DASH_Library COMMAND REFERENCE

DESCRIPTION

DASH_library, version 2.1 – addon-level pack of 350 functions for single-player mode; it consists of 2 files: **DASH_library.pbo** (main part) and **DASH_library_Settings.pbo** (optional file with start settings).
All functions, settings and most of the data are stored in the **stringtable.csv**.

There are 20 categories in test missions:

- **First steps** – the most important system functions;
- **Arrays** – generic array processing methods;
- **Convertion** – make one thing from another;
- **Custom EH** – event handler-like scripts;
- **Environmen** – weather, landscape params and so on;
- **EH** – functions to use with Event Handlers;
- **Geometry** – angles, distances etc;
- **Groups and units** – ranks, call signs...;
- **GUI** – methods to work with GUI controls;
- **Indexes** – tricks with array indexes ;
- **Inventory** – weapons and ammo;
- **Machine Learning** – Machine Learning methods;
- **Markers** – map markers;
- **Math** – math fns and matrix operations;
- **Random** – random number generation and random element selection;
- **Search** – find something good in array;
- **Sort** – changing an order of array's elements;
- **System** – deals with variables and user-defined properties;
- **Text** – strings and string arrays;
- **Vehicles** – vehicles and crews

PURPOSE

Extending possibilities of scripters and mission makers,
saving their time during difficult scripts development and to support my other OFP projects with different code

WHAT'S NEW (2.1 vs 2.0)

- Nearly 100% OFP 1.96 compatibility (CWA still required for **overrideMoves**);
- Settings for library launch;
- 2 powerful data storage/access techniques;
- 36 fns added

INSTALLATION

OFP 1.96 required, CWA 1.99 recommended.
Put the files **DASH_library.pbo** and **DASH_library_Settings.pbo** to **Addons** folder in the game root or to the same folder of any mod folder you like.

Drop test missions here:
…\Users\YourProfile\missions and put test campaign to **Campaigns** folder in game root

If you want to use Fwatch-dependent functions – take the files from Fwatch 1.14 folder of this package and put 'em to the game root. Use exactly these executables – there were some conflicts 'tween Fwatch on/off detection and huge DASH_Library stringtables with official 1.13/1.14 versions. Thanks to Faguss for the hotfix!

To make shortcuts for the mods you need, do this:
...\fwatchCWA.exe" **-nosplash -nomap -mod=@ECP;@EKLMN;@EPRST**
or ...\fwatch.exe" **-nosplash -nomap -mod=@ECP;@EKLMN;@EPRST** for old good OFP.
BTW, official Fwatch site: **http://ofp-faguss.com/fwatch/**

LAUNCH

Add this code to init.sqs or to player's init field:

```
call localize "dashlib"
```

See the next chapter for more information

DESCRIPTION OF FUNCTIONS

First steps

call localize dashlib – DASH_Library initialization. Must be called at first.
Creates:
global resource array **DASH**;
global request array **DASH_EH_Quey** to handle «Custom Event Handlers»;
sideUnknown constant of «side» type;
global functions: defaults are **alias**, **loadFns**, **setProjecTag**, **print** and **test**

It's convinient to run it from init.sqs or from player's init field.
If the line **call localize "dashlib"** will be added to "Init" EH in some unit's config file, it doesn't matter, how many units of this type will be placed on map - **dashlib** can be run only once. And if you have no DASH_Library.pbo, this code will just do nothing without error messages.

From version 2.1 on you can change some launch settings.
There is **dahlib.ini** file in the **DASH_library_Settings.pbo** for this purpose.
For frequent experiments with launch settings one can change the path, defined in the **DASH_library_Settings\stringtable.csv**, for example, to use game root as an options storage:

```
Language, English
...
"dashlib_settings_path", "..\dashlib.ini"
```

and to place **dashlib.ini** to this place. Settings will work without need of PBOing, it saves much time.

If a file **DASH_library_Settings.pbo** has no changes or just absent then defaults used:

```
_weather = true;
_globalFns = ["alias", "loadfns", "setprojecttag", "print", "test"];
_afterLaunch = {};
```

Parameter description.
On the default launch weather settings will be redefined and native commands **setOvercast** and **setFog** won't work. To switch off weather recording script, one need to open **dashlib.ini** and set **_weather=false**.
In this case native fog/overcast commands works as usual, but some functions (**fog**, **overcast**, **setOvercast2**, **setFog2**) don't. See **Environment** section for more details.

To change the set of global (i.e. loaded on start) functions, use **_globalFns** option, all function names must be in lover case.
To run some code (sqs or sqf) after library launch – use **_afterLaunch** setting.

function_name_in_lower_case call loadFns
alternative syntax:
array_of_fns_names call loadFns – global function to load another fns from stringtable.csv, returns 2-element array:
0 - "OK", if all functions were loaded, error report otherwise;
1 – array of loaded fns names.
Shy result - one or more functions from stringtable.csv becomes global.
With **loadFns** you can easily upload all other functions without adding any garbage to the global variable scope.
Example:

```
["debug", "hintdebug"] call loadFns
```

It will load **debug** and **hintDebug**, functions for SQF/SQS debugging

correctVarName call setProjectTag – global function, sets the global prefix to load functions with **loadFns**. String **correctVarName** will be added to the start of every function you load with **loadFns**. Returns true, if argument was a string

```
"FN_" call setProjectTag
["splitbyrule", "inverse", "allvariants"] call loadFns
```

shy result will be **FN_splitByRule**, **FN_inverse** and **FN_allVariants**

[name_in_dashlib, name_u_prefer] call alias – loads another function from stringtable.csv under **name_u_prefer**. An alternative to prefix usage – if you have some global vars in your project with the same names like some of the DASH_library functions – you can use these fns under aliases. Returns true, if function **name_in_dashlib** was found

There is one more method of function usage – direct call with localization.
In the **stringtable.csv** all the functions but **dashlib** are stored with names like **f_nameinlowercase**. Exactly this way is used for inner needs of DASH_library:

```
"Language", "English"
...
"f_iscwa", "!({getWorld} call localize {f_isnil})"
```

How to run:

```
call localize "f_iscwa"
[Min, Max] call localize "f_rndfloat"
NumericArray call localize "f_sum"
```

These are the actions you won't saw in the most of the test mission examples, I hide 'em to protect your mind from seeing almost the same things 250+ times. 2 times will be enough – here and in test mission *«Examples (part 1).Intro»*,
see «First steps» category
Back to subject list

