under the topic: The configuration files.

Another characteristic of the DAC AI is that the units respond not only to your presence, but also to all hostile units. So e.g. a generated east troop will go also into covering, if it discovers a generated west troop.

It will attack it and will request also reinforcements for itself (if possible). If the DAC AI is activated, the unarmed vehicles will recon at their waypoints and the generated helicopters will additionally have a group of paratroopers with itself (presupposing that the helicopter has the capacity and equipment for them), and they'll jumps off to assist if the help is needed.

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 feel that this is a good compromise.

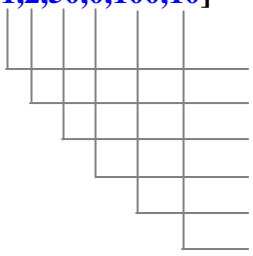
Only the following units are respawned: **Infantry**, **unarmed Vehicle**, **armed Vehicle**

Here is an example of a **Spawn camp**:

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

You need 6 parameters in order to provide one or more **Spawn camps** per zone. For **helicopters** you need at least 3 parameters, with the 4th parameter being optional. The meaning of the parameters:

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



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

[3,2,50,0,100,5] This would mean that in this 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 a group of the camp is extinguished then the remaining spawns are **lost**.

Attention: the more camps you would like to generate in a zone, then the more space that is needed. Give the zone enough free space for the number of camps that will be placed in a zone. If there's more than one camp in a zone, then the respawn will take place **randomly** in one of the camps.

If possible, as soon as the respawns are finished in a camp then the group stationed with a camp dissolves. The DAC will automatically mine the camp, so that it destroys itself a little later.

In each camp one will find an ammobox, whose contents generate themselves from the units of a zone. Assuming one generates a zone with RHS units and one also generates one or more spawn camps, then one will also find weapons and ammo of RHS units in the ammoboxes.

Please note: the respawn of a group takes time, for technical reasons this time span is 15-45 seconds between the time of death and recreation, and is based on the group's strength.

More details on adjustment possibilities can be found under the topic: The configuration files

Reducing infantry-groups and caching units

One must not underestimate these two capabilities of the DAC, as the reduction of infantry groups to a certain range is a way to save on system 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, the *remainder* of the group is *again* produced and the group is made *complete* again.

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

type of unit | skill of unit | dammage of unit | ammo and weapon

Only the number of magazines will be saved not each bullet.
Once the group is reduced to the leader unit, resourcess are saved because the engine doesnt have to make proximity checks for each unit of the group regarding its distance to the enemies.

Two examples for the editor on this topic: **DAC_06_ReduceInfantry + DAC_ReduceBigZone.**

The cache of units is directly connected to the respawn and reduction of the group. That is, if you make neither use of the respawn, nor of the group reduction, you can confidently deactivate this characteristic.

Everyone who has tried to create new units in a running mission, will possibly have noted some lag is produced because OFP has to load the "new units" characteristics: such as model, textures and configurations. In order to avoid this, place one of each possible type of unit in the editor, so that OFP already has the data of the units in the memory with the start of the mission.

With the respawn and also during the group reduction, new units are produced, and it thus makes sense to activate the cache of units in this connection. Other than activating "the cache" by parameters, one has to do nothing further. This work the DAC does for you. The DAC determines the possible unit types on the basis of the zones and produces a dummy unit for each case.

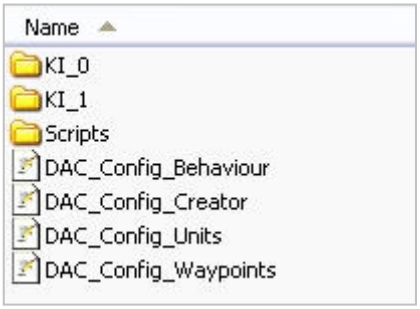
Please note: according to the different unit types, that you create, the DAC has to create the respective dummy units. This could take a negative effect to the performance.

You ´ll find adjustment possibilities in the next topic.

