Documentation

Dynamic Artillery Pieces (DAP)

Last update: v1.5.1

What is this

DAP is an *Arma 3* script that allows the Mission Editor (you) to create real (or virtual) artillery/mortar fire-missions faster and smarter for one or multiple sides, using *Eden* marker's positions and an external fire-missions list where you plan the caliber, ammo type, rounds, cycle of repetition and more. *DAP* doesn't change any original *Arma* AI behavior, saving server performance, and preserving the *Arma* integrity and compatibility with any mod.

Creation concept: make use of artillery-pieces practical, fast, and scalable for multiplayer or single-player missions.

Why use *DAP* to manage artillery:
Unlike other scripts, *DAP* does not require the editor to specify artillery-pieces for each fire-mission. As soon as a fire-mission is triggered, *DAP* will search for artillery-pieces available at that time that fit the requested caliber, the requested ammo-type, and have range to the target. This means that for the same target, *DAP* can use different pieces on the map; just as the same artillery-piece can be used for different fire-missions with different targets.

What you set for each fire-mission with *DAP*:

1. Real or virtual fire-mission;
2. The side that owns each fire-mission;
3. Potential target sectors (*Eden* markers);
4. How many artillery-pieces you want in each fire-mission;
5. What caliber these pieces will be (Light, Medium, Heavy, Super Heavy or Combined);
6. Ammunition type (HE, Cluster, Smoke, Flare etc);
7. Control the volume of rounds per piece;
8. How many repetition cycles does a fire-mission have;
9. Trigger method to release each fire-mission (trigger activation, timer, kill/destruction);

What you set globally with *DAP*:

- Custom callsign for each artillery side;
- Which piece calibers can use CommandChat to report (On/Off);
- Infinite ammunition (On/Off);
- Prevent artillery-pieces self-propelled to change position (On/Off);
- Prevent artillery-pieces from starting match with no magazines (On/Off);

- Fire-mission areas visible on the player map (On/Off) WIP
- Custom cooldown by caliber among cycles of fire-mission repetition;
- Pre-defined whitelist of pieces working (Arma, DLCs, CDLCs, RHS, CUP, etc);
- Pre-defined whitelist of ammunition working (Arma, DLCs, CDLCs, RHS, CUP, etc);
- Pre-defined blacklist of currently bugged vehicles/static turrets;
- Pre-defined blacklist of currently bugged ammunition;
- Debug mode with simple or deep detailing;
- Etc…

Automatically *DAP Library* supports content from:

- *Arma 3*;
- Expansion *Apex*;
- DLC *Tanks*;
- DLC *Contact*;
- CDLC *Western Sahara*;
- CDLC *Reaction Forces*;
- CDLC *Expeditionary Forces*;
- CDLC *Global Mobilization*;
- Mod *RHS*;
- Mod *CUP*.

How *DAP* works technically: (before installation for advanced editors' valuation)

Phase 1/3: Identification:

As soon as the match is loaded on the server, *DAP* identifies all artillery-pieces (vehicles and/or static turrets like mortars) that you have named with "dap" or "DAP" in their variable-names. Within a hundredth of a second, *DAP* searches for all target-markers you have also named with "dap" or "DAP". Once this initial process is complete and the mission starts, *DAP* reads each fire-mission present as planned for those sides using the script and gives a unique "City" codename for each one. After getting to know a fire-mission, *Phase 2* starts individually for each one.

Phase 2/3: Waiting activation:

DAP waits to act again only when a trigger activates a fire-mission. During this waiting moment, *DAP* simultaneously checks all fire-mission triggers every 10 seconds. If one of them is activated, *Phase 3* starts. If no fire-missions are available anymore, *DAP* automatically turns off itself successfully while the match keeps going.

Phase 3/3: Filtering and action:

As soon as a fire-mission is released by trigger, target killed/destroyed, or timer, *DAP* reads the requirements of the released fire-mission, initially understanding whether it's a fire-mission with real artillery-pieces or a virtual one (where the projectiles just fall from the sky, without a shooter somewhere on the map). Some process behind the scene:

If Virtual Artillery: After some time since the fire-mission was released, the shells start to come in the chosen sectorized-target. If no repetition cycle is set, the fire-mission ends successfully. *DAP* ends for this fire-mission.

If Real Artillery: in the hundredth of a second after the fire-mission is released, *DAP* starts looking for candidates to perform the fire-mission, checking these in order of priority:

1. Identifies which registered pieces (classname of a vehicle, for example) in *DAP Library* correspond to the requested caliber;
2. Identifies which registered magazines (classname of them) in the *DAP Library* correspond to the requested ammo-type;

3. *DAP* refreshes the SIDE pieces database, removing all those destroyed or with no gunner anymore;
4. Filters by only SIDE pieces that are operational with some ammunition;
5. Filters by the requested caliber;
6. Filters by the requested ammo-type;
7. Filters by pieces with operational range (not too close, not too far) from the target;
8. Finally, the fire mission team is assembled in a few seconds, defining their fire-mission team leader and executing the shelling.

If no fire-mission repetition cycle is set, the fire-mission ends successfully. If no more fire-missions in the waiting phase, *DAP* ends successfully still the mission keeps running.

For multiplayer and single-player

DAP works for all game purposes, including multiplayer and single-player.

For Hosted and Dedicated servers

DAP was built for both server types.

Compatible with RHS, CUP, or any others

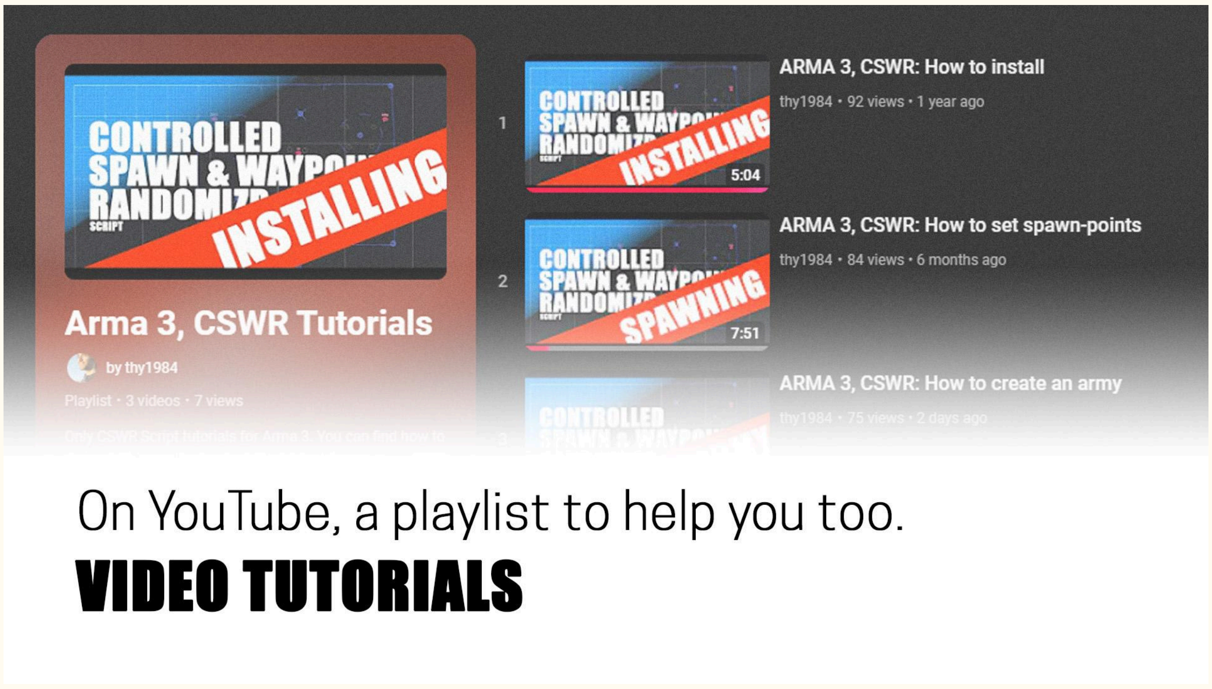
Yes! Also, you can add even more content to the *DAP Library*.

Debug monitor and feedback available

DAP was built to make the Mission Editor life easier, so if you want to get to know the debug monitor now, check this out.

Video Tutorials

Soon. But subscribe to my channel to be notified on the day I release the DAP video tutorials: https://www.youtube.com/@thy1984


CONTROLLED SPAWN & WAYPO RANDOMIZ SCRIPT INSTALLING
Arma 3, CSWR Tutorials
by thy1984
Playlist • 3 videos • 7 views
1
ARMA 3, CSWR: How to install
thy1984 • 92 views • 1 year ago
5:04
2
ARMA 3, CSWR: How to set spawn-points
thy1984 • 84 views • 6 months ago
SPAWNING
7:51
ARMA 3, CSWR: How to create an army
thy1984 • 75 views • 2 days ago
On YouTube, a playlist to help you too.
VIDEO TUTORIALS

If you need an SQF editor

Sure, I’m using *Visual Studio Code* with this customs specific for *Arma 3*:
https://forums.bohemia.net/forums/topic/239960-vs-code-tutorial-how-to-config-vs-code-for-arma-3-2023/

If you need something simpler:
https://notepad-plus-plus.org/, install it and, when you open some script file, go to Notepad++ main menu, “Language” and select “C” as file language. That’s it.