Arrays

Array call allSubsets – all subsets of **Array** elements not including **Array** itself.
Length limitation for **Array**: <=13

ArrayOfArrays call allVariants – all combinations of subarray elements: for
`[[true, false], [1, 2, 3], ["a", "b"]] call allVariants`
it will return 12 subarrays from [true, 1, "a"] to [false, 3, "b"]

Array call array2Pairs – convert array with even length like
[name1, v1, name2, v2, name3, v3] to «key-value» pairs
[[name1, v1], [name2, v2],[name3, v3]].
Works good with **addWeaponCargo**/**addMagazineCargo**.
You can create functions with variable argument number and ORDER:
`["radius", 2000, "pos", getPos boom, "EMP", true] call nuclearExplosion`
will be the same as
`["EMP", true, "pos", getPos boom, "radius", 2000] call nuclearExplosion`

[Arr1, Arr2] call arrayAddArray – add one array to another element-by-element
Returns an array

[Arr, Nmb] call arrayAddNumber – add number to each number in array.
Returns an array

[Arr1, Arr2, expr] call arrayFncArray – apply **expr** for each pair of correspondent elements of 2 equal-length arrays. Returns an array

[Arr, Var, expr] call arrayFncScalar – apply **expr** for each pair **[Arr(i), Var]**.
Returns an array

NestedArray call arrayMean – array of arithmetic averages of **NestedArray** by each dimension. Returns an array with the same length as (**NestedArray** select 0)

[Arr1, Arr2] call arrayMultArray – multiply one array to another element-by-element.
Returns an array

[Arr1, Arr2] call arraysAreIntersecting
alternative syntax:
[Arr1, Arr2, ignoreCase] call arraysAreIntersecting – true, if arrays have at least 1 common element. If **ignoreCase** = true, then function will ignore string case.
It will make sense if you have array of magazines/weapons of unit and want to check if there are some common elements in this array and array of strings like
`["m16", "m4", "aK47"]`

NestedArray call arraySimplify – decrease nesting depth of array to 1.
Array of arrays will become simple array. In case of simple array it will just copy it

[Arr1, Arr2] call arraysIntersection
alternative syntax:
[Arr1, Arr2, ignoreCase] call arraysIntersection – return elements of **Arr2**, found in **Arr1.**
If **ignoreCase** = true, then function will ignore string case.
This option is usefull if you want to work with class name arrays

[Array1, Array2] call compareArrays – compares 2 arrays element-by-element, returns true, if **Array1 = Array2**. Works with numeric, string, side and object arrays.
Arrays of different length are always not equal

[Array1, Array2] call compareHeaps – compares 2 arrays, not giving the hell to element order. Case sensitive, but can work with boolean and mixed arrays.
Can compare magazines of unit in 2 different time moments

Var call isArray – returns true, if **Var** is array. You can also use not-array check: **_this in [_this]**

[start, end, step] call makeArray
alternative syntax:
[start, end] call makeArray – array of ascending/descending numbers, starting from **start**, not exceeding **end**, with given **step**. Default progression step==1

Array call pop – remove last element from **Array** and return it. **Array** will be resized

[Integer, Any] call populate - make an array of **Integer** copies of element **Any.** If **Any** is an array then array copies will be used (instead of references)

NumericArray call product – product of all array elements

[Array, element] call pushForward – push element to the front of array. Return no result, array will be resized

[Array, element] call pushNew – if an element is not in array – add 'em to the end. If success - returns true

[Array, Old, New] call replace – in **Array** it will replace all instances of **Old** with **New**. Returns new array

[Array, ArrOld, ArrNew] call replaceM – replaces each occurance **ArrOld** elements in **Array** with correspondent element from **ArrNew**. Returns new array

[Array, Condition] call splitByRule - split **Array** into 2 parts according **Condition**. The result is 2-element array, an elemets, satisfying the **Condition** will be in first one

Array call split1212 – split **Array** into 2 parts, the elements with odd indexes will be in the first subarray, an even elements will be in second one.
If an initial array is like **[key1, val1, key2, val2, key3, val3...]**, then after **split1212** keys and values will be in different subarrays

[Array, NestedArray] call splitByArrays - distribute elements of **Array** between subarrays of **NestedArray** (round robin)

NumericArray call sum – sum of array elements

[Array, N] call take - if **N**>0 then take **N** first elements from **Array**, if **N**<0 then take **N** last elements. Returns an array

ArrayOfArrays call transpose – matrix columns become rows, rows becomes columns. All subarrays must have the same length. Example: **[[1, 2, 3], [4, 5, 6]] ==> [[1, 4], [2, 5], [3, 6]]**

Array call typeList – returns 2-element array:
0 - array of unique elements;
1 – count of each element in **Array**
Function is case-sensitive

Array call unique – returns unique (nonrepeating) elements from **Array**.
An **Array** can contain numbers, strings, objects, sides or groups, types can be similar or mixed. Case-sensitive

[Vals, Numbers] call unwrapList – creates simple array, with **Numbers**[0] of **Vals**[0], **Numbers**[1] of **Vals**[1]...An antipode of **typeList** function

[NumArray, NumArray2] call without
[StringArray, StringArray2] call without – an alternative to Array1-Array2.

Example:
```
[["m4", "m4", "m60", "m21"], ["m4", "m21"]] call without
```
returns ["m4", "m60"]

Practice: «Examples (part 1).Intro»
Back to subject list

Convertion

Angle call angle2compass – direction in degrees to compass heading

Angle call angle2hour – direction in degrees to hour angle (1-12)

Array call any2pos
alternative syntax:
SomeObject call any2pos
alternative syntax:
SomeGroup call any2pos
alternative syntax:
MarkerName call any2pos – returns 3D position

[FloatValue, StrOld, StrNew] call convert - converts **FloatValue** from one length units to another. Available units: "**ml**", "**yd**", "**ft**", "**in**", "**cm**", "**m**", "**km**" – miles, yards, feets, inches, centimeters, meters, kilometers

StrQuad call grid2pos – map quad name (string) – to [X, Y] coordinates.
With mods @WGL or @Invasion44 map grid of the most islands becomes 6-digit, **grid2pos** function use correct format automatically

Str call miltime2daytime – string with US army «military time» like "0730" -> to real time in range [0.0000...23.9999]

pos call pos2grid – position **pos** – to map quad name (for example, "Ec58").
With mods @WGL or @Invasion44 map grid of the most islands becomes 6-digit, so something like "114098" will be returned