The config files

The configuration files are in the DAC main root. We have 4 different config scripts, that can be adjusted to one's own desires. Let's start with the global adjustments in **DAC_Config_Creator.sqs**:

DAC_Players = [s1,s2,s3,s4,s5,s6,s7,s8,s9,s10]
DAC_KI_Count_Level = [[2,4],[2,6],[2,9],[2,12]]
DAC_Reduce_Value = [600,650,0.1]
DAC_KI_Spawn = [30,60,0,120]
DAC_Delete_Value = [60,300,30]
DAC_KI_Cache = [1,1]
DAC_Dyn_Weather = [180,10,0.6]
DAC_Com_Values = [2,1]
DAC_KI_AddOn = 1
DAC_KI_Level = 4
DAC_Res_Side = 0
DAC_Marker = true



DAC_Players = [s1,s2,s3,s4,s5,s6,s7,s8,s9,s10]

The above array denotes which players and/or units of the player's group are stored.You can extend the array by arbitrary entries. The DAC settles this array automatically, and deletes redundant entries. It is important that the editor places units **on the player's side** get a name from the array so that the AI reacts to these units.

DAC_KI_Count_Level = [[2,4],[2,6],[2,9],[2,12]]

Here the group sizes are stored that the DAC has to generate. The array for producing infantry needs 4 parameters: **[5,2,50,6]** The 2nd parameter accesses array **DAC_KI_Count_Level**. 2 means 2-6 units are generated because the second entry in the array contains the appropriate values for it . There are always 2 values in the array **DAC_KI_Count_Level**. The 1st value is the minimum number of units that can be produced, the 2nd value corresponds to the maximum number that can be generated.

If you change the entries please consider limitating of the group size to 12 units. With armed vehicles the determined value is automatically multiplied by 3, to produce 3 crew members per vehicle. So generating 4 vehicles per group will produce the maximum number of personnel.

It is not a problem to supplement this array around with further entries. E.g. you can add to this entry, if the groups in a zone should exactly number 6 men ...

DAC_KI_Count_Level = [[2,4],[2,6],[2,9],[2,12],[6,0]]

In this example the array has an added 5th entry. In the script call of the zone one must indicate only the number **5**, so that one's groups contain exactly 6 units: **[5,5,50,6]**

DAC_Reduce_Value = [600,650,0.1]

As the name already states, these are the settings for group reduction. The 1st value specifies the distance developing the group. The 2nd value determines the distance for the group's reduction.

1st value = at this distance the units of a group are spawned, the distance is the distance between the next enemy unit and the leader unit of the group (the leader unit is never "reduced;" it's the unit that is present during the whole mission, until *killed*).

2nd value = at this closest range between a unit of a group and the next enemy unit the group gets reduced to only the leader unit.

The engine doesn't have to keep track of all the units of a group regarding the distance to the next enemy unit, since the group will be reduced to only the leader unit. This of course saves the system a good amount of resources, causing the engine to run smoother.

Attention, the 2nd value should be always bigger then the 1st value . So the reduction script inserts a small break, and is not permanently busy with "reduce" and "constructs" ;-)

The 3rd. value determines the "structure speed" of the group. The value **0.1** is actually a good choice.
On slower pc´s you can increase the value e.g. to **0.3**.

The group reduction is deactivated, by setting the "structure speed" to **0**.

DAC_KI_Spawn = [30,60,0,120]

The time performance and the type of the respawns are set by this array.

The 1st value determines the minimum time for the respawn .The 2nd value determines the time, that's added to the 1st value randomly. The above entry would trigger the respawn of a group after at least 30 seconds and 90 seconds max.

Attention: as mentioned already further above, the respawn needs a certain time until the old group can completely be provided again.This time window amounts to approx. 15-45 seconds, depending upon the size of the group.

It is not worth it to set the time performance of the respawns below this threshold.
DAC_KI_Spawn = [5,10,0,120] would mean that the respawn of a group needs between 5 and 15 seconds. It is however very improbable that this time limit, for reasons mentioned, is ever reached. Therefore leave the 1st value to a minimum of 30 seconds. The 2nd value can be adjusted as desired.

The 3rd value in this array, defines whether the respawn functions locally or globally...

0 = **global**: Each group can be respawaned, either at the original spawning point or along its waypoints. This adjustment is very useful, if you wish to send e.g. new units from elsewhere into the mission area. Thus each new group takes the waypoints of the old group, the new group will move to where the old group had its mission area.

1 = **local**: Only the groups, in whose *master* zone at least one Spawn Camp was generated can be spawned. Naturally spawns are available only as long as the camp is in working order.

The 4th value determines if a camp with no respawns left will be destroyed after a certain time. The ammo crate will always be preserved. After 120 seconds the self-destruct will be activated.....**0** = the camp is preserved | **>0** = the camp is destroyed in the given time.

DAC_Delete_Value = [60,300,30]

These values determine the deletion of units. This characteristic improves performance because too many "dead bodies" can cause lag.