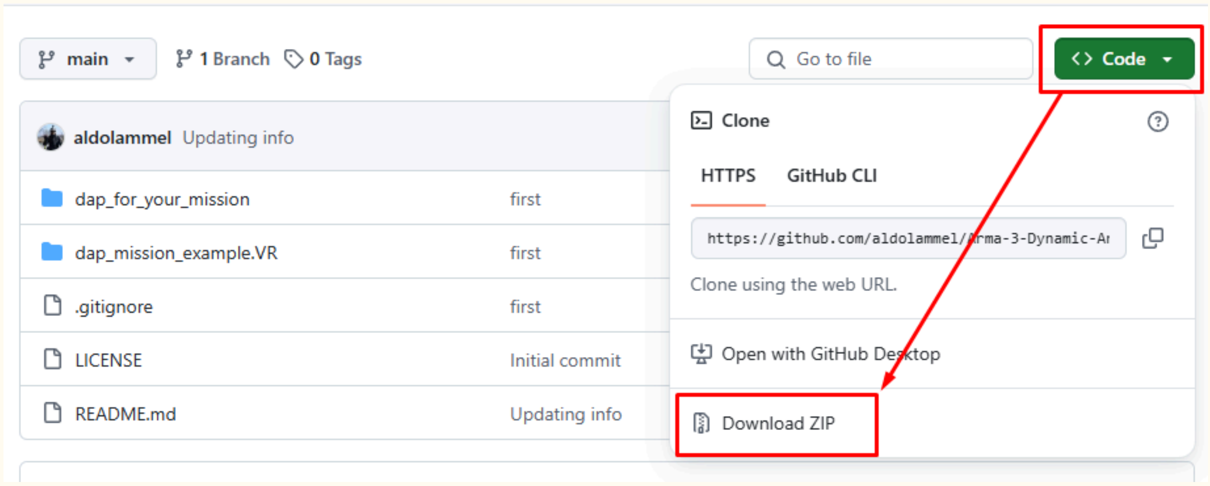
Run the script for a first look

1. Go to https://steamcommunity.com/sharedfiles/filedetails/?id=3371824030
2. Subscribe and wait for Steam to download it;
3. Open *Arma 3*, go to Multiplayer > Server browser > Host server > click Host Server button;
4. Select "Virtual Reality" map and, after that, select "*DAP* (...)";
5. Play.

Important: if you want to apply this script in your missions, check this out.

Install the script in my mission ★★★

1. Go to: https://github.com/aldolammel/Arma-3-Dynamic-Artillery-Pieces-Script
2. Download the zip and open it;



3. In zip, get in the "dap_for_your_mission" folder;
4. Copy all "dap_for_your_mission" content to your mission folder root: DRIVE:\Users\yourName\Documents\Arma 3\yourProfile\missions\yourMission\
5. WARNING: in the mission folder root, if you already have a "description.ext" file, don't replace the current file with the new one. Just add the code below in the current file:

```
class cfgFunctions
{
		// DAP: DYNAMIC ARTILLERY PIECES
		#include "DynamicArtilleryPieces\THY_DAP_functions.hpp"
};
```

6. WARNING: still in the mission folder, if you already have an "init.sqf" file, don't replace the current file with the new one. Just add the code below in the current file:

```
// DAP > HIDE THE SCRIPT MARKERS:
if ( !DAP_isOn || !DAP_debug_isOn ) then {{private _mkr = toUpper _x; private
_mkrChecking = _mkr splitString DAP_spacer; if (_mkrChecking find DAP_prefix
isNotEqualTo -1) then {_x setMarkerAlpha 0}} forEach allMapMarkers};
```

7. WARNING: finally, in the mission folder, if you already have an “initPlayerLocal.sqf” file, don’t replace the current file with the new one. Just add the code below in the current file:

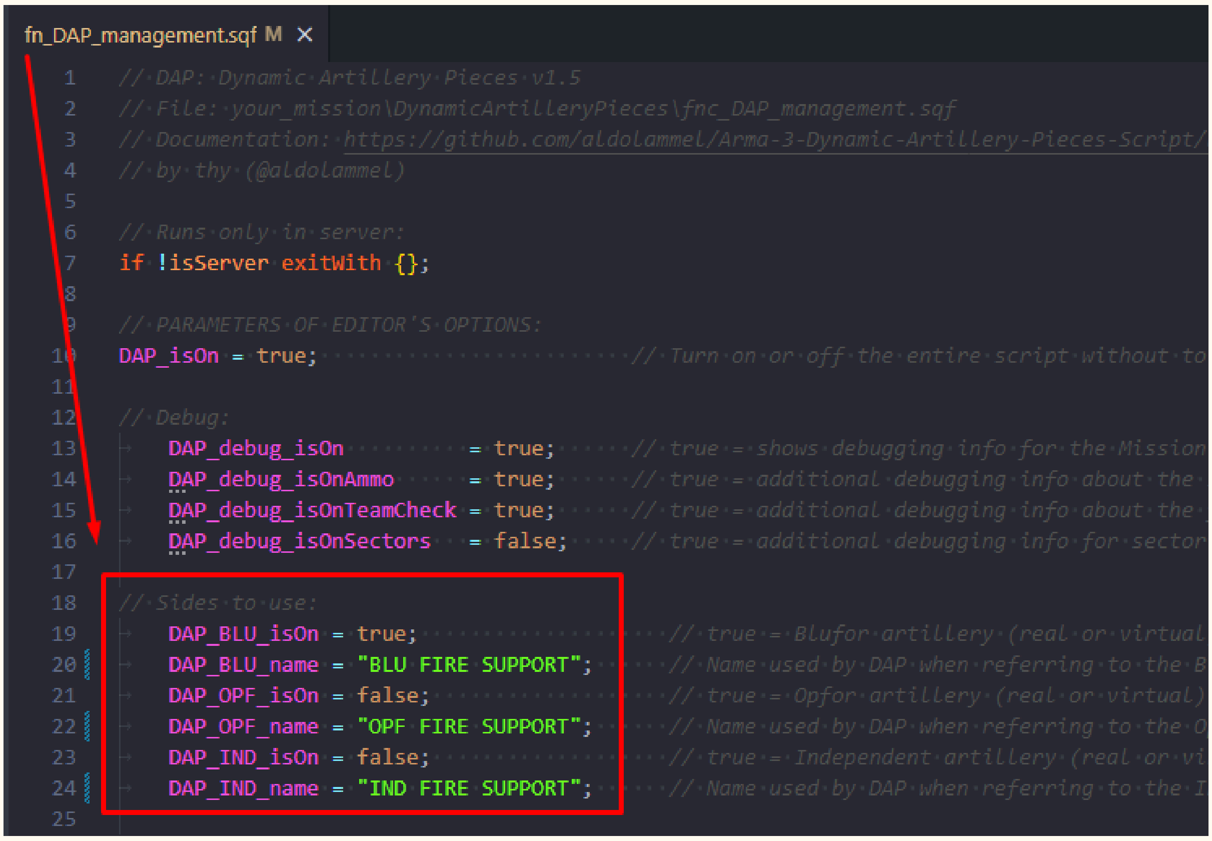
```
// DAP: DYNAMIC ARTILLERY PIECES
[player] execVM "DynamicArtilleryPieces\fn_DAP_playerLocal.sqf";
```

8. Now, let’s tell DAP what you want from it!

Choose which side(s) will use DAP

In *Arma 3* there are 4 sides available: Blufor, Opfor, Independent, and Civilian. Civilian is not an option for obvious reasons. That said, open the *DAP Management*: *\DynamicArtilleryPieces\fn_DAP_management.sqf*

Set as *true* the side you want to build fire missions:


fn_DAP_management.sqf M ×

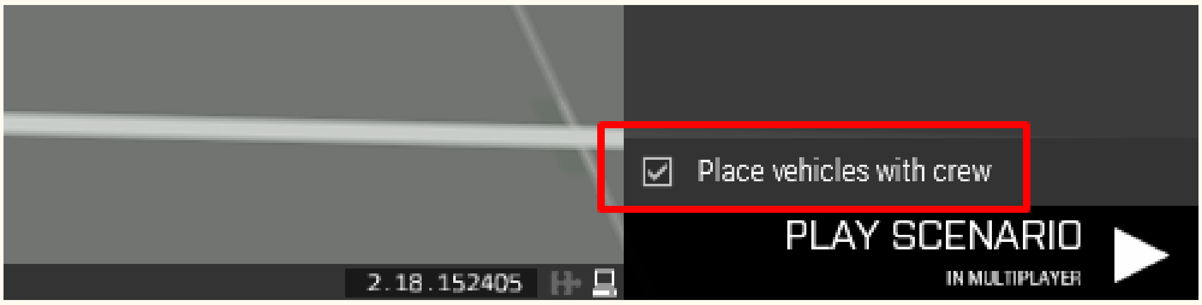
```
// DAP: Dynamic Artillery Pieces v1.5
// File: your_mission\DynamicArtilleryPieces\fnc_DAP_management.sqf
// Documentation: https://github.com/aldolammel/Arma-3-Dynamic-Artillery-Pieces-Script/
// by thy (@aldolammel)

// Runs only in server:
if !isServer exitWith {};

// PARAMETERS OF EDITOR'S OPTIONS:
DAP_isOn = true;                         // Turn on or off the entire script without to

// Debug:
    DAP_debug_isOn          = true;     // true = shows debugging info for the Mission
    DAP_debug_isOnAmmo      = true;     // true = additional debugging info about the
    DAP_debug_isOnTeamCheck = true;     // true = additional debugging info about the
    DAP_debug_isOnSectors   = false;    // true = additional debugging info for sector

// Sides to use:
    DAP_BLU_isOn = true;                     // true = Blufor artillery (real or virtual
    DAP_BLU_name = "BLU FIRE SUPPORT";       // Name used by DAP when referring to the B
    DAP_OPF_isOn = false;                    // true = Opfor artillery (real or virtual)
    DAP_OPF_name = "OPF FIRE SUPPORT";       // Name used by DAP when referring to the O
    DAP_IND_isOn = false;                    // true = Independent artillery (real or vi
    DAP_IND_name = "IND FIRE SUPPORT";       // Name used by DAP when referring to the I
```