[dTime, TimeFormat] call time2string
alternative syntax:
[dTime] call time2string - convert **dTime**
(in hours) to string, according **TimeFormat**.
,
Formats available: "**HH**", "**HHMM**", "**HH:MM**", "**HH:MM:SS**", "**HH:MM;SS:MM**".
If **TimeFormat** not defined then used default "**HHMM**"
(«military time» in the Uncle Sam's army)

Practice: «Examples (part 2).Intro»
Back to subject list

Custom Event Handlers (CEH)

[Owner, CEHType, CodeToRun] call AddCustomEH
alternative syntax:
[ArrayOfOwners, CEHType, CodeToRun] call AddCustomEH
alternative syntax:
[Owner, CEHType, CodeToRun, Path2ScriptFolder] call AddCustomEH
alternative syntax:
[ArrayOfOwners, CEHType, CodeToRun, Path2ScriptFolder] call AddCustomEH
– adds one or several owners (units or groups) to monitoring script, which calls **CodeToRun** expression on event **EHType**.

The function is similar toAddEventHandler command

Inner script will be started with the first call of **AddCustomEH** with the type **CEHType** and it will be ended with the last owner removing
In **CodeToRun** is :
_this select 0: EH owner;
_this select 1: old value of monitored parameter;
_this select 2: new value of monitored parameter.
EHs "**weapons**" and "**magazines**" (fires on equipment changing)
returns more values:
first 3 are standard;
_this select 3: added weapons/magazines;
_this select 4: dropped weapons/magazines

To avoid FPS drop on mass CEH adding I recommend to use array of owners as 0^{th} argument instead of using forEach.
2 or more CEHs of the same type when added to one unit will override each over, so only the last will work.

Available CEH types:

"**ammoprimary**" – changed ammo count for primaryWeapon;
"**behaviour**" – behaviour change;
"**rating**" – rating change;
"**canfire**" – ability to fire on/off;
"**canmove**" – ability to move on/off;
"**canstand**" – ability to stay on/off;
"**crew**" – crew changed;
"**crew3**" – actual crew changed (driver, gunner or commander), works even for in-vehicle rearrangements;
"**freefall**" – free fall detection;
"**health**" – health changed (+-);
"**magazines**" – magazines changed (the order is not taken into account);
"**primary**" – primary weapon changed;
"**secondary**" – secondary weapon changed;
"**weapons**" – weapon set changed (something added and/or removed)

Path2ScriptFolder optional param allows the scripter to use his own folder with custom Ehs. All CEHs scripts («\DASH_library\Sqs\EH» folder) have 2 areas marked for easy editing and you better don't touch the rest of code.

```
;Start modify 1 ---------------------

_type="formation!"
_get={formation _this};
_delay=0.367

;End modify -------------------------
```

_type – CEH name, used to add/remove CEH and to set it's request time.
_get – expression, calculated every _delay seconds, the result can be of any type.
The simplest case is when the result is number, string, side, group or object:

```
;Start modify 2 ----------------------

?(_w!=_w2):[_ui, _w, _w2] call (_e select _i)

;End modify --------------------------
```

_ui – event owner (usualy unit or group), _w – old value, _w2 – new value.

Editable areas of "weapons" CEH:

```
;Start modify 1 --------------------

_type="weapons"
_get={weapons _this};
_delay=0.896

;End modify ------------------------

.......

;Start modify 2 --------------------

_w3=_w2-_w
_w4=_w-_w2
?(count(_w3+_w4)!=0):[_ui, _w, _w2, _w3, _w4] call (_e select _i)

;End modify ------------------------
```

[Owner, EHType] call RemoveCustomEH - similar to expression
unit RemoveAllEventHandlers type.
Owner (unit or group) will be «retired» from the **EHType** supporting script.
FPS drop can be caused by mass usage of this function

[EHType, Number] call SetCEHDelay - set time delay for custom EH of type **EHType.**
Every CEH has embedded recommended request delay, which can be decreased for faster reaction or increased to fight with FPS drop

Practice: «Examples (part 3).Intro»
Back to subject list

Environment

Building call countBuildingPos – number of available positions in **Building**

call countStatic - number of static objects on map

call daylight - returns 0, if called at night, 1 – if called at day time and returns 0.5 – if it's twilights. Precizion is 10 seconds. To get relevant string, use something like that:

```
["night", "twilights", "day"] select (2*call daylight)
```

call **fog** - current fog level.
DASH_library without Fwatch sets it to 0,
with Fwatch 1.14 on, fog will be set according weather info from mission.sqm.
Use **setFog2** function instead of **setFog** command!

[Pos, R] call groundSlope
alternative syntax:
[Pos, R, Step] call groundSlope - average slope and direction of maximal slope in circle with center **Pos** and radius **R**, search **Step** (in degrees) is optional, default value is 15

[Position, Radius] call inForest - true, if there is some forest in circle with given **Radius** near given **Position**. Several trees are not considered as a forest

Obj call inTown - returns true, if object is in the town/village.
Sometimes it doesn't work with small villages because there are no enterable buildings there

Pos call inWater - returns true, if the position is under water

call isWinter - returns true, if season is winter.
Precision of the method is about a week. Works on North hemisphere islands only

call mapName – returns mission folder extension – like getWorld command do
in CWA, but it also works for 200 islands in OFP 1.96

Pos call nearestVillage – returns 3-element array:
0 – name of the nearest village/city;
1 – 2D position of it's center;
2 – how far it from given **Pos**
Works on 25 islands present in database

call **objectID** – returns map number of static object, returns 0 for normal vehicles.
Fwatch 1.14+ needed!

call **overcast** – cast level.
Without Fwatch DASH_library sets cast 0.5,
with Fwatch 1.13 weather info from mission.sqm will be taken into account.
Use **setOvercast2** function instead of **setOvercast** command!