The 1st value determines the time in seconds until a corpse will be deleted. The 2nd value is a distance value. It determines how far away from all the player's units the corpses will begin deleting. One can use both values individually, by deleting by time, or by deleting by distance.

The best variant is to combine both values, as in the example above: **[60,300,30]**

The unit will be deleted if 1st indicated time ran off, and 2nd the distance to all valid players is exceeded. That guarantes that nobody will be informed that a unit is deleted off the mission field. In addition you can "avail yourself" at the corpses, as long as they died within the indicated radius and the player doesn´t continues moving out of the radius.

The 3rd value is for dealing with empty but not yet destroyed vehicles, such as vehicles abandoned due to difficulties, or even if it takes a too lengthy investigation course, or all are killed in the vehicle. The vehicle is preserved.

Sometimes, especially if the respawn is used, many empty vehicles could be left on the battleground which would also unnecessarily slow performance. To avoid this, the vehicle destroys itself after this indicated time. If a vehicle can be destroyed, it can also be deleted. The example from above would mean that if a vehicle "loses" its crew, it destroys itself after 30 seconds. At the expiration of 60 seconds the distance is examined and the vehicle is deleted after exceeding the predefined distance limitation.

DAC_KI_Cache = [1,1]

Here are the terms for activating/deactivating units. If one neither uses the group reduction, or the respawn one can simply deactivate this function,.

The **1st** value specifies whether units of the type "soldier" are in the cache or not.
The **2nd** value specifies whether units of the type "vehicle" are in the cache or not.

1... means, the function is activated
0... means, the function is deactivated.

If you create a mission without vehicles, you can set the 2nd value to 0 to deactivate "the cache" for vehicles or reverse (1 -> activate). If one sees that “soldiers” are in the cache even though one has deactivated this type of unit, please be aware that this info is present so that *crews* for the vehicles are generated. To completely deactivate set both values to **0**.

DAC_Dyn_Weather = [180,10,0.6]

Here you’ll find the settings for the dynamic weather. The smallest adjustable weather value in OFP is 0.02.

This means that there are 50 weather steps between the span of 0 (best weather) to 1 (worst weather). The DAC uses these steps, to develop weather changes that evolve very gradually.

An example**: The current weather value is 0.4 and the weather is to change to 0.8. (a change of 20 steps)**

This change of weather is not done in one step, thus this avoids abruptly changing weather.

Back to the settings: **DAC_Dyn_Weather = [180,10,0.6]** is converted by DAC as follows...

Every 180 seconds a weather change will occur (determined randomly). The steps to change the weather will expire with a speed of 10 seconds. Our example: the change from 0.4 to 0.8, takes 200 seconds (10 sec. x 20 stages). The next change will then start after the 180 seconds expires again.

The 3rd value determines the maximum fog value. The fog is determined independently and randomly from the other weather effects so that all the possible weather variants can be developed.

DAC_Com_Values = [1,1]

We’ll now cover the settings for text insertions at the initialization of DAC, the system messages of DAC and the radiograms.

During the initialization of DAC the creation of the zones will be indicated, you see the progress of the waypoint and units generation.

You can fade out these progress announcements by setting the 2nd value of **DAC_Com_Values** to **0**. If you would like neither types of the DAC system messages appearing, or having the radiograms indicating, set the 1st value to **0**.

If the DAC shall only show the radiograms, which are faded in whenever certain reactions from the AI are generated, then set the 1st value to **1**. To also display the DAC system messages, set the 1st value to **2**. The DAC will then declare if units are deleted, units will be respawed, groups will be reduced and zones will be activated/deactivated.

DAC_KI_AddOn = 1

It is planned to make different AI Addons deployable for the DAC. In the future it will be possible to choose which AI addon will be loaded. For example, “Grouplink II” from KeyCat is scheduled to be integrated but it has not yet been converted yet.

For the moment only the DAC’s customized AI addon is available, and it holds its own pretty well. ; -)

TRANSLATORS note: the German term KI stands for “künstliche Intelligenz,” meaning AI (artificial intelligence). Since DAC internally uses KI in some variables we’re still referring to them correctly.

With **DAC_KI_AddOn = 1** the DAC AI will be loaded and with **DAC_KI_AddOn = 0** only the most necessary scripts will be loaded, to send e.g. the generated units from WP to WP.

To learn more about the DAC AI have a look at the preview.pdf and also the demo **DAC_10_AI_Reaction**.

In the demo **DAC_11_AI_Reaction** DAC shows that the AI does not only react to you, but also to all generated hostile units. This demo is a good example for many points for it shows how dynamically a DAC generated mission can develop and expire.