Select all artillery pieces you want

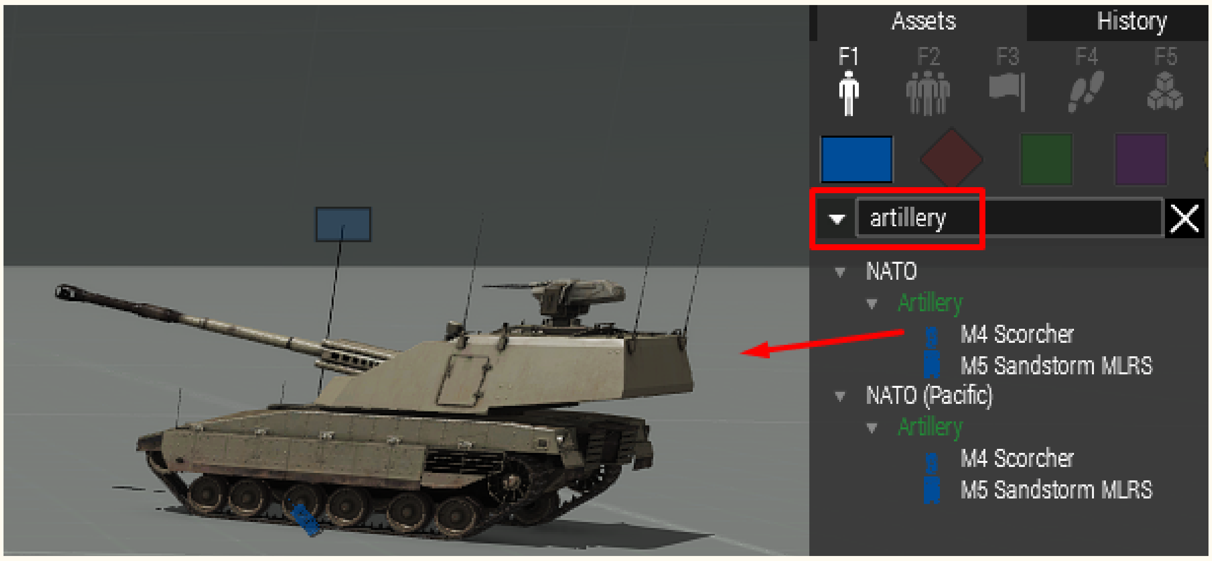
Using real artillery

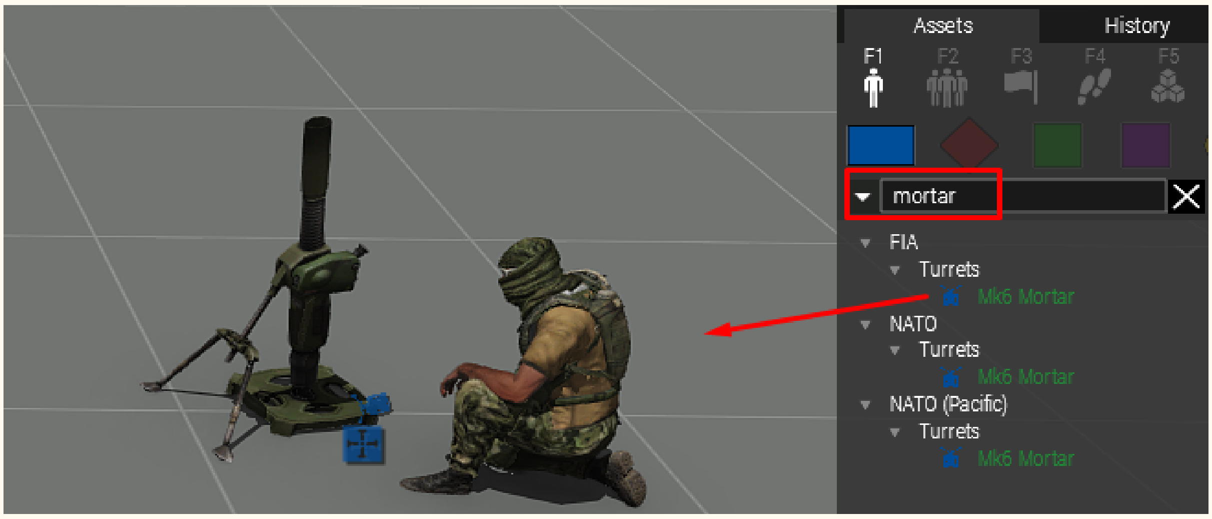
The next step is to set in your mission those vehicles and static/turret weapons (both are considered here “artillery pieces” or just “pieces”) with artillery or mortar capacities. But let’s start with just one to make the process easier.

Press “F” on Eden and keep the “Place vehicles with crew” checked:



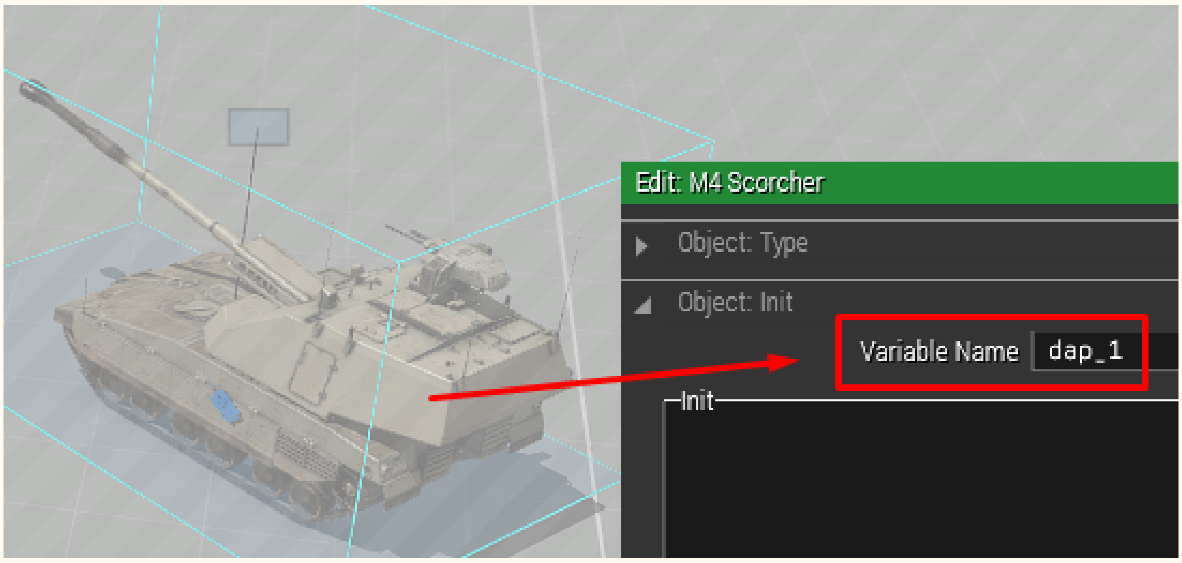
DAP doesn’t work with empty pieces.





Open the piece attributes and add this name or whatever you wish since it brings "dap_" in its variable name, for example:

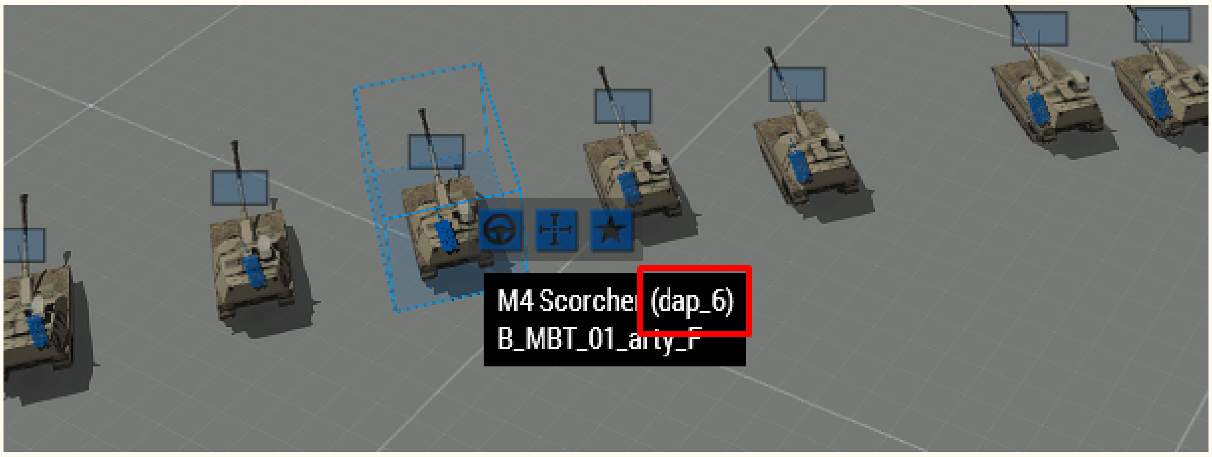
- dap_1 (Recommended)
- an_dap_example
- an_example_dap



Variable Name dap_1

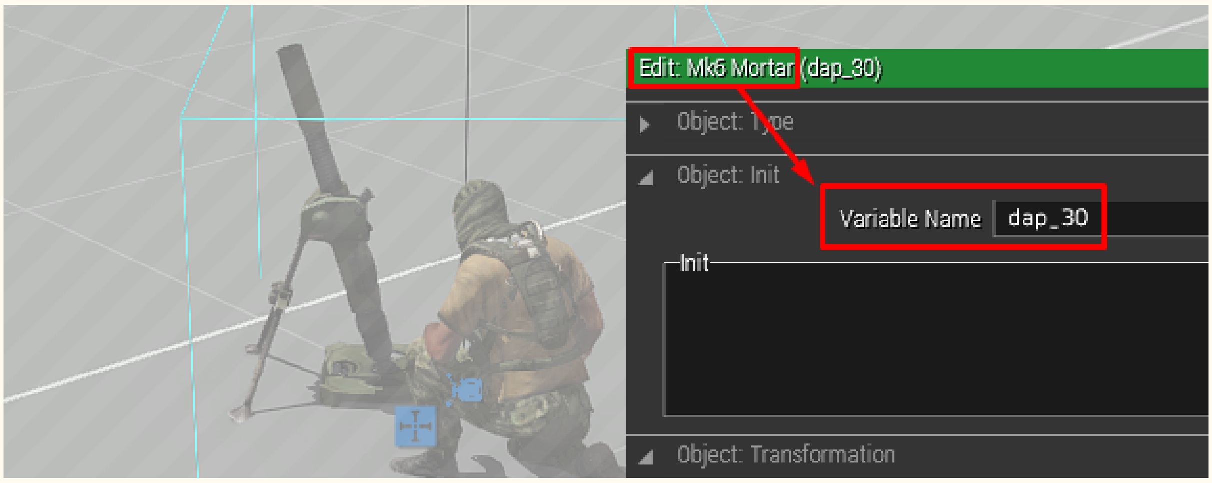
Notice the variable "dap_" MUST be set in the piece, NOT in crew members. *Eden* will create some for them.