[Pos, Ra] call overObstacle
alternative syntax:
[Pos, Ra, ObjectArray] call overObstacle – returns true, if inradius **Ra** from the position **Pos** there is obstacle which makes impossible to land chopper here.
BlackList of harmless obstacles **ObjectArray** can be defined as argument (2)

rain – rain level. DASH_library sets rain level=0, you should use **setRain2** function instead of **setRain** command if you want to change it

Number call setTime – set day time to **Number**

[Sec, Level] call setFog2 – similar to **Sec setFog Level** command,
but it makes possible to check fog level at any time with **fog** function. Time to change is in seconds, fog level can be from 0 (clean air) to 1 (heavy fog)

[Sec, Level] call setOvercast2 – similar to **Sec setOvercast Level** command, but it makes possible to check cast level at any time with **overcast** function.
Time to change is in seconds, cast level can be from 0 (clean sky) to 1 (low dark clouds)

[Sec, Level] call setRain2 – similar to **Sec setRain Level** command,
but it makes possible to check rain level at any time with **rain** function.
Time to change measured in seconds, rain level can be from 0 (no rain) to 1 (heavy shower)

CharVectorVillage call villageInfo – returns 3-element array
[village correct name, misspeling error (0...1), 2-D position] of given village despite of typos in the name. Function implementation is limited with several dozens of islands. Returns [] on empty island

CharVectorVillage call villagePos – returns 2-D position of given village despite of typos in the name. Function implementation is limited with several dozens of islands. Returns [] on empty island

wind – returns 2-element numeric array:
0 – wind direction in degrees;
1 – wind speed in m/s

windVector – returns 2-element numeric array:
0 – wind speed X component (м/s);
1 – wind speed Y component (м/s)

Practice: «Examples (part 2).Intro»
Back to subject list

Event Handlers

[UnitInfo, oldAnim1, newAnim1, oldAnim2, newAnim2...] call overrideMoves -
overrides a list of animations for AI and player. Requires CWA 1.99.
UnitInfo – unit, unit array, group or trigger.
Works good with movement animations, works bad with the rest, in this case it's often fires AFTER animation to be replaced then it must occure INSTEAD of it.

[UnitInfo, oldAmmo1, newAmmo1, oldAmmo2, newAmmo2...] call replaceBullets -
it replaces fired projectiles of one types with other projectile types.
UnitInfo – unit, unit array, group or trigger, empty strings means what projectile will be simply removed without replacement

_this call shotGeometry – recommended to use in «fired» EH to get more information about shot. Returns an array:

0 – fired projectile (object);
1 – it's starting position;
2 - 3D velocity vector (m/s);
4 – absolute speed (m/s);
5 - azimuth (degrees);
6 – elevation angle (degrees);

Practice: «Examples (part 3).Intro»
Back to subject list

Geometry

[pos1, pos2, pos3] call clockwiseOf – returns true, if movement from **pos1** to **pos2** and than from **pos2** to **pos3** is clockwise.
Another application: if the line determined with first 2 points, then resulting «true» means «**pos3** is to the right from the line» and «false» means «**pos3** is to the left»

[pos1, pos2, K] call delimPoint
alternative syntax:
[pos1, pos2] call delimPoint – point between two 2D positions dividing this line segment with ratio **K**, if not defined then **K**=1/1

[Watcher, Target] call dir2obj – direction from object **Watcher** to object **Target**

[Pos1, Pos2] call dist2d – distance between 2 flat (2D) positions

[Pos1, Pos2] call dist3d – distance between 2 3D positions.
It's better if positions are measured At See Level (f.e. with GetPosASL)

[Pos1, Pos2] call dirToPos – direction from one position to another

Obj call dirOfMove – azimuth of object **Veh** movement

Pos call elevation – altitude of position At See Level

[obj1, obj2] call elevationAngle – vertical angle from one object to another. If is **obj1** is higher than **obj2** – the result will be > 0

[Watcher, Target] call facingDiff – difference between direction of **Watcher** and angle of beam **Watcher - Target**

[Shelter, Guard, StepBack] call hiddenPos – find position in **StepBack** meters after object **Shelter**, to hide from **Guard** (it's supposed what **Shelter** is big enough)

[pos, Range, Step] call highestLowest – the highest and lowest Asl positions in quad with center **pos** and size **Range**, step of measurment **Step**

obj call hrzSpeed – horizontal component of object's speed (m/s)

[obj1, obj2] call inFOV
alternative syntax:
[obj1, obj2, FOV] call inFOV – true, if **obj1** is in the Field Of View (FOV) of **obj2**, default **FOV** value is 86 degrees

[Pos1, Pos2] call lineParams – a and b factors of line-representing equation Y=a*X+b, where the positions **Pos1** and **Pos2** lay at

[Unit1, UnitArray] call nearestFromArray
alternative syntax:
[Pos1, UnitArray] call nearestFromArray – nearest **UnitArray** element (object) from **Unit1 / Pos1.**

[obj1, obj2] call relPos2d – position of **obj2** in coordinate system with base **obj1**, Y axis direction == **obj1** direction

[Obj, Height] call setZ – set altitude of object (AGL height), can be < 0

[Pos1, Pos2, Pos3] call triangleArea – signed area of triangle defined with 3 positions. Negative value means *«point order is clockwise»*. Apply **abs** to result if you need only area.

Practice: «Examples (part 2).Intro»
Back to subject list

Groups and units

unitArray call any2Units
alternative syntax:
groupBravo call any2Units
alternative syntax:
trigger1 call any2Units – unit array/group/trigger – to unit array

Unit call boss – unit becomes leader in his group

[ClassName, pos, grp] call createUnit2
alternative syntax:
[ClassName, pos, grp, initString] call createUnit2
alternative syntax:
[ClassName, pos, grp, initString, skill] call createUnit2
alternative syntax:
[ClassName, pos, grp, initString, skill, strRank] call createUnit2 – similar to "createUnit" command, but returns unit added and you can get rank value of that soldier. If squad number==12 then ObjNull will be returned

UnitArray call enableQueryGroups – creates inner array of groups to make **getGroups** function work. Pointless under Fwatch 1.13+

UnitArray call enableQueryRank – enables to query ranks of unit array.

Not required to use with units created with **createUnit2**

UnitArray call filterGroups – get all groups from the unit list

UnitArray call getBoss – returns person with highest rank from UnitArray.
If there are several soldiers of that rank – the most skilled one will be returned

Unit call getGroupID
alternative syntax:
Group call getGroupID – array of 2 elements – callsign and group color of unit or group

Side call getGroups
alternative syntax:
SideArray call getGroups – list of groups of given side(s).
To work without Fwatch you need big trigger of activation type Any - Present and **enableQueryGroups** function.