You can pursue everything on the map. The appropriate radiograms show the dynamic behavior of the AI. The DAC system messages inform you about the actions which the DAC settles on during the mission.

DAC_KI_Level = 4

You can select the KI level in 4 stages, 4 is the strongest level and 1 represents the weakest level.

At level 1 the units react only to hostile units. They have slow reactions and as reinforcements they rarely show up. So they actually behave like some "bloody rookies." ;-)

DAC_Res_Side = 0

Attention: in this build of the DAC it only supports the WEST or the EAST sides for the player!

In **DAC_Res_Side** one indicates on which side the resistance works, so editors please setup this parameter here. Please note that the editor setting for resistance "on both sides" is *not* supported by the DAC.

DAC_Res_Side = 0 means... the resistance fights on side of the east units
DAC_Res_Side = 1 means... the resistance fights on side of the west units.

This is a good way of supplying the east or west with more than 63 groups and it also opens access to activate zones later on. Thus this allows for some relatively large-scale missions.

An example: You play as west unit against the bad east. You would like to generate 4 zones with 20 groups each, that are gradually activated and/or deactivated on eastern side. In all zones the same units are generated, prefering RHS units right here. Now it is not possible to use RHS units on resistance side normally because they are firmly assigned in the editor to the east side.

But with DAC, you can generate all units on every side and so also RHS units on resistance side.

You only have to adjust DAC_Res_Side = 0, so that the resistance fight on eastern side, and so the variant above is possible.

DAC_Marker = true

This variable activates/deactivates the DAC markers. These markers are to indicate the initialization of DAC and the generated waypoints and units.

This way one can prove whether the generated result corresponds to one's designs, whether the waypoints are correctly placed, where the spawn camps are produced etc. You can also see the units moving within their assigned zones and you get a feeling of how the DAC has converted/translated your settings.

Please remember, you need the DAC markers, if you'd like to observe how the DAC has done its work on a map.

Better yet, copy these markers from any DAC editor *example* into your mission.

DAC_Config_Unit.sqs

This Config Script is very comprehensive. One will find many unit classes that can be addressed by the DAC. These classes are divided into "blocks" and each block possess a clear number.

Within a block there are 4 unit categories specified, that again are addressed by the DAC script call.

This script can be extended as desired, so you can also place your "favourite units" there.
You can combine units from different addons, and assign them as individual compositions of units.

A block looks as follows ...

```
;\\\\\\\\\\\\\\\\\\\\\\\\\
#Unit_Config_0
;/////////////////////////

_Unit_Pool_S = ["SoldierECrew","SoldierEPilot","OfficerE","Soldiereb","Soldiereg","Soldierelaw".....]
_Unit_Pool_V= ["Ural","UAZ","BRDM"]
_Unit_Pool_T= ["BMP","T72","T80","ZSU"]
_Unit_Pool_A= ["MI17","MI24"]
_Unit_Pool_C= ["rus_vlajka.pac","CampEast","Shed",[100,3]]
```

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

Below the Config No. **0** the standard east units are stored that are used in most DAC demos. I copied again the script call here, so you have the parameters easily at view, and it illustrates the setup of the different unit Config Numbers.

The No. **215** e.g. produces German BW Mod units, and thus these units are clearly identified to be placed into the 4 categories....

```
;\\\\\\\\\\\\\\\\\\\\\\\\\
#Unit_Config_215
;/////////////////////////
_Unit_Pool_S =  ["BWMOD_CrewT","BWMOD_HeloPilot","BWMOD_Officer","BWMOD_Soldier"...]
_Unit_Pool_V = ["BWMOD_MAN5tgl","BWMOD_MAN7tgl","BWMOD_MAN10tgl"]
_Unit_Pool_T = ["BWMOD_Leo1A5","BWMOD_Wiesel","BWMOD_Marder...."]
_Unit_Pool_A = ["BWMOD_EC135","BWMOD_BO105PAH1","BWMOD_BO105VBH"]
_Unit_Pool_C = []
```

following categories are found here:

_Unit_Pool_S = category **infantry**
_Unit_Pool_V = category **unarmed vehicle**
_Unit_Pool_T = category **armed vehicle**
_Unit_Pool_A = category **helicopter**
_Unit_Pool_C = category **spawn-Camps**

One can just create those categories, for which you have entries put.

Attention: even if you don't want to create infantry units you have a condtion to set to "true" in **Unit_Pool_S**:
One *always* has to have a set of 3 entries !