Once you named one piece, just *Copy and Paste* how much you want because *Eden* will follow the *DAP* naming logic, adding a different number for each new copied piece (below).



M4 Scorcher (dap_6)
B_MBT_01_arty_F

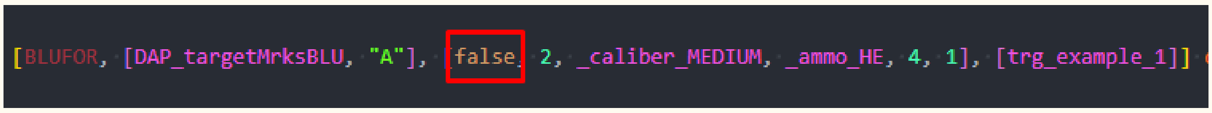
For another type of piece, again, create the first one and name it using a higher number to avoid duplicate (don’t worry, *Eden* tells you if it happens):



Using virtual artillery

Sometimes, as an editor, I wanna be straight to the point, marking where and when I want some *booms* and that’s it. Even though *DAP* is a light artillery script, you might be prioritizing the server/mission performance and, for this case, virtual artillery reduces two-thirds of the *DAP's* actions. So you don’t need to drop any new asset in your mission, just set each side fire-mission as *false*:

\DynamicArtilleryPieces\fn_DAP_fireMissions.sqf

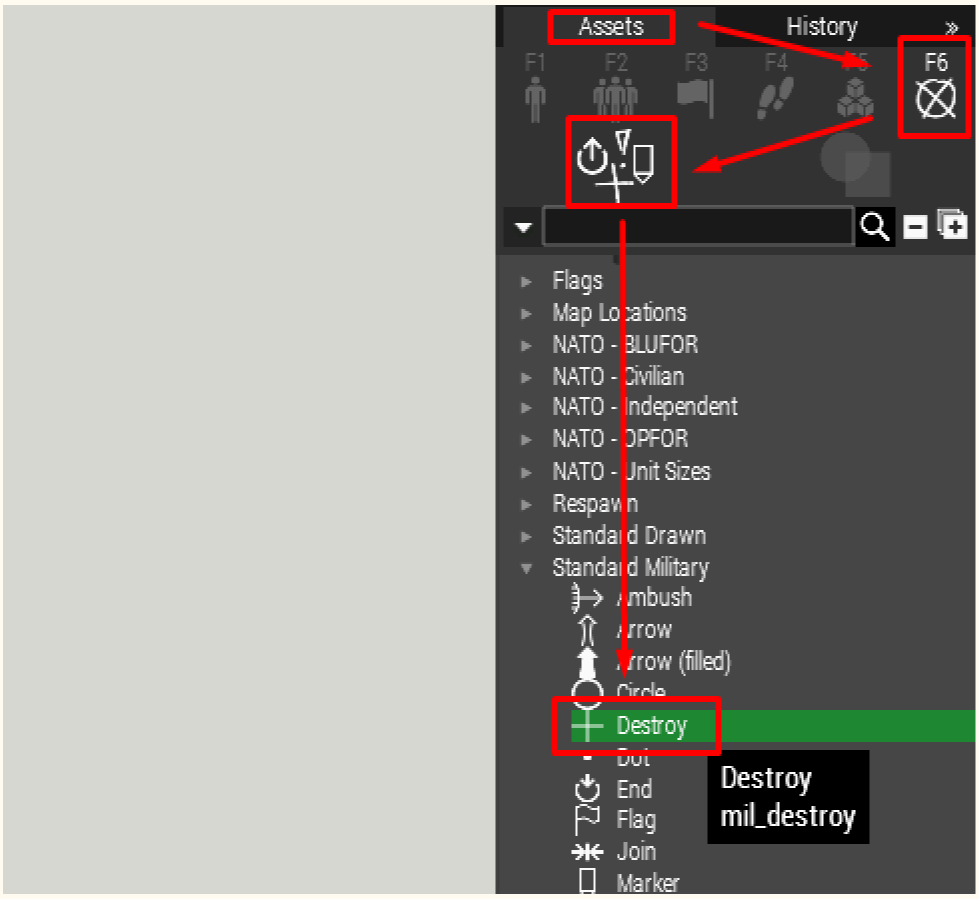


```
[BLUFOR, [DAP_targetMrksBLU, "A"], false, 2, _caliber_MEDIUM, _ammo_HE, 4, 1], [trg_example_1]]
```

Awesome, you got your fire support team, virtual or not. So, let’s set some targets now.

Defining targets

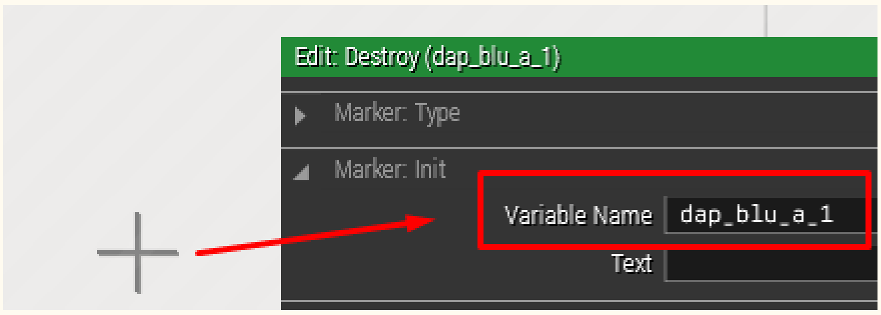
Targets are *Eden* markers you will *Drag and Drop* in your mission. You will always use any "*Destroy*" markers for *DAP*, never another marker type.



Above, the Asset-Browser on *Eden Editor*.

Unlike piece naming, targets need to receive the side tag and a sector letter. Sounds weird but it's easy. Check these examples:

- dap_blu_a_1
- dap_blu_a_20
- dap_blu_z_1
- dap_blu_z_999


Edit: Destroy (dap_blu_a_1)

Marker: Type

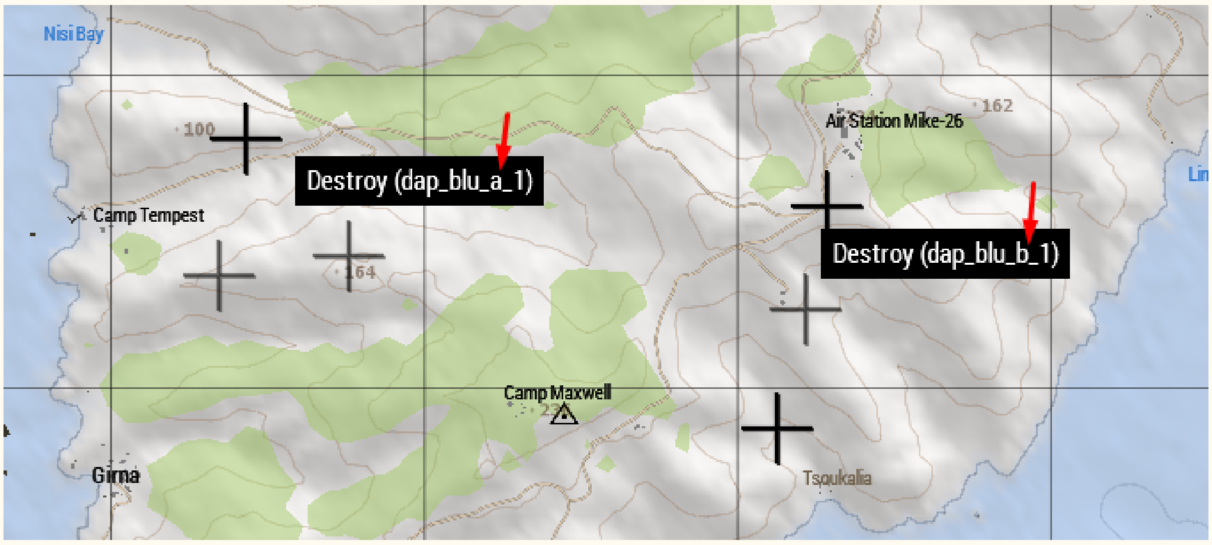
Marker: Init

Variable Name dap_blu_a_1

Text

Above, editing a target-marker attributes on *Eden*. The name is telling DAP this target belongs to the BluFor fire teams and they calling this target sector "A".

Now, *Copy and Paste* how many targets "A" you want to give for BluFor attacks. After that, if you want, create some targets for sector "B" and so on, if needed.



Important: if you set a fire mission to hammer sector "A", *DAP* will look for artillery pieces you placed on the mission that better fit that request. Once *DAP* has the team to execute the fire mission, *DAP* will select randomly just one of those target markers of the sector "A". Use how many target markers you want. It doesn't affect any server performance.

Perfect! Before we jump into the coolest part, you need to set a trigger for each fire mission you have in mind.

Defining triggers

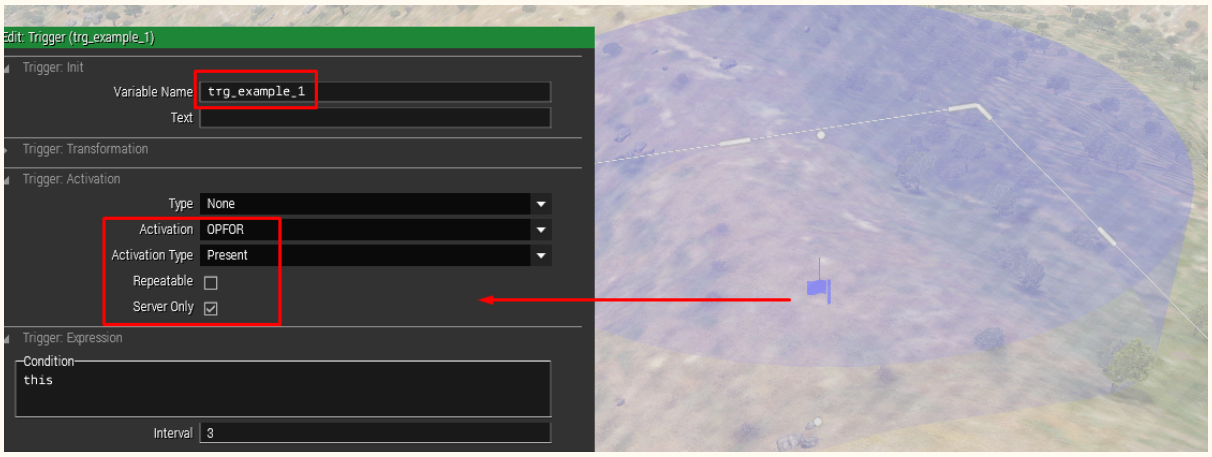
A fire mission doesn't take place out of the blue. You always need to tell *DAP* (NO EXCEPTION) what should initiate bombs being dropped from the sky.