Group call getGroupSpeed – average speed of group members

Group call getGroupSpread – approximate diameter of group

Unit call getIntSide – returns 0 for east, 1 for west, 2 for resistance, 3 for civilian

Unit call getRank – rank of the unit (string)

Unit call getRankID – numeric value of unit's rank

[unitArray, surName] call getUnitByLastName – returns unit with last name **surName**, if he is present in **unitArray** or objNull otherwise

Grp call groupIsCargo – checks if all units of group **Grp** are mounted on some vehicles

[Unit1, Unit2] call isEnemy – true if **Unit2** is a visible enemy of **Unit1**

[UnitArray, Unit] call joinSilent – join units to the **Unit** without radio messages. To avoid problems with enableRadio you can use function **enableRadio2** instead

Grp call mostInjured – most injured alive unit in **Grp**

sideUnknown – additional side, the side of projectiles, triggers and objNull

unit call squadNumber – return unit's number in squad – this number is present in his radio messages

GrpLeader call squadOrder – units of **GrpLeader** in the ascending order of their numbers. Group leader is always 0-th

[Unit1, Unit2] call swapIdentity – swap 2 personalities (face+voice+glasses) for **Unit1** and **Unit2**, double use restores status-quo.
Works good in PBOed missions and campaigns, with editable missions placed to «Missions» folder. SaveGame command needed in editable missions.

[grp, pos] call teleportGroup
alternative syntax:
[grpLeader, pos] call teleportGroup – whole group will be teleported to the position **pos**. No one will leave his transport, the relative distances will be saved. Leader's position will be **pos**

[UnitArray, Angle] call watchDir – make all units watch in the same direction. If **Angle**==-1, units will watch task will be completed

Practice: «Examples (part 1).Intro», «TestCampaign»
Back to subject list

GUI

[minIndex, maxIndex] call allCtrls – find idc of all visible GUI elements in range from **minIndex** to **maxIndex** inclusive. Returns an array

[idc, StringArray] call fillBox – clears and fills ListBox or ComboBox **idc** with strings from **StringArray**

Practice: «Examples (part 1).Intro»
Back to subject list

Indexes

[Array, Index] call idxDelete – remove element with this index from the array. Returns no result, array will be resized

[Array, IntegerArray] call idxDeleteM – remove elements with indexes **IntegerArray** from the **Array.** Returns no result, array will be resized

NumericArray call idxMax – find the index of maximal array element

NumericArray call idxMin – find the index of minimal array element

[Array, Index] call idxToEnd
alternative syntax:
[Array, Indexes] call idxToEnd - shift elements with given **Indexes** to the end of **Array.** Returns no result, array will be changed

[Array, Index] call idxToStart
alternative syntax:
[Array, Indexes] call idxToStart - shift elements with given **Indexes** to the head of **Array**. Returns no result, array will be changed

[Array, Indexes] call idxTake – returns elements with given **Indexes** from **Array.** Returns new array

Practice: «Examples (part 1).Intro»
Back to subject list

Inventory

Unit call ammo3 – modification of **ammo** command. Returns 3-element numeric vector:
0 - ammo for primary weapon,
1 - ammo for secondary weapon
2 - pistol ammo
If some of these weapon slots is empty then correspondent value will be -1

Man call emptySlots - how many empty slots for primary and secondary weapon mags has unit **Man**

Man call emptySlots2 – returns 2-element numeric vector:
0 - empty slots for primary and secondary weapon mags of **Man**;

1- empty slots for pistol magazines of **Man**

Man call handgun – pistol-slot weapon of **Man.**
Returns "" if no such item. Pistol-slot item is not exactly pistol, say, if you uses SLX_Melee you maybe need additional check if it's pistol or not

[UnitInfo, WeaponInfo, magazines] call invAdd
alternative syntax:
[UnitInfo, WeaponInfo, mag1, amount1, mag2, mag3, mag4, count4...] call invAdd – weapon(s) and magazine(s) will be added to **UnitInfo.**

UnitInfo can be a unit, unit array, group or trigger. **WeaponInfo** can be weapon name or an array of weapon names. All consequent elements are magazine info – array of magazine names or several strings and numbers

[UnitInfo, WeaponCargoInfo, MagazineCargoInfo] call invAddCargo - weapon(s) and magazine(s) will be added to cargo section for each **UnitInfo** element.
WeaponCargoInfo can be like that: `["LAWLauncher", 4, "M16", 10, "M21", "M60", 2]`, if there is no amount after class name – default value of 1 will be used, or just be an array of weapon names. **MagazineCargoInfo** has the same structure

[UnitInfo1, UnitInfo2] call invClone – copy inventory of argument (0) (units in **UnitInfo** format) and set this inventory for each unit defined by **UnitInfo2.** If count of **UnitInfo1** units is less then count **UnitInfo2** units, then the rest units of **UnitInfo2** will be equiped like the last one from **UnitInfo1**

Unit call invCreate
alternative syntax:
ClassName call invCreate – creates inventory array – nested array of weapons and magazines of the given unit. The 2rd variant of this function is not recommended to use in big loops because of camcreate command usage. Created inventory can be used this way: **[[UnitInfo]+Inventory] call invSet**

Unit call invCreate2
alternative syntax:
ClassName call invCreate2 – similar to **invCreate**, but also returns «abilities» like "Put" or "Throw"

UnitInfo call invGet: creates inventory – array of 2 elements: all weapons and all magazines of **UnitInfo**. **UnitInfo** may be a group, unit, units or trigger

UnitInfo call invGet2 – similar to **invGet2**, but also returns «abilities» like "Put" or "Throw"

[UnitInfo, WeaponInfo, magazines] call invOwners
alternative syntax:
[UnitInfo, WeaponInfo, mag1, amount1, mag2, mag3, mag4, count4...] call invOwners – finds owners of given weapons with given magazines among **UnitInfo** units.
UnitInfo may be a group, unit, units or trigger. **WeaponInfo** can be weapon name or weapon name array. The rest elements are considered as magazine names and counts

[UnitInfo, WeaponInfo, magazines] call invRemove
alternative syntax:
[UnitInfo, WeaponInfo, mag1, amount1, mag2, mag3, mag4, count4...] call invRemove – removes weapons and magazines in **UnitInfo.**
UnitInfo may be a group, unit, units or trigger. **WWeaponInfo** can be weapon name or weapon names array. The rest elements are considered as magazine names and counts

[Unit, WeaponList, MagazineList] call invQueryCargo – returns 2 arrays - how many units of each weapon from **WeaponList** and how many units of each magazine from **MagazineList** are stored in cargo section of **Unit**. Works in campaigns only

[UnitInfo, WeaponInfo, magazines] call invSet

alternative syntax:
[UnitInfo, WeaponInfo, mag1, C1, mag2, mag3, mag4, C4, mag5..] call invSet –
sets given weapon(s) and magazines to **UnitInfo**.
UnitInfo may be a group, unit, units or trigger. **WeaponInfo** can be weapon name or weapon name array. The rest elements are considered as magazine names and counts