And it is setup this way for the following reason: these first 3 entries are ***reserved*** for DAC!

The **1st** entry places the type of unit for **the crew** of **armed vehicles.**
The **2nd** entry places the type of unit for **pilots** of **helicopter.**
The **3rd** entry places the type of unit for **the leader** of an **infantry** group.

In the case of doubt, if an addon does not contain suitable units, replace these by standard units or other suitable units. It doesn't matter, what kind of units are registered as long as they *are* registered.

Everything which comes after the 3rd entry in **_ the Unit_Pool_S** is intended for the random selection of units.
The more units you register, the more varied groups are generated.

If one would like to produce infantry groups, which consist of only one type of a unit in a zone, e.g. from LAW units, one has to do the following: **_ Unit_Pool_S** = **["SoldierECrew"," SoldierEPilot", "Soldierelaw", "Soldierelaw"]**

For our example, the first two units are of little importance, and our main focus is on the 3rd entry, the *leader* of the group, who is also a LAW unit. In the 4th entry, where the random choice begins, the LAW unit also needs to be registered.

Thus now the DAC only has to produce LAW units.

Tip: **If one puts a new "block" with unit classes from e.g. a new addon, then create the complete block *first*, with all unit types included in the addon. From this "complete block" one can now "derive" further blocks for use by the DAC. Thus one always has access to all units of the addon and one only needs to copy the suitable units from this "complete pool." Of course all derivations must have their own *unique* config number.**

Tip: **You can enter unit classes into the pool multiply . This increases the chance of creating this type of unit.**

I had almost forgotten this: _Unit_Pool_C = **["rus_vlajka.pac","CampEast","Shed",[100,3]]**

In this array you can give the data for the spawn camps to each config block:

The 1st entry contains the texture for the flag, which is set up in the camp.
The 2nd entry contains "building 1" that is set up in the camp . The building does not have any function.
The 3rd entry contains "building 2" that is set up in the camp. Here the units are respawned.
The 4th entry contains the settings for the "ammobox", which is generated in the camp:
1 st value = percentage figure, how many weapon-sets of this ammobox (inclusive ammo) in the zone are contained.
2 nd value = how many weapon-sets of each type are generated into the ammobox.

DAC_Config_Behaviour.sqs

In this Config Script different kinds of basic behaviour can be stored. They are loaded for each zone separately. "Basic behaviour" means, how generated units of a zone behave, how they process their waypoints without having hostile contact. Here you see a "block" from this script...

```
;\\\\\\\\\\\\\\\\\\\\\\\\\\\\
#Behaviour_Typ_3
;////////////////////////////
_setSkill      = [0.8,1]
_setCombat     = ["white","yellow"]
_setBehav      = ["aware","combat"]
_setSpeed      = ["normal","full"]
_setForm       = ["line","vee","column","wedge","stag column","ech left","ech right"]
_setFleeing    = [0]
_setFlighth    = [50,150]
_setPause      = [5,5,10,25,5]
_setSay        = []
_setSupport    = 1
```

```
[z1,[1,0,0],[ ],[ ],[ ],[ ],[0,0,3]]exec "DAC\Scripts\DAC_Init_Zone.sqs"
```

Here again the script call and the appropriate connection with the parameter. This zone is fed with the Behaviour_Typ_**3**. The settings are as follows:

_setSkill

This array always needs 2 entries...
1st entry = **minimal skill**, the units of a zone will get.
2nd entry = **maximum skill**, the units of a zone will get.

_setCombat

This array can be filled with the possible **combatmode values** und needs one entry at least. The units change their combatmode at each waypoint randomly.

_setBehav

This array can be filled with the possible **behaviour values** und needs one entry at least. The units change their behaviour at each waypoint randomly.

_setForm

This array can be filled with the possible **formation values** and needs one entry at least.The units change their formation at each waypoint randomly.

_setSpeed

This array can be filled with the possible **speed values** and needs one entry at least. The units change their speed at each waypoint randomly.

_setFleeing

This value determines the **behaviour of fleeing** of units.The value needs to be between 0 and 1

_setFlight
::::::::::::::::::::::::
This array needs 2 entries at least.
1st entry =**minimal height** for helos in that zone.
2nd entry =**maximum height** for helos in that zone.