There are 3 methods:

- by *Eden* trigger;
- by killing or destroying a target (unit, vehicle, building);
- by timer;
- and also you can combine them.

Fire mission released by Trigger

As any other trigger. You define what conditions will activate it. You only need to set a varname on it to use later.


Edit: Trigger (trg_example_1)

Trigger: Init

Variable Name: trg_example_1

Text:

Trigger: Transformation

Trigger: Activation

Type: None

Activation: OPFOR

Activation Type: Present

Repeatable: ☐

Server Only: ✓

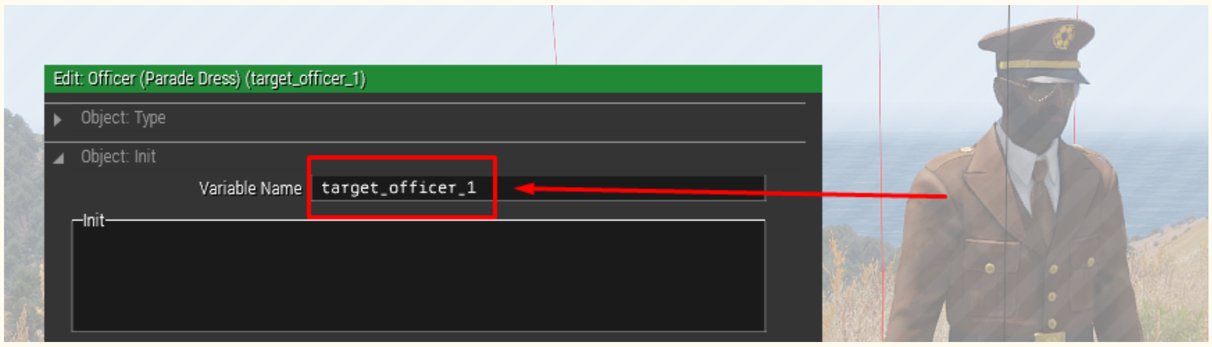
Trigger: Expression

Condition:
```
this
```

Interval: 3

Fire mission released by Target

You could start a fire mission if a unit dies or, for example, a vehicle blows up. Even a breakable structure collapse would release a fire mission with this method.


Edit: Officer (Parade Dress) (target_officer_1)

Object: Type

Object: Init

Variable Name: target_officer_1

Init

Fire mission released by Timer

This method you do directly in the fire missions file you will see next.

Combining all of them

And the combination you do also directly in the fire missions file.

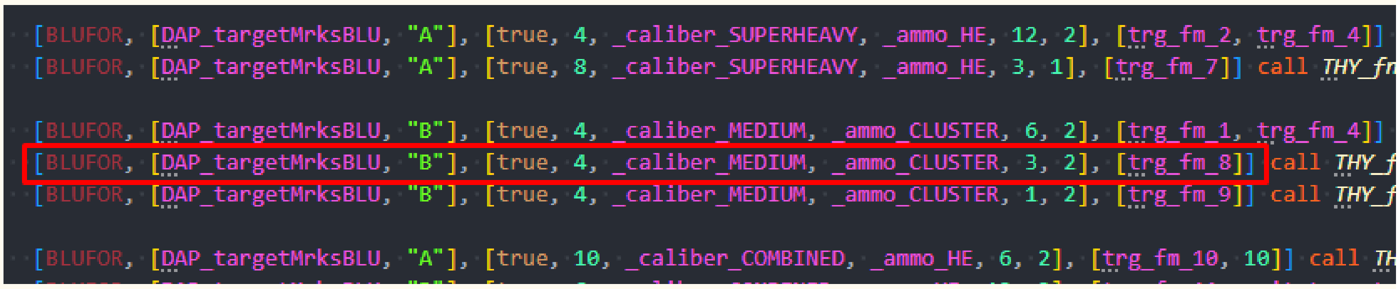
Okay, so let’s do it!

Defining fire-missions

And then the fire-missions, finally. Open your Fire-Missions file:

\DynamicArtilleryPieces\fn_DAP_fireMissions.sqf

This row is literally a fire mission. You can create fire missions as much as you want or the server supports with no performance drops.



```
[BLUFOR, [DAP_targetMrksBLU, "A"], [true, 4, _caliber_SUPERHEAVY, _ammo_HE, 12, 2], [trg_fm_2, trg_fm_4]]
[BLUFOR, [DAP_targetMrksBLU, "A"], [true, 8, _caliber_SUPERHEAVY, _ammo_HE, 3, 1], [trg_fm_7]] call THY_fn

[BLUFOR, [DAP_targetMrksBLU, "B"], [true, 4, _caliber_MEDIUM, _ammo_CLUSTER, 6, 2], [trg_fm_1, trg_fm_4]]
[BLUFOR, [DAP_targetMrksBLU, "B"], [true, 4, _caliber_MEDIUM, _ammo_CLUSTER, 3, 2], [trg_fm_8]] call THY_f
[BLUFOR, [DAP_targetMrksBLU, "B"], [true, 4, _caliber_MEDIUM, _ammo_CLUSTER, 1, 2], [trg_fm_9]] call THY_f

[BLUFOR, [DAP_targetMrksBLU, "A"], [true, 10, _caliber_COMBINED, _ammo_HE, 6, 2], [trg_fm_10, 10]] call TH
```

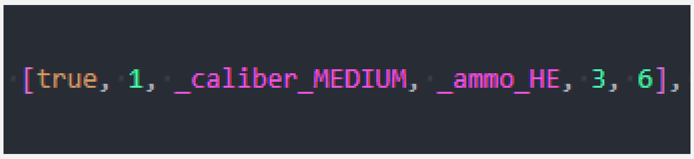
Let's understand what each column of a fire mission row:

Side	Target markers and sector	Is real artillery, Num of pieces involved, caliber of them, ammo type, rounds per piece, and repetition cycle	Triggers
That side owns it.	Each side has just one target marker label, e.g. "*DAP_targetMkrsBLU*". Target segmentation is done through the use of a letter which is called a 'sector letter'. It means this fire mission will only consider targets from sector "A".	**Is real artillery:** "true" means that the fire-mission uses real artillery-pieces, subject to the dangers of the battlefield. If "false" it will be a virtual shelling. **Number of pieces involved:** You can request more than one artillery piece, but if you don't have enough in-game, *DAP* will use just those available that fit all fire mission requirements. **Caliber:** In *DAP* the pieces are chosen by their calibers. If you ask for heavy caliber, *DAP* will investigate what howitzers, MRLs, and/or mortars fit with heavy calibers and are/still available in the mission. **Ammo type:** If you request High Explosive ammo, *DAP* will filter the options, looking for just those pieces with this ammunition type. **Rounds per piece:** You got it. **Repetition cycle:** Hope you got it too.	Ways to release a fire mission. You must use at least one method for each fire mission: **Trigger:** A Eden-trigger with its configs that you are used to do on *Eden Editor* or via script. **Target:** An object like a vehicle, unit or building. **Timer:** A cooldown (in minutes) for the fire-mission starts without triggers or targets. **If you combine methods**, to release the fire-mission just one of them needs to be reached, not all.

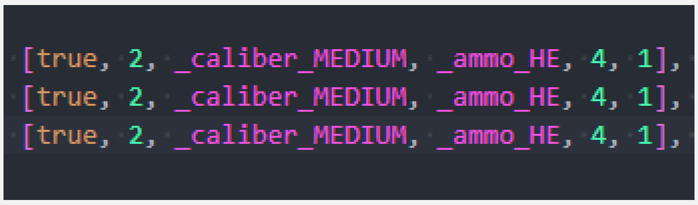
Strategies for your fire-missions

Here *DAP* suggests some strategies to reach your creativity. And don't forget to check the pocket guides to make everything more visual for you and make your fire-missions planning faster.

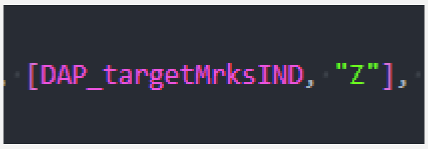
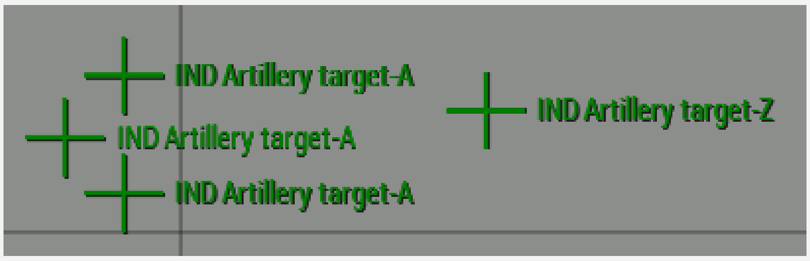
Shelling slowly a target / keeping hit for a long time

How	in fn_DAP_fireMissions.sqf file, example:
Just set a small amount of rounds and increase the number of cycles. The target will be hit with the intensity you requested but slowly, with a caliber cooldown among the cycles. Remember, you can see (and custom) the rearm cooldown by caliber through *fn_DAP_management.sqf* file.	 `[true, 1, _caliber_MEDIUM, _ammo_HE, 3, 6],`