[UnitInfo, WeaponCargoInfo, MagazineCargoInfo] call invSetCargo - sets given weapon(s) and magazines to cargo sections of **UnitInfo**.
UnitInfo may be a group, unit, units or trigger.
WeaponCargoInfo can looks like: ["LAWLauncher", 3, "M16", 10, "AALauncher", "M60", 4], if there is no count after given classname – default value (1) will be used, or just be an array of weapon names. **MagazineCargoInfo** has the same structure

Unit call isRTO – true, if **Unit** has radio

Unit call isMGSoldier – true, if **Unit** has machine gun

Unit call isSniper – true, if **Unit** has sniper riffle

Unit call isLaserSoldier – true, if **Unit** has laser designator

Unit call isAASoldier – true, if **Unit** has portable SAM

Unit call isRPGSoldier – true, if **Unit** has reloadable grenade launcher like RPG

Unit call isATSoldier – true, if **Unit** has heavy AT launcher

[UnitArray,Type] call getSpecialist
alternative syntax:
[Grp,Type] call getSpecialist – returns the first found unit with such equipment **Type**;
There are 7 different types: **"RTO", "AA", "LD", "MG", "SNP", "RPG", "AT"**

[UnitArray,Type] call getSpecialists
alternative syntax:
[Grp,Type] call getSpecialists - returns all found unit with such equipment Type;
There are 7 different types: **"RTO", "AA", "LD", "MG", "SNP", "RPG", "AT"**

MagazineName call magSize – determines, how many slots does it need to keep this ammo in inventory

Man call radioState – the state of unit's radio.
>0: radio charged, 0: no batteries, -1: unit has no radio

Practice: «Examples (part 3).Intro», «TestCampaign»
Back to subject list

Machine Learning

[Data, ClassLabels] call ANN1Learn
alternative syntax:
[Data, ClassLabels, Ages] call ANN1Learn – simple 1-layer neural net,
can solve simple classification problems with good speed.
Data – array of arrays (points in multydimensional space of numeric patterns).
Subarrays lengths==number of patterns.
Array **ClassLabels** contains integers 1 and 2 – labels for 2 classes of some objects.
Ages – number of learning circles, default value = 200.
Returns a simple array: weight of each pattern + [bias weight]

[Weights, NewPoint] call ANN1Use – use **Weights**, calculated by **ANN1Learn**,
to find the class number of unclassified point **NewPoint**.

Returns 1 or 2

[Weights, NewPoints] call ANN1UseM – use **Weights**, calculated by **ANN1Learn**,
to find the class numbers of all points in array **NewPoints**.
Returns an integer array of 1 and 2

Array_0_1 call boolNot – turn each zero in pseudo-boolean array to 1 and vice versa

[Array_0_1, anotherArray_0_1,strCode] call boolOperation – perform dyadic boolean operation on each pair of corresponded array elements.
Arrays must contain 1 and 0 only. **StrCode** can be the one of 8 possible strings, each operation has a synonym:
"and" - "&&", "or" - "||", "xor" - "!=", "equal" - "=="

6 functions below is a measures of difference between 2 points in multidimensional space of numeric patterns.
All but **h_dist** works with numeric data.
All but **cos_dist** returns numbers in range [0, +inf]

[NumArray1, NumArray2] call che_dist - «chess» or Chebyshev distance = maximal of pairwise distances between 2 arrays. Not very accurate, but fast.

[NumArray1, NumArray2] call cos_dist – cosine distance.
Popular in fuzzy text match.
Geometrical mean: if **NumArray1** и **NumArray2** - lines, defined by 2 points each, then 1-(**[NumArray1,NumArray2] call cos_dist**) is a cosine of angle 'tween lines

[NumArray1, NumArray2] call e_dist – eucleadian distance. One of the most popular metrics in machine learning

[Array1, Array2] call h_dist – Hamming distance 'tween the arrays, the number of mismatches between corresponded positions. The fastest distance of all.
Arrays can contain elements of any type exept arrays, booleans and nil, but I recommend to use only integers and strings. The arrays must be of single type - not mixed

[NumArray1, NumArray2] call m_dist – «manhatten» or «city-block» distance between arrays. One of the fastest to compute

[NumArray1, NumArray2] call k_dist – Canberra measure. Weighed variation of **m_dist.** Use it if other metrics doesn't work good

[Data, Labels, K] call etalonLearn
alternative syntax:
[Data, Labels, K, DistFunction] call etalonLearn – creates array of etalons (arrays)of length **K**, by compression original **Data** with «K means» clustering method and an array of corresponded class labels.
2 functiones described below can use the result to get class labels of unclassified points

[Etalons, ClassLabels, Point] call etalonUse
alternative syntax:
[Etalons, ClassLabels, Point, DistFunction] call etalonUse – determines class label of **Point** by using class label of the nearest «etalon» - realization of popular «nearest neighbor» method. **DistFunction** determines, what expression to use to calculate distance between 2 arrays. Default method is **e_dist**

[Etalons, ClassLabels, Points] call etalonUseM
alternative syntax:
[Etalons, ClassLabels, Points, DistFunction] call etalonUseM – determines class labels of all **Points** by using class labels of the nearest «etalons» - realization of popular

«nearest neighbor» method. **DistFunction** determines, what expression to use to calculate distance between 2 arrays. Default method is **e_dist**

[Array, Condition] call getMatchArray – returns an array containing 1 where the **Condition** is right when applied to curent **Array** element and containing zeroes – where the **Condition** is wrong

[Data, K] call kMeans
alternative syntax:
[Data, K, DistFunction] call kMeans – find **K** clusters in multidimensional **Data** by «k means» algorithm. It can be used to compress datasets. Don't use too big **K** unless you want empty cluster problem and bad clasterisation as a result. **DistFunction** determines, what expression to use to calculate distance between 2 arrays. Default method is **e_dist**

[Array, Xpression] call makeDataRecords – call **Xpression** for each **Array**, element, the result is a nested array, similar to database records.
Xpression sample:
{_x knowsAbout player}, {getDammage _x, skill _x}
Attention: don't enclose **Xpression** in quad brackets – an expression will be enclosed in brackets automatically.
Purpose: get information from unit array to use in machine learning methods

[Array1, Array2] call match – percent of match between 2 equal-length arrays. Purpose: in superwised learning it compares the original class labels of test data and class labels calculated by some machine learning algorithm