_setPause
::::::::::::::::::::::::
This array needs 5 entries.
1st entry = max **pause intervall** at waypoints for infantry
2nd entry = max pause intervall at waypoints for umarmed vehicles
3rd entry = max pause intervall at waypoints for armed vehicles
4th entry = max pause intervall at start/landing zones for helicopters
5th entry = max pause intervall for units stationed at the spawn camp

_setSay
::::::::::::::::::::::::
You can put standart sounds or custom sounds in here. The leader of a group will randomly speek that sounds in certain situations. DAC stores 5 different situations therefore. For each situation you can put down several sounds from which will be choosen randomly.

You have got 5 situations:
1. the group **reaches its waypoint**
2. the group **detects hostile units**
3. the group is **under fire**
4. the group **orders support**
5. the group **looses enemy contact**

If you want to use this characteristics of DAC the array *_setSay* needs 5 arrays entered, even if you just want to use sounds for 1or 2 situations.

Example: _setSay **= [["rus3","rus4"],["rus6","rus10"],["rus2"],[ ],["rus11","rus14","rus19"]]**

The groups´leader says **"rus3" or "rus4",** if he reaches one of his waypoints.
The groups´leader says **"rus6" or "rus10",** if his group has enemy contact.
The groups´leader says **"rus2",** if his group is under fire.
The groups´leader says **nothing**, if his group orders support.
The groups´leader says **"rus11","rus14" or "rus19",** if his group looses enemy contact

If the AI Addon is deactivated, the situation 1 is active only -> if the group reaches a waypoint.

_setSupport
::::::::::::::::::::::::
This value determines if a unit will support other friendly units that are calling for help
With that you can force groups and units to not leave their zone so that zones are not empied by the time.
This only works when the DAC AI Addon is activated.

DAC_Config_Waypoints.sqs

In this **Config_Waypoints.sqs** the parameters for the generation of waypoint are put down. The settings are global to all zones. It is planned to make this Config expandable, just as the Configs for units and bevaviour, so that each zone can be filled with individual parameters .

If you must adapt these settings, e.g. for a special island, to check it use the **DAC_Marker**. The markers will show you how DAC converts your settings.

Well, in this script are 5 "blocks", and each block codes for a unit category. I explain the parameters on the basis of the block "tank", an armed vehicle.

On the next page the extraction...

```
;\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\
#WP_Typ_3
;----------> | Tank |
;////////////////////////////////
```

_CheckRadius = 40
_checkAreaH = 35
_checkMaxh = 1000
_checkMinH = 1
_checkNear = 0
_checkObjH = 14
_checkCount = 200

_CheckRadius = 40

That is the radius, in which a random position is scanned for objects. The larger this radius is the more time is needed for scanning. This radius indication of 15 for infantry units is relatively small, therefore waypoints for infantry are found substantially faster. That is because of the fact that the waypoints for infantry can be also in forests.

You have to set a larger value for vehicles, so that no waypoints are generated in wooded areas. That has the following reason: forests consist of compound segments (a segment contains again several trees). These segments are completely normal OFP objects.

Since these segments are relatively large, they are set accordingly wide from each other. So that the DAC scanner also really seizes each segment, the radius must be large enough. If the radius is too small sized, waypoints sometimes happen to be generated in the middle of the forest.

The values for helos and spawn camps are even larger. That excludes that no disturbing objects are in the radius... especially important for helos, because those need place; -) You can try to reduce the value for infantry. As smaller the radius as more probably the generated waypoints are in villages.

Attention, change this value only in steps of 5m, please. Possible values are ... 5.10.15.20.25.30 etc.

_checkAreaH = 35

This value determines the maximum difference in height within the scanned area. You can determine how rough the area may be for a waypoint. For helos and camps the range for a valid position must be rather flat. Otherwise a helo could be generated at a mountain-slope. The same applies to spawn camps. The area must be rather flat here, too.

_checkMaxh = 1000

That is the maximum elevator indication for waypoints. Above this height no waypoints will be generated. So you exclude that waypoints will be generated on unattainable heights.

_checkMinH = 1

That is the minimum elevator indication for waypoints. Below this value no waypoints will be generated. So you can ensure e.g. that no waypoints are generated directly at the beach or in water and so also no units will be generated there.

_checkNear = 0

This value "forces" a minimum distance between waypoints. That is particularly important for helos and spawn camps. So the spawn camps will not be generated directly next to each other but will be placed at a certain distance to each other.

Attention, with this value (if > 0), the generation of waypoints sometimes lasts substantially longer, since all waypoints are compared with each other.