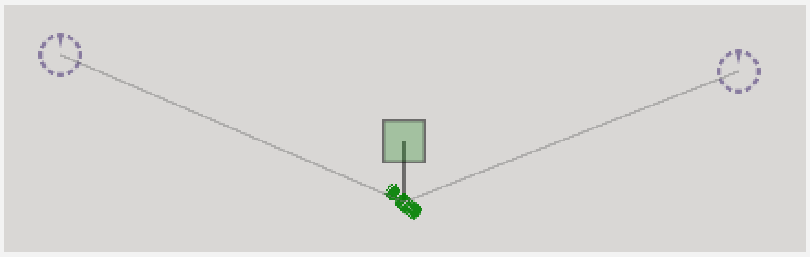
Shelling more areas with the same number of pieces

How	in fn_DAP_fireMissions.sqf file, example:
Instead of setting one fire-mission using 6 artillery-pieces, split this one into other 4 smaller fire-missions, each one using only 2 artillery-pieces. It will make *DAP* select more targets based on those you dropped through the map.	 `[true, 2, _caliber_MEDIUM, _ammo_HE, 4, 1],` `[true, 2, _caliber_MEDIUM, _ammo_HE, 4, 1],` `[true, 2, _caliber_MEDIUM, _ammo_HE, 4, 1],`

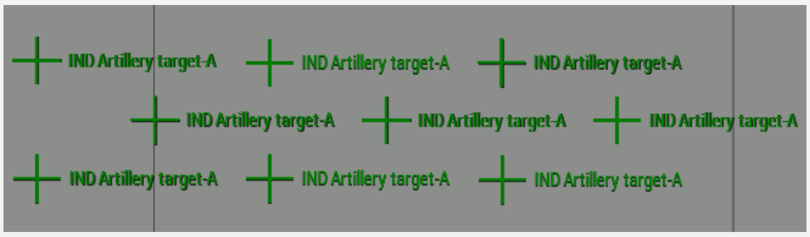
Be sure the target will be shelled, no matter what

How	in fn_DAP_fireMissions.sqf file, example:
As *DAP* works with sectorized targets, just set in one or more fire-missions a target that the sector has just one target to be considered.	 `[DAP_targetMrksIND, "Z"],`
	in Eden Editor, example:
While fire-missions focused in sector "A" have 3 options to choose from, sector "Z" has only 1 target. It means fire-missions designated to hit sector "Z" will hit for sure that position. In the right, a target-marker using the variable-name *dap_ind_z_1*	 IND Artillery target-A IND Artillery target-A IND Artillery target-A IND Artillery target-Z

Make artillery positions unpredictable

How	in fn_DAP_fireMissions.sqf file, example:
	None.
	in Eden Editor, example:
DAP is friendly to original *Arma* stuff. Use the *Eden* solution where you drop some “empty” markers and connect the artillery (only through 2D view) on them. Right-click over the piece and then “Connect” and “Set Random Start”. In the right, an example of 3 positions where that MRL can spawn.	

Make target marker positions unpredictable

How	in fn_DAP_fireMissions.sqf file, example:
	None.
Use how many target-markers your goals ask. Markers have zero impact in server performance once each one is just a position (x,y,0) and they are visible only if the *Debug Mode* is On.	 IND Artillery target-A IND Artillery target-A IND Artillery target-A IND Artillery target-A IND Artillery target-A IND Artillery target-A IND Artillery target-A IND Artillery target-A IND Artillery target-A

Pocket guide: Artillery Piece Calibers

Basically, the artillery is composed of 3 categories of pieces: Mortars, Howitzers, and MRLs (Multiple Rocket Launchers). Despite the mortars being those with lower calibers, sometimes you can find a mortar allowed to execute fire missions that ask for *_caliber_MEDIUM*.
That said, let me present something easy to remember:

- (generally, lighter calibers) Mortars
- (generally, average calibers) Howitzers
- (generally, heavier calibers) MRLs

Calibers Use this in fn_DAP_fireMissions.sqf	**Description**	
_caliber_LIGHT	Only caliber less than 123mm, regardless if it belongs to Howitzer, MRL, or mortar.	
_caliber_MEDIUM	Only caliber between 123mm and 159mm, regardless if it belongs to a Howitzer, MRL, or mortar.	
_caliber_HEAVY	Only caliber between 160mm and 299mm, regardless if it belongs to Howitzer, MRL, or mortar.	

_caliber_SUPERHEAVY	Only caliber equal or greater than 300mm, regardless if it belongs to Howitzer, MRL, or mortar.	
_caliber_COMBINED	Random calibers are combined if different pieces are available. Important: if you're using this, I do recommend using *_ammo_HE* once all calibers of artillery-pieces can use that ammo-type.	

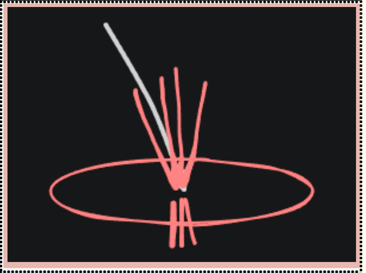
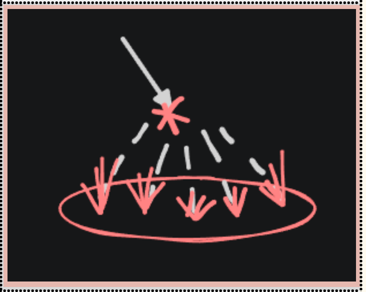
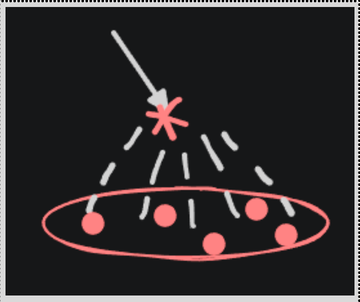
Important 1: *Arma 3* and its DLCs don't have howitzers and mortars able to perform for *_caliber_HEAVY* or *_caliber_SUPERHEAVY* fire-missions, even *RHS* and *CUP* don't have functional examples. In this case, you can drop in your mission MRL.

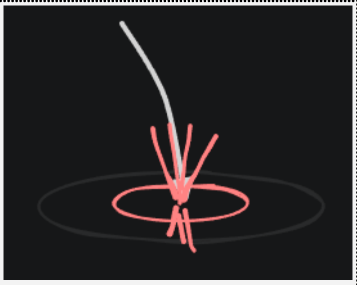
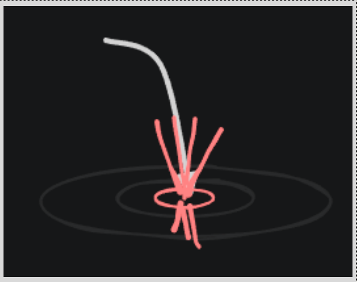
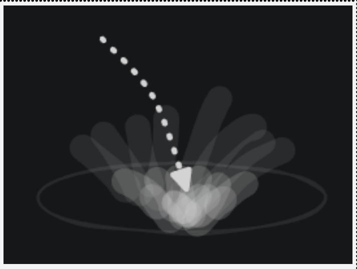
Important 2: there's no point in asking for a *_caliber_SUPERHEAVY* fire-mission, but only providing the *DAP* with 82mm mortars. The fire-mission will never happen.

What about we check the ammunition-types available?

Pocket guide: Magazines (Ammo)

Right after *DAP* gets to know what caliber the Editor wants to use (in other words, how destructive the fire mission needs to be), the script will try to connect that caliber with the ammunition type chosen by the editor too. *DAP* doesn't add any magazines to the game. The script just understands those the server/mission has loaded and checks if the pieces available can fire them in that range.

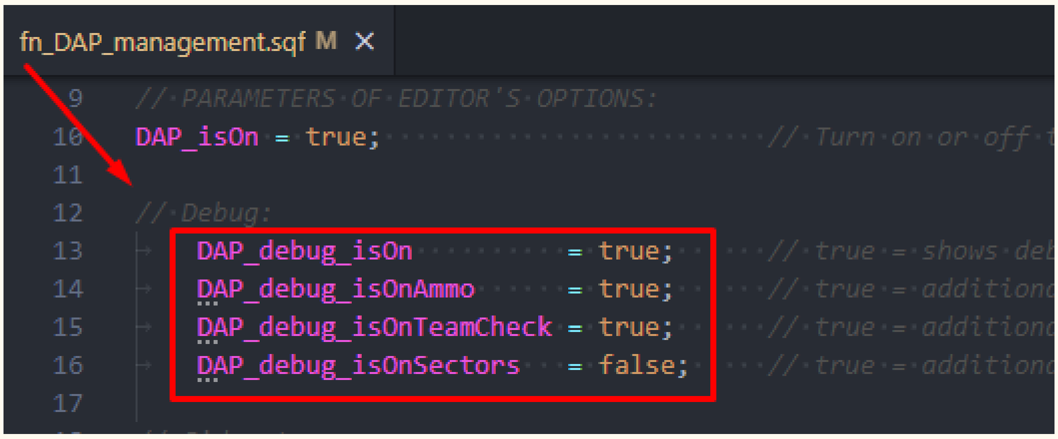
Ammo magazines Use this in fn_DAP_fireMissions.sqf	**Description**	
_ammo_HE	High Explosive ammo is a great choice against infantry, buildings, and light-medium armor vehicles. *DAP recommends*	
_ammo_CLUSTER	Cluster ammo is the greatest choice against infantry in trenches and forests, spreading fragmentation grenades on a large area. *DAP recommends*	
_ammo_CLUSTER_MINE_AP	Cluster dropping Anti-personnel-mines.	
_ammo_CLUSTER_MINE_AT	Cluster dropping Anti-tank-mines.	

