DAC V3.0 Bonus Functionality - by MadRussian

Description of the six bonus functions

The package contains this readme and 2 additional demo missions.
In each of these missions, the main DAC package is utilized.
This Readme contains a thorough reference for using the bonus material.

The following Addon is needed: DAC_Source (which requires MAP_EU_V1.00)

# DAC_fInsideOfZones

This new function allows you to determine with precision which zones a point resides inside of.

It is important to note that starting with DAC v3.0, a "location" now underlies every new DAC zone, and with this function it is possible to retrieve either the list of zones or locations the point resides inside of.

This function works with Rectangle and Ellipse zones, but not polygon zones. DAC must already be initialized, in order to use this feature. Here is the script call:

```
_zones = [position, info_requested] call DAC_fInsideOfZones;
```

1. any position (array)
2. info requested (can be "Zones" or "ZoneLocs")

Examples:
```
_zones = [position player, "Zones"] call DAC_fInsideOfZones;
_locations = [position player, "ZoneLocs"] call DAC_fInsideOfZones;
```

In addition, you can get a zone's corresponding location, or vice-versa:

Examples:
```
_zoneLoc = z1 getVariable "DAC_ZoneLoc";
_zone = _zoneLoc getVariable "DAC_Zone";
```

# **DAC_fReadMarkers**

With this function it is possible to quickly create a multitude of custom positions and read them into an array via a single command. Note that this may be useful inside and outside the DAC.

First, create a marker in the editor. (Using a small icon like a "Dot" is recommended.) After naming this initial marker, copy and paste to quickly create as many custom positions as desired.

When your set of positions is ready, here's the script call:

```
_points = [base_string, info_requested, blank_limit, delete] call DAC_fReadMarkers;
```

1. base string (string) - the name of the first marker in the set
2. info requested (string) - one of the following: "Pos", "Pos&Dir", "Pos&Dir&Txt", or "Pos&Dir&Txt&Name"
3. blank limit (numeric) - the number of marker "gaps" to ignore, 1000 recommended
4. delete (boolean) - whether or not to delete the markers upon reading them

Examples:

_positions = ["WayPt", "Pos", 1000, true] call **DAC_fReadMarkers**;
would return an array of positions with basename "WayPt", and delete the set of markers

_posDirPairs = ["StartingVehics", "Pos&Dir", 1000, false] call **DAC_fReadMarkers**;
would return an array of position/direction pairs with basename "StartingVehics", and not delete the set of markers

Hints:

Feel free at any time to delete any markers within a set (and perhaps come back later to paste away again) without fear of DAC_fReadMarkers missing any positions. That's what the "blank_limit" is for. For example, assuming the markers present on the map are "WayPt", "WayPt_1", "WayPt_2", "WayPt_4", and "WayPt_7", using a blank limit of 2 or more would return info on all five markers. Alternately, using a blank limit of 1 would return info on the first four markers only.
Best to set this value to 1000 and forget!

Keep your base markers for each marker set in a somewhat memorable location on the map, or risk losing them in a sea of points!

For a clean briefing map, read your markers in early in your "init.sqf" (and before any "sleep" commands). This way, all of your custom markers will be deleted *before* the player even sees the briefing.

Best of all, observe that creating a huge number of custom marker positions via this copy/paste/delete/paste-some-more-without-worry method is extremely fast!

# DAC_fGetZoneWaypoints

Here you can directly retrieve the waypoints inside of any DAC zone.
DAC must already be initialized, in order to use this feature. Here is the script call:

```
_positions = [zone, zone_type] call DAC_fGetZoneWaypoints;
```

1. zone
2. zone type (can be "S", "V", "T", "A", or "C")
    "S" for Soldier points
    "V" for Vehicle points
    "T" for Tracked points
    "A" for Aircraft points
    "C" for Camp points

Examples:

```
_positions = [z1, "S"] call DAC_fGetZoneWaypoints;
_positions = [z1, "T"] call DAC_fGetZoneWaypoints;
```

# **DAC_fReplaceZoneWaypoints**

It is also possible to directly replace the waypoints inside of any DAC zone.

Units inside the zone will begin using the new waypoints, within several seconds of the function call. When Camp points are replaced, the zone must be deactivated, then reactivated for the camps to relocate.

DAC must already be initialized, in order to use this feature. Here is the script call:

```
_result = [zone, zone_type, replacement_waypoints, max_replace (optional)] call
DAC_fReplaceZoneWaypoints;
```

1. zone
2. zone type (can be "S", "V", "T", "A", or "C")
   "S" for Soldier points
   "V" for Vehicle points
   "T" for Tracked points
   "A" for Aircraft points
   "C" for Camp points

3. replacement waypoints (array) -	Replacement set of waypoint positions
4. max replace (numeric, optional) -	Maximum number of waypoints replaced. If used, the function will randomly select this number of waypoints from the replacement set. If omitted, the entire replacement set is used.