NumArray call scale01 – scale an array to make all it's elements lay in range [0...1]

[NumArray, A, B] call scaleMinMax – scale an array to make all it's elements lay in range [**A**...**B**]

[Data, Labels] call split4LearnAndTest
alternative syntax:
[Data, Labels, ratio] call split4LearnAndTest – split data and correspondent class labels into learn and test sets. It's possible to define the percent of data to use in learning part, recommended range is [25...75] percents or [0.25...0.75].
By default, the data and labels will be splitted into approximately equal-size parts.
The resulting array has 4 elements:
0 – data to learn (matrix);
1 – class labels to learn (array of 1 and 2);
2 – data to test (matrix);
3 – class labels to test (array of 1 and 2)

Practice: «Examples (part 1).Intro»
Back to the subject list

Markers

[MarkerName, Delay] call animateMarker
alternative syntax:
[MarkerName, Delay, ScaleFactor] call animateMarker – make **MarkerName** pulse. Values of **ScaleFactor** lays in range [0...1000], **ScaleFactor** < 1 means that the size of that marker will be reduced with pulsing.
Default value of scale factor is 0.5 – size of the marker will be decreased twice

[Unit, MarkerName] call attachMarker
alternative syntax:
[Unit, MarkerName, Delay] call attachMarker – attach marker to unit.
As an option, you can define delay of marker position updating

MarkerName call detachMarker – remove **MarkerName** from attachment script.
If you want to attach it again it's recommended to wait several seconds after detachment

MarkerType call getMissionMarkers
alternative syntax:
MarkerTypeArray call getMissionMarkers – needs Fwatch 1.14+ to work, returns all markers of given type(s) exept the user-defined ones

MarkerName call hideMarker – hides the marker by reducing it's size 5000 times

[MarkerName, Pos] call markerDist
alternative syntax:
[MarkerName, Unit] call markerDist
alternative syntax:
[MarkerName, MarkerName2] call markerDist – distance from **«Marker1»** to something else

[Pos, MarkerArray] call nearestMarker
alternative syntax:
[Pos,MarkerArray, MaxDistance] call nearestMarker – find the closest to **Pos** marker in **MarkerArray**. Option: the search radius

MarkerName call showMarker – makes hidden marker visible by restoring it's size (*5000)

MarkerName call stopMarkerAnim – **MarkerName** will be removed from animation script.
To animate this marker again it's recommended to wait several seconds

call userMarkers – an array of all existing user-defined markers on the map

Practice: «Examples (part 1).Intro»
Back to subject list

Math

Number call ceil – ceiling, the smallest integer not less than **Number**

[Matrix, I] call column – select I-th column from the matrix

[Matrix, IntArray] call columns – select matrix columns, enumerated in **IntArray**.
Returns new matrix with reduced subarray size

Int call dec2bin – integer number – to array of 0 and 1

QuadMatrix call determinant – determinant of (quad) matrix, the matrix is not a movie but rather array with equal-length subarrays, «quad» means what length of this array==subarray length. To use in algebraic and applied geometry tasks

Int call digits – integer number to array of digits

Number call floor – the largest integer not greater than **Number**

[Number, [min1, max1], [min2, max2]...] call isInRanges
[Number, [min1, max1, min2, max2...] call isInRanges – true, if **Number** lays within one of ranges

[Number, Base] call log2arg – logarithm of **Number** with base **Base**

NumberArray call mean – arithmetic average

[Matrix, I, J] call minor – returns new matrix by removing row **I** and column **J** from old **Matrix**. If **I** or **J** is less than 0, then this row/column will be left on it's place

Array call mode – the most frequent element in **Array**

[Any, Nrows, NCols] call mtxBuild
alternative syntax:
[Array, Nrows, NCols] call mtxBuild
alternative syntax:
Array call mtxBuild – creates matrix – array with equal-length subarrays.

In first case it will be an array of **Nrows** rows (subarrays) and **NCols** columns (subarray length), filled with element **Any.**
In second case the matrix will be filled with **Array** values.
In third case, if **Array** array length == N*N (quad of integer number) then quad matrix (N rows, N columns) matrix will be created

Matrix call mtxCopy – creates independent copy of given matrix

QuadMatrix call mtxInverse – inverse of (quad) matrix – nested array of equal-sized rows, the rows number==column number (length of row). To use in applied geometry tasks

[Matrix1, Matrix2] call mtxProduct – multiply 2 matrixes according the rules of linear algebra. Usualy it's not equal to this: **[Matrix2, Matrix1] call mtxProduct**

Number call round
alternative syntax:
[Number, NumbOfDigits] call round – round the number to the nearest integer or with given precision

Number call sign – sign of number: -1 for negative values, 0 for 0, 1 for positive numbers

[Array1, Array2] call vectorProduct – sum of pairwise products of 2 array elements

[bool, bool2] call xor – «exclusive or», analogue of **A!=B** for boolean vars

Practice: «Examples (part 3).Intro»
Back to subject list

Random

NumArray call rndCase – random integer in range [0, (count **NumArray**)-1].
The bigger element from NumArray – the better probability of it's index _i.
Probability of presence of _i = **NumArray**[_i] / SUM(**NumArray**)

Array call rndElement – select random element from array. All elements can be picked with the same probability 1/(count array)

[Array, NumArray] call rndElementPro – select random element with non-equal chances.
Probability of _i-th element selection:
NumArray[_i] / SUM(**NumArray**)

[Min, Max] call rndFloat – random float number between **Min** and **Max**

Max call rndInt – random integer 'tween 0 and **Max**-1 inclusive

call rndLandPos – random position on land. Works on islands with non-zero static objects number (doesn't work on Desert Island)

[Array, N] call rndSample – **N** random elements from **Array**, indexes are unique

[Array, N] call rndSample2 – **N** random elements from **Array**, indexes may be repeated

Array call shuffle – shuffle array randomly. Returns no result, **Array** will be changed

NestedArray call shufflePivoted – shuffles all **NestedArray** subarrays, saving pivot between subarrays, i.e. all index rearrangements will be the same

Practice: «Examples (part 2).Intro»
Back to subject list

Search

[Array, ConditionList] call bestFit
alternative syntax:
[Array, ConditionList, Weights] call bestFit – select element from **Array**, which fits maximal number of conditions in **ConditionList**, if the importance of conditions is different, their weight factors will be taken into account