_ammo_GUIDED	Commonly a HE ammunition with superior accuracy.	
_ammo_GUIDED_LASER	Commonly HE ammunition with maximum accuracy.	
_ammo_SMOKE	Smoke ammo recommended choice to preserve structures but blind the enemy for a while. *DAP recommends*	
_ammo_FLARE	Flare ammo recommended choice to draw attention to a specific area or temporarily light up the dark. *DAP recommends*	

Debug Mode: investigating what is happening

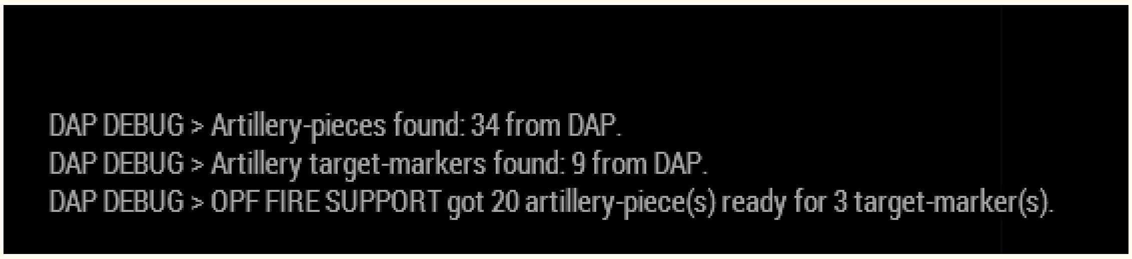
First of all, make sure you are NOT running another script debug together because it could break the data presentation on your screen. Work with only one script debug mode ‘true’ at a time.

That said, turn *true* all debug layers you need for your checking/investigations.



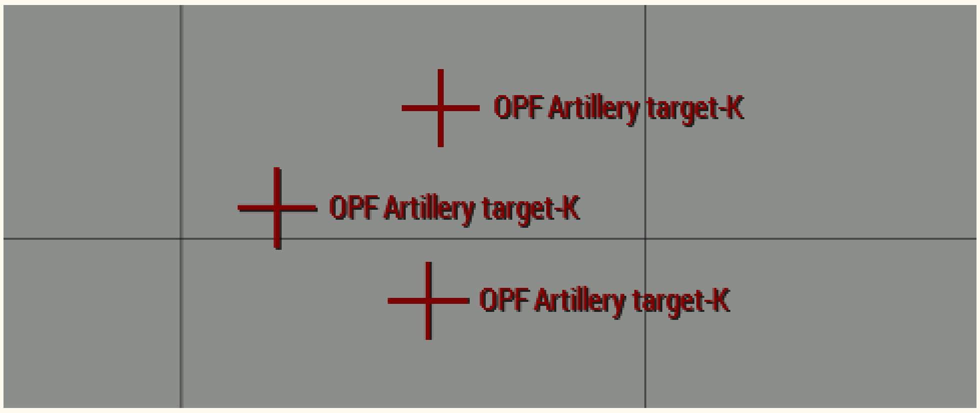
```
fn_DAP_management.sqf M ×
// PARAMETERS OF EDITOR'S OPTIONS:
DAP_isOn = true;                         // Turn on or off t

// Debug:
    DAP_debug_isOn            = true;     // true = shows deb
    DAP_debug_isOnAmmo        = true;     // true = additiona
    DAP_debug_isOnTeamCheck = true;       // true = additiona
    DAP_debug_isOnSectors     = false;    // true = additiona
```

DAP will tell you what is happening most of the time during your mission creation time.



```
DAP DEBUG > Artillery-pieces found: 34 from DAP.
DAP DEBUG > Artillery target-markers found: 9 from DAP.
DAP DEBUG > OPF FIRE SUPPORT got 20 artillery-piece(s) ready for 3 target-marker(s).
```

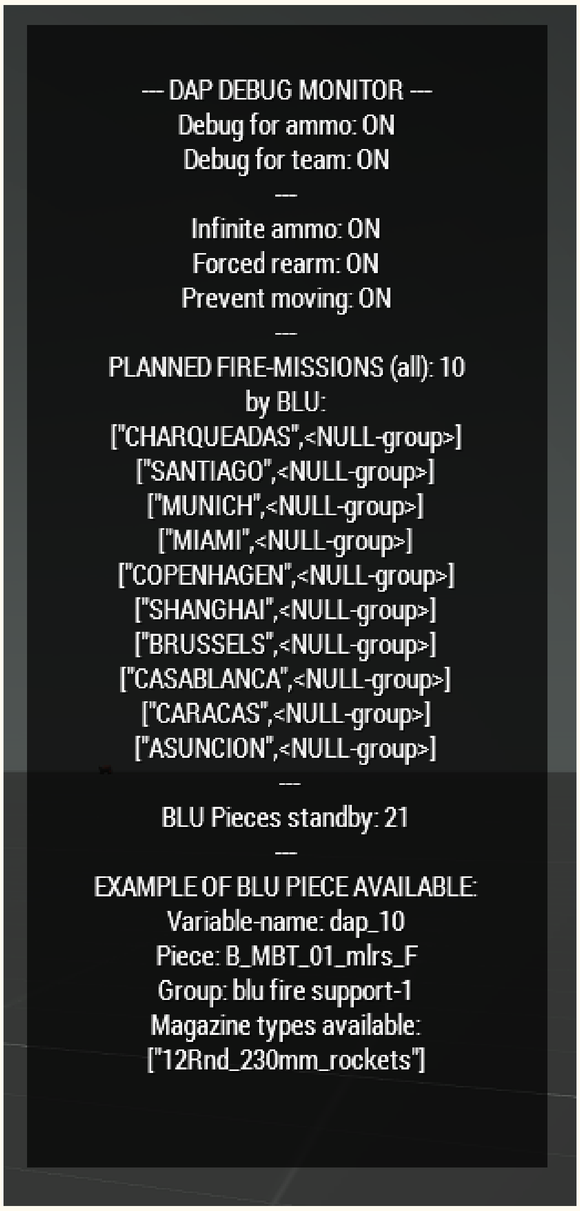
For each new match, *DAP starts* presenting a brief of what was found to be correctly used.



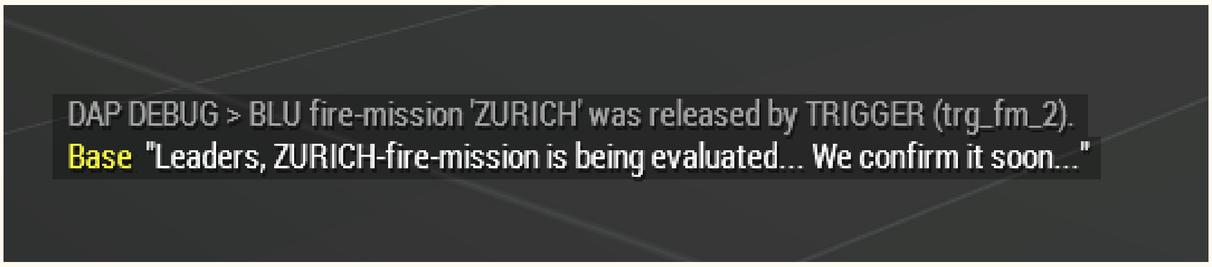
Debug mode also turns visible the target-markers of all enabled *DAP* sides in-game.

With debug mode as *true*, 100% of the time you should see this window continuously in your upper-right corner.

It's updated every 10 secs.

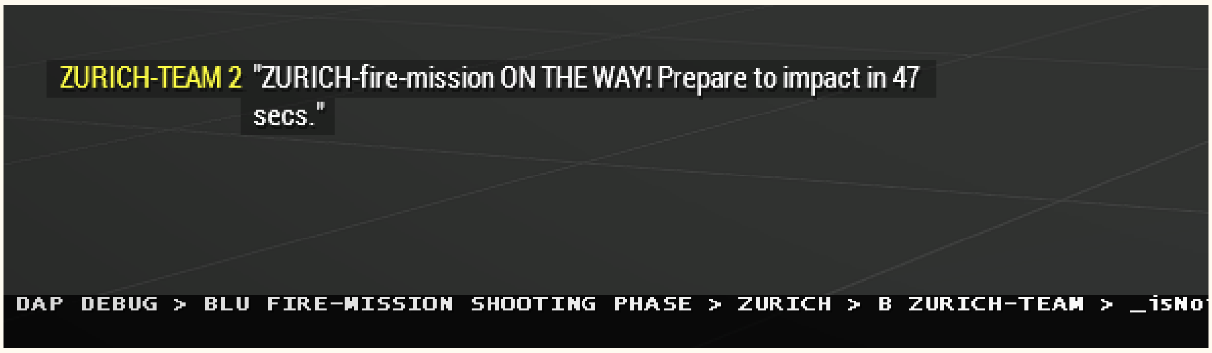


```
--- DAP DEBUG MONITOR ---
Debug for ammo: ON
Debug for team: ON
---
Infinite ammo: ON
Forced rearm: ON
Prevent moving: ON
---
PLANNED FIRE-MISSIONS (all): 10
by BLU:
["CHARQUEADAS",<NULL-group>]
["SANTIAGO",<NULL-group>]
["MUNICH",<NULL-group>]
["MIAMI",<NULL-group>]
["COPENHAGEN",<NULL-group>]
["SHANGHAI",<NULL-group>]
["BRUSSELS",<NULL-group>]
["CASABLANCA",<NULL-group>]
["CARACAS",<NULL-group>]
["ASUNCION",<NULL-group>]
---
BLU Pieces standby: 21
---
EXAMPLE OF BLU PIECE AVAILABLE:
Variable-name: dap_10
Piece: B_MBT_01_mlrs_F
Group: blu fire support-1
Magazine types available:
["12Rnd_230mm_rockets"]
```



```
DAP DEBUG > BLU fire-mission 'ZURICH' was released by TRIGGER (trg_fm_2).
Base "Leaders, ZURICH-fire-mission is being evaluated... We confirm it soon..."
```

If debug is turned on, you will see messages like this, bringing more information about what is happening behind the scene.