_checkObjH = 14

This value determines the maximum hight of objects, so the scanner still permits it. If objects in the scan range are smaller/lower then the indicated value, the position will be recognized as valid waypoint. If an object in the scan range with an elevator value higher than the indicated value, the scan at this position will stop immediately and a new position will be checked.

At the category helo e.g., the value 4 is relatively low. That is absolutely necessary to prevent starting or landing positions from being disturbed by rather large objects. Bushes will be accepted, though.

_checkCount = 200

This value serves to limit finding a valid waypoint. So you (by an error message) will be informed promptly by DAC, if problems appear at the waypoint generation.

Ensure that you place the appropiate zone to a more advantageous position, or increase the dimensions of the zone so that DAC is able finding more possibilities.

Multiplayer

If you want to use the DAC in a multiplayer mission you need to place a GameLogic named Server in the editor. Using the DAC in multiplayer games you have the following limitations:

The status of the DAC initialization (measuring zones, creating units and waypoints) is not displayed.
The clients only get a message when DAC was started and when it's finished the intialization succesfully.
Radio calls and DAC system reports will not be displayed.
The sounds of the AI speaking in the different situations will not be heard.

If you want to start one or more scripts at the server or client side after DAC is ready, you can do it at the following point:

In the script **\DAC\DAC_Config_Creator.sqs** there's a place where you can enter the appripriate script calls.

```
#StartScriptOnClient
                    ← enter your script calls for all Clients here
exit


#StartScriptOnServer
                    ← enter your script calls for the Server here
exit
```

tips and tricks

You need to change the DAC_Config_Waypoints.sqs for the islands of "Malden" and "Tonal", so that waypoints will be generated correctly.The following value needs to be changed:

_checkObjH = 8 (Standard = _checkObjH = 14) at **#WP_Typ_2** and **#WP_Typ_3**

With DAC I was inclined to generate too many units... it is also quite simple. Best prefer to generate rather large zones with fewer units, than small zones with too many units. Also the group reduction feature temps to create more groups in the distance. But as you're getting nearer thos groups the groups are build up again and can cuase lag.

Remember, DAC can indicate only a certain number of waypoints by DAC_marker. If the contingent of markers is full, nothing more will be indicated. That sometimes leads to the impression that DAC has not generated all indicated waypoints and units.

You can save time at the initialization by excluding "measureing zones", if you indicate its size for each zone. DAC has two optional parameters ready:

standart : [z1,[1,0,0],[ ],[ ],[ ],[0,0,0]]exec "DAC\Scripts\DAC_Init_Zone.sqs"

adjustet: [z1,[1,0,0,500,800],[ ],[ ],[ ],[0,0,0]]exec "DAC\Scripts\DAC_Init_Zone.sqs"

Size of the X axis of the zone

Size of the Y axis of the zone

Attention that makes sense only, if you implement <u>all</u> zones with this optional parameters.

The disadvantage in changing sizes of the zones is, that these parameters have to be adapted.
Otherwise DAC "ignores" the size changes. That is not noticeable immediately.

It's possible as well to out source the script calls from the zones into an external script.
That could look like this:

```
;[]exec „CallZones.sqs“
@time > 1
[z1,[1,0,0],[3,2,20,5],[ ],[ ],[0,0,0]]exec "DAC\Scripts\DAC_Init_Zone.sqs"
[z2,[1,0,0],[3,2,15,5],[ ],[2,1,25,12],[ ],[0,0,0]]exec "DAC\Scripts\DAC_Init_Zone.sqs"
[z3,[2,0,0],[ ],[ ],[3,1,5,8],[ ],[0,0,0]]exec "DAC\Scripts\DAC_Init_Zone.sqs"
[z4,[2,0,0],[ ],[ ],[25],[ ],[0,0,0]]exec "DAC\Scripts\DAC_Init_Zone.sqs"
[z5,[3,1,0],[ ],[ ],[3,1,20,8],[ ],[0,0,0]]exec "DAC\Scripts\DAC_Init_Zone.sqs"
[z6,[4,1,0],[ ],[ ],[ ],[1,2,50,0,100,10],[0,0,0]]exec "DAC\Scripts\DAC_Init_Zone.sqs"
.....
exit
```

In the zones you won't need the script call nor the condition (time>1) anymore. The only thing is to take care that all zone names in the script and in the trigger corespond.