Returned _result consists of:
[replacement_report, actual_replaced, of_total, _type, applied_positions] where:

replacement_report (string) =	Report of the operation
actual_replaced (numeric) =	Number of waypoints actually replaced
of_total (numeric) =	Total number of zone waypoints (of the given WP type)
type (string) =	Type of waypoint operation
replaced_positions (array) =	Array of actual waypoints replaced

Examples:

[z1, "V", _customVehPoints] call **DAC_fReplaceZoneWaypoints**;
would replace from z1, as many Vehicle type points as possible with points from _customVehPoints

[z1, "C", _customCampPoints, 10] call **DAC_fReplaceZoneWaypoints**;
would replace from z1, as many Camp type points as possible with points from _customCampPoints, up to a maximum of 10.

# **DAC_fReadAllCustomPoints**

This function allows reading in of all custom waypoint marker types simultaneously in-mass, and is designed for use with DAC_fApplyPointsWithinZone.

DAC must already be initialized, in order to use this feature. Here is the script call:

_result = [**base_S, base_V, base_T, base_B, base_A, base_C, delete, blank_limit**] call **DAC_fReadAllCustomPoints**;

1. base_S (string) - base name for Soldier points
2. base_V (string) - base name for Vehicle points
3. base_T (string) - base name for Tracked points
4. base_B (string) - base name for Both points (both Tracked and Vehicle, see below for more info on this special category)
5. base_A (string) - base name for Air points
6. base_C (string) - base name for Camp points
7. blank_limit (numeric) - the number of marker "gaps" to ignore per type, 1000 recommended
8. delete (boolean) - whether or not to delete markers for each type, upon reading them

Returned _result consists of a meta-array of the (point) locations read in. (Note - These locations are used internally by the DAC, and are separate from the area zone locations mentioned above in DAC_fInsideOfZones. As they may be useful, they are returned.**)**

Examples:

["ptS", "ptV", "ptT", "ptB", "ptA", "PtC", 1000, true] call **DAC_fReadAllCustomPoints**;
would read in all types of custom waypoints, and delete all the markers

["ptS", "ptV", "ptT", "", "", "", 1000, false] call **DAC_fReadAllCustomPoints**;
would read in the only the first three types of custom waypoints, and not delete any markers

Notes:

As mentioned above, the "Both" type has special properties. You may note that tracked vehicles can go pretty much anywhere wheeled vehicles can go? This is where "Both" waypoints come into play. When a "Both" type marker is read in, it will become a Vehicle or a Tracked point (with a 50% chance of either). The intent is to allow care-free land-vehicle placement of custom points, while minimizing the amount of redundant clutter in the editor.

When creating an island-full of custom points, you may wish to keep the resulting marker layout in its own separate mission as a template, and later "merge" the layout into future DAC missions.

Refer to the included demo mission "DAC_quick_custom_points_2" to see this dynamic custom point system in action.

# DAC_fApplyPointsWithinZone

With this function you can apply mass replacement of the specified types of custom waypoints simultaneously to the target zone. It is designed for use with DAC_fReadAllCustomPoints.

DAC must already be initialized, in order to use this feature.
IMPORTANT - For this function to work, DAC_fReadAllCustomPoints must be utilized first!

Here is the script call:

```
_result = [zone, paras_S, paras_V, paras_T, paras_A, paras_C] call
DAC_fApplyPointsWithinZone;
```

1. zone
2. [usage, cap] for Soldier point replacement
3. [usage, cap] for Vehicle point replacement
4. [usage, cap] for Tracked point replacement
5. [usage, cap] for Air point replacement
6. [usage, cap] for Camp point replacement

where:
    usage (numeric, 0..1) = the amount of custom points detected within the zone to apply
    cap (numeric) = the maximum number of zone waypoints to replace

Returned _result consists of:
[application_report, applied_positions] where:
application _report (string) = Report of the operation
applied_positions (array) = Meta-array of all position replacements applied

Examples:

_result = [z1, [0.4, 10], [0.4, 10], [0.4, 10], [0.4, 10], [0.4, 10]] call **DAC_fApplyPointsWithinZone**;
would replace all types of z1's waypoints with custom points, using 40% the custom points of each type detected within z1, and capping each type of replacement at a maximum of 10 points.

_result = [z1, [], [], [], [1, 10000], [1, 10000]] call **DAC_fApplyPointsWithinZone**;
would replace Air and Camp types of z1's waypoints with custom points, using 100% the custom points of Air and Camp detected within z1, capping each at a maximum of 10000 points (effectively specifying no cap limit).

Notes:

See the included demo mission "DAC_quick_custom_points_2" for a dynamic real-world example of this custom point system.