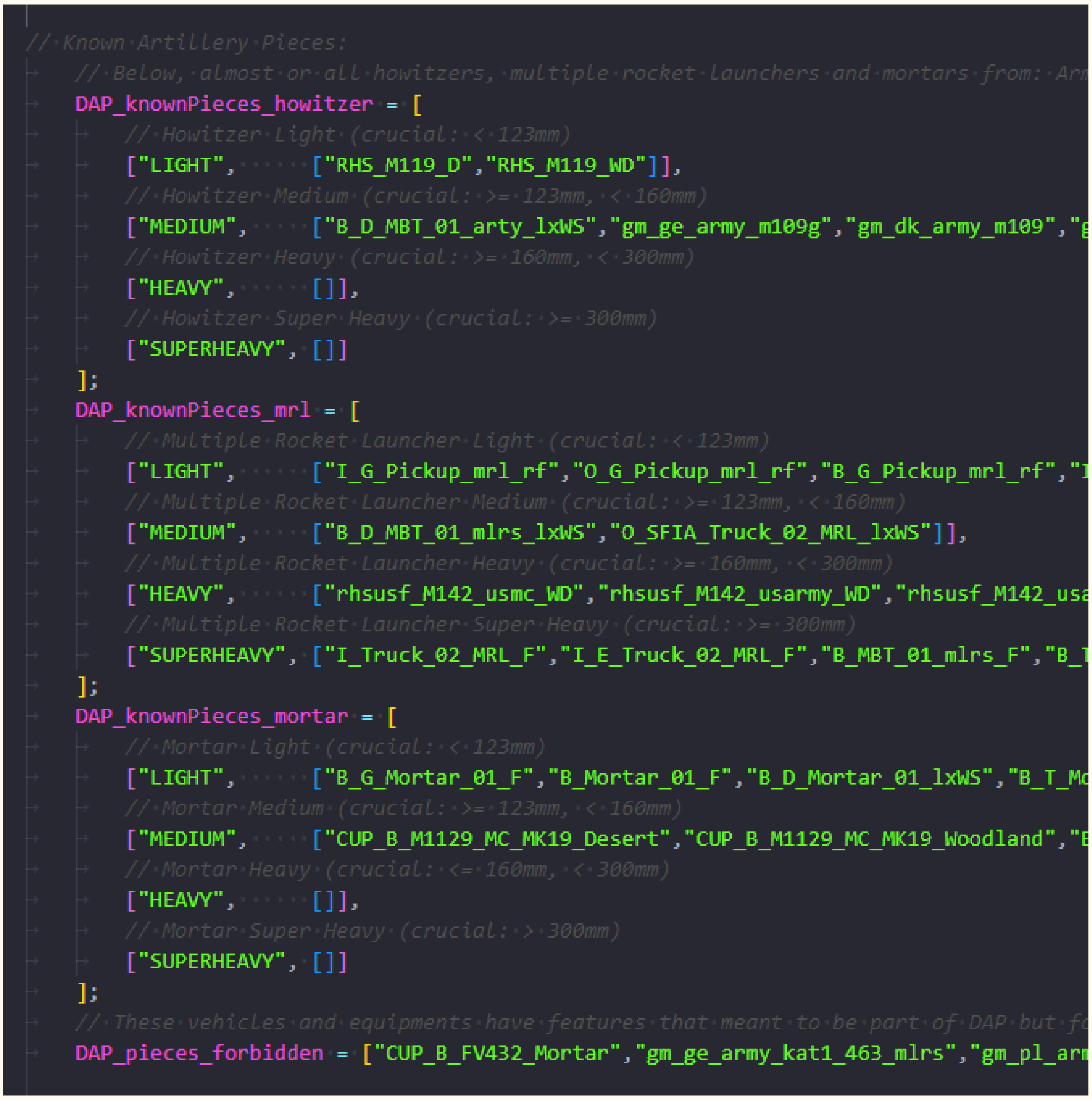
ZURICH-TEAM 2 "ZURICH-fire-mission ON THE WAY! Prepare to impact in 47 secs."

```
DAP DEBUG > BLU FIRE-MISSION SHOOTING PHASE > ZURICH > B ZURICH-TEAM > _isNo
```

Critical: turn to *false* the debug mode to before you publish your mission. To keep debug mode not *false* will increase significantly the *DAP* time processing in all phases of script.

Fixing: a piece or ammunition is not recognized by DAP

In the *fn_DAP_management.sqf* file there is an extensive list of artillery pieces and magazines already tested with DAP and working, including some DLCs and mods. If you are facing issues with a magazine or piece, turn on the *DAP Debug Mode* so that the script will tell what's happening. If DAP asking you to register an artillery piece or a magazine, here (below) is the place to.



```
// Known Artillery Pieces:
    // Below, almost or all howitzers, multiple rocket launchers and mortars from: Arm
    DAP_knownPieces_howitzer = [
        // Howitzer Light (crucial: < 123mm)
        ["LIGHT",      ["RHS_M119_D","RHS_M119_WD"]],
        // Howitzer Medium (crucial: >= 123mm, < 160mm)
        ["MEDIUM",     ["B_D_MBT_01_arty_lxWS","gm_ge_army_m109g","gm_dk_army_m109","g
        // Howitzer Heavy (crucial: >= 160mm, < 300mm)
        ["HEAVY",      []],
        // Howitzer Super Heavy (crucial: >= 300mm)
        ["SUPERHEAVY", []]
    ];
    DAP_knownPieces_mrl = [
        // Multiple Rocket Launcher Light (crucial: < 123mm)
        ["LIGHT",      ["I_G_Pickup_mrl_rf","O_G_Pickup_mrl_rf","B_G_Pickup_mrl_rf","I
        // Multiple Rocket Launcher Medium (crucial: >= 123mm, < 160mm)
        ["MEDIUM",     ["B_D_MBT_01_mlrs_lxWS","O_SFIA_Truck_02_MRL_lxWS"]],
        // Multiple Rocket Launcher Heavy (crucial: >= 160mm, < 300mm)
        ["HEAVY",      ["rhsusf_M142_usmc_WD","rhsusf_M142_usarmy_WD","rhsusf_M142_usa
        // Multiple Rocket Launcher Super Heavy (crucial: >= 300mm)
        ["SUPERHEAVY", ["I_Truck_02_MRL_F","I_E_Truck_02_MRL_F","B_MBT_01_mlrs_F","B_T
    ];
    DAP_knownPieces_mortar = [
        // Mortar Light (crucial: < 123mm)
        ["LIGHT",      ["B_G_Mortar_01_F","B_Mortar_01_F","B_D_Mortar_01_lxWS","B_T_Mo
        // Mortar Medium (crucial: >= 123mm, < 160mm)
        ["MEDIUM",     ["CUP_B_M1129_MC_MK19_Desert","CUP_B_M1129_MC_MK19_Woodland","B
        // Mortar Heavy (crucial: <= 160mm, < 300mm)
        ["HEAVY",      []],
        // Mortar Super Heavy (crucial: > 300mm)
        ["SUPERHEAVY", []]
    ];
    // These vehicles and equipments have features that meant to be part of DAP but fo
    DAP_pieces_forbidden = ["CUP_B_FV432_Mortar","gm_ge_army_kat1_463_mlrs","gm_pl_arm
```

Always use classnames, checking first if the classname is not already in here.

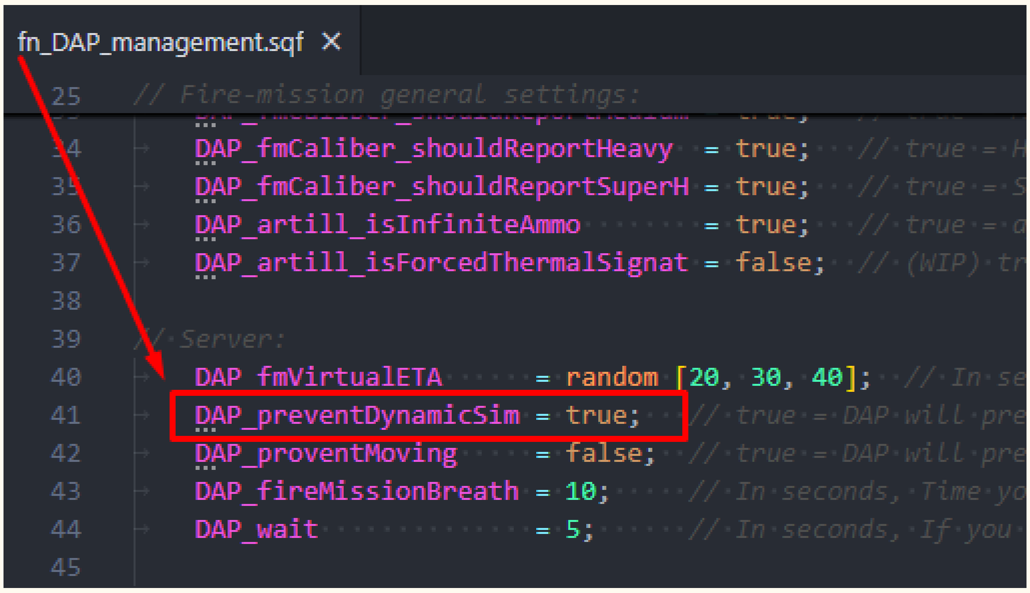
How to know a vehicle or equipment classname:

On Eden, right-click over the object > Log > "Log classes to clipboard".

Now, just paste it! ;)

Fixing: some artillery pieces are frozen

Make sure you have this option as true in *fn_DAP_management.sqf*.



fn_DAP_management.sqf

```
// Fire-mission general settings:
    DAP_fmCaliber_shouldReportHeavy  = true;   // true = H
    DAP_fmCaliber_shouldReportSuperH = true;   // true = S
    DAP_artill_isInfiniteAmmo        = true;   // true = a
    DAP_artill_isForcedThermalSignat = false;  // (WIP) tr

// Server:
    DAP_fmVirtualETA      = random [20, 30, 40];  // In se
    DAP_preventDynamicSim = true;   // true = DAP will pre
    DAP_proventMoving     = false;  // true = DAP will pre
    DAP_fireMissionBreath = 10;     // In seconds, Time yo
    DAP_wait              = 5;      // In seconds, If you
```

If the problem persists, it’s because probably the piece has some ammunition issue or, in case the vehicle has no mobile turret, the vehicle is not aligned with the selected target (I’m working on it to find a fair solution).

Contribute to the *DAP* script

Discussion on Bohemia Forums

https://forums.bohemia.net/forums/topic/290962-release-dynamic-artillery-pieces-dap/

Changelog on GitHub

https://github.com/aldolammel/Arma-3-Dynamic-Artillery-Pieces-Script?tab=readme-ov-file#changelog

Author

Based in Porto Alegre, Brazil

thy @aldolammel

If you care

Just give a like for *DAP* on Workshop to spread the word :)

https://steamcommunity.com/sharedfiles/filedetails/?id=3371824030