The advantage of this is on the one hand to execute changes at only one place and on the other hand to insert variables and conditions as well. E.g.

```
;[]exec „CallZones.sqs“
_KiLevel        = 4
_KiType         = 33
@time > 1
[z1,[1,0,0],[3,2,20,5],[ ],[ ],[ ],[0, _KiType, _KiLevel]]exec "DAC\Scripts\DAC_Init_Zone.sqs"
[z2,[1,0,0],[3,2,15,5],[ ],[2,1,25,12],[ ],[0, _KiType, _KiLevel]]exec "DAC\Scripts\DAC_Init_Zone.sqs"
[z3,[2,0,0],[ ],[ ],[3,1,5,8],[ ],[0, _KiType, _KiLevel]]exec "DAC\Scripts\DAC_Init_Zone.sqs"
[z4,[2,0,0],[ ],[ ],[25],[ ],[0, _KiType, _KiLevel]]exec "DAC\Scripts\DAC_Init_Zone.sqs"
[z5,[3,1,0],[ ],[ ],[3,1,20,8],[ ],[0, _KiType, _KiLevel]]exec "DAC\Scripts\DAC_Init_Zone.sqs"
[z6,[4,1,0],[ ],[ ],[ ],[1,2,50,0,100,10],[0, _KiType, _KiLevel]]exec "DAC\Scripts\DAC_Init_Zone.sqs"
.....
exit
```

The two variables _KiLevel + _KiType might be global as well. So they can be called from a dialog or the multiplayer settings.

DAC deals with stationary units as well. ☺ Best is to write an extra **Unit_Config.** See the example below. Enter the stationary unit under _Unit_Pool_T. Then enter the first entry in _Unit_Pool_S with a unit of your choice. This unit will serve as crew later on. Regularily there's a regular crew to man armed vehicles.

```
;\\\\\\\\\\\\\\\\\\\\\\\\\\
#Unit_Config_501
;//////////////////////////

_Unit_Pool_S   = ["Soldiereb "," Soldiereb "," Soldiereb ","Soldiereb"]
_Unit_Pool_V   = [ ]
_Unit_Pool_T   = ["tpillboxeast"]
_Unit_Pool_A   = [ ]
_Unit_Pool_C   = [ ]

goto "SetPool"
```

Credits

First of all I'ld like to thank BIS ☺ what would we be without them? ;-)

I'ld like to thank the whole Mapfact-Team as well for cooperating very good and giving me the time to develop DAC.

Special thanks go to:

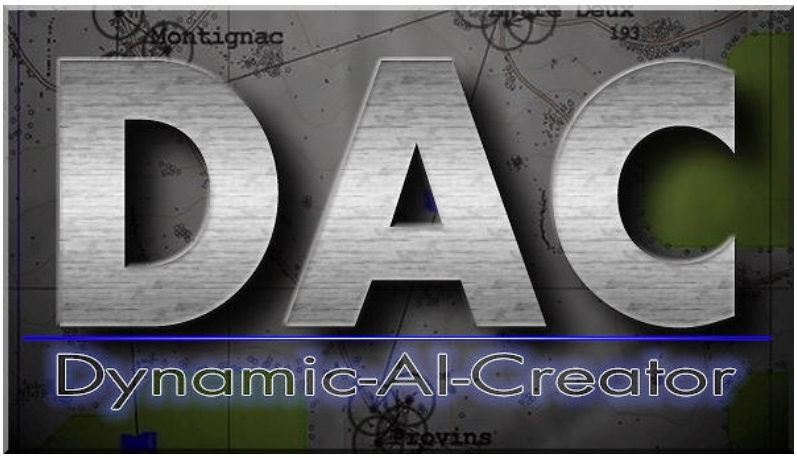
MCPXXL	DACWriter, Unit_Config, testing
Simba	translation, testing
BadAss	translation, testing
Mr.Murray	testing
SnipingJack	DAC-Logo
Aron	testing
Honchoblack	testing
Sanctuary	testing
Tacrod	testing
Dr.Strangelove	testing
Kyle_K_ski	proofreading thru pg. 21

Hopefully I have mentioned all, but if I've forgotten some one, I'm sorry!

Many thanks !!!

DAC is inofficial so do not blame BIS or Codemaster.
Don't blame any computer problems on me!
Installing and using of DAC is at your own risk!

You see DAC has a lot of possibilities.
This readme file was really hard and long
and everyone who finished reading this file,
has interest in DAC, but i think it is well worth it.



Greeting
Silola

silola@mapfact.net
silola@freenet.de

ICQ: 287-388-751

Here a few more clues what I would move with pleasure in the next version of DAC:

 # Movable zones.
 # Waypoint settings for each zone.
 # Object generation in the zones.
 # GroupLink II v. KeyCat