ClassName call csvQuery – creates query function to get information from CSV-based OOP-style database. Developed to store config information.
DASH_Library has 4 «classes»: "ammo", "mag", "weapon" and "letter".
Scripters can create their own «classes» in the stringtable.csv of addon/mod/mission:

```
Language, English
"ClassTag|base", "{_propName1}, defaultValue1, … {_propNameN}, defaultValueN"
```

Class name is case-sensitive, but property names are not.
All property names must be correct, non-repeating local variable names.
If query arguments are incorrect (no such class or no such properties) then empty string will be returned and globalChat message occures.
Default values types can be string, number, side or mixed array of these types.
The instances of class starts from class tag.
4 example:

```
Language, English
...
"mag|base", "{_ammo},[],{_count},1,{_size},256,{_pic},{},{_norm},{},{_hd},{}"
...
"mag|JAM_AT4Rocket","_ammo={JAM_AT4ammo};_size=2*256;_pic={\JAM_Magazines\pics\JAM_AT4Rocket.paa}"
...
"mag|JAM_CAVS_AT4Rocket", "_base={JAM_AT4Rocket};_ammo={JAM_CAVS_AT4ammo}]"
```

There is 1 property name what must not be defined in base class: "_base".
It provides inheritance among records.
"mag|JAM_CAVS_AT4Rocket" has the same property values as his "parent", "mag|JAM_AT4Rocket", but "_ammo" param is different.
Note, what "_count" value is not present in both cases – default value from the "constructor" (1) is used.

3 functions below are implementation of **csvQuery** technology for the infantry ammunition config values requests. They are all case-sensitive.

ClassName call cfgAmmo – returns an array of config properties of given ammo **ClassName**.

0 - _sim – simulation. Default: "shotBullet";

1 - _hit – array of [hit, indirectHit, indirectHitRange]. Default: [0, 0, 0];
2 - _range – array of [minRange, maxRange]. Default: [0, 0];
3 - _ir – IRLock. Default: 0 (can attack infantry);
4 - _air – AirLock. Default: 0 (can't attack planes && choppers);
5 - _col – not quite from config. Colors for smokes and flares from this list:
"red",
"green",
"yellow",
"black",
"white",
"purple",
"blue",
"orange".
Default: {}

ClassName call cfgMag – returns an array of config properties of given magazine **ClassName**.

0 - _ammo – ammo class(es) – string or array. Default: [];
1 - _count – count of projectiles (clip size). Default: 1;
2 - _size – size of magazine (slot size). Default: 256;
3 - _pic – picture path. Default: {};
4 - _norm – for High Dispersion mags returns their normal version. Default: {};
5 - _hd – for normal mags can return their High Dispersion versions. Default: {};

ClassName call cfgWeapon – returns an array of config properties of given weapon **ClassName**.

0 - _name – weapon name (char array, not always matches config values). Default: [];
1 - _type – weapon type. Available types:
"AR" – assault riffle;
"AR+GL" – assault riffle with grenade launcher;
"SMG" – pistol-caliber automatic weapon;
"SMG+GL" – SMG with grenade launcher;
"GL" – grenade launcher like M79 or 6G30;
"AT" – anti-tank weapon like RPG or CarlGustav;
"AA" – portable SAM;
"LMG" – machinegun what takes 1 slot;
"MG" - machinegun what takes 1+16 slots;
"SHG" – shotgun;
"R" – rifle (not automatic primary weapon);
"SR" – sniper rifle what takes 1 slot;
"AMR" – heavy sniper rifle (slots 1+16);
"P" – pistol or revolver;
"MP" – pistol-slot automatic weapon;
"LD" – laser designator;
"RADIO" – radio;
"RUCK" – rucksack;
"PROXY" – static weapon component like tripod;
"MELEE" – knifes, swords etc;
"PUT" – put ability;
"THR" – throw ability;
Default: {};
2 - _side – faction which uses this gun. Default: side objectNull;
3 - _mz – secondary muzzle name. Default: {};
4 - _scope – true, if SMG/riffle/machinegun is scoped. Default: false;
5 - _mag – magazines 4 first muzzle (string or array). Default: [];
6 - _mag2 – magazines 4 second muzzle like GP25 or M203 (string/array). Default: [];
7 - _pic – picture path. Default: {};
8 - _sd – true, if weapon is silent. Default: false

Note: _mag and _mag2 from **cfgWeapon** can be handled with **cfgMag** and _ammo from **cfgMag** result can be handled with **cfgAmmo** – so all these params are case-sensitive.

Yet another stringtable-based technology: access to global array elements with point-separated string keys: **db*** functions.

3-level structure of global array must be defined in stringtable. _nodes variables provides possibility to create whole array with default values (call _init) using function **dbCreate**. Key names are case-sensitive, but variable names are not. DASH_Library resource array is a good example:

```
"gdb|dash", "_get={dash};_set={};_nodes=[{addonflags},{propnames}...

"gdb|dash.propnames", "_get={dash select 4};_set={dash set[4,_this]}"

"gdb|dash.fog", "_get={dash select 18};_set={dash set[18,_this]};_nodes=[{now},{forecast},{changespeed}]"

"gdb|dash.fog.now", "_get={(dash select 18)select 0};_set={(dash select 18) set[0,_this]};_init={0}"

"gdb|dash.fog.forecast","_get={(dash select 18)select 1};_set={(dash select 18) set[1,_this]};_init={0}"
```

So, then dash array will be created, **dash.propnames** will be [] (default value), **dash.fog.now** will be 0

[stringKey, notArray] call dbAdd
alternative syntax:
[stringKey, [notArray1, notArray2, notArray3...]] call dbAdd
– adds one or many values to some global array according **stringKey**

[stringKey, notArray] call dbAddNew
alternative syntax:
[stringKey, [notArray1, notArray2, notArray3...]] call dbAddNew
– adds one or many values to some global array according **stringKey**, only values not present in database will be added

stringKey call dbCreate – creates global array with default values

[stringKey, notArray] call dbDelete
alternative syntax:
[stringKey, [notArray1, notArray2, notArray3...]] call dbDelete - removes one or many values from some global array according **stringKey**

stringKey call dbGet – returns global array element.
Arrays values will be returned by reference

stringKey call dbHint – shows content of given array

[stringKey, anyValue] call dbSet – sets new value for global array element

[Array, Var] call find2 – case-sensitive standard search function. Works as find command in CWA, works much slower in OFP, but array can contain any values as well

[StringArray, String] call findString – returns the index of first occurance **String** in **StringArray** returns -1 if such a string was not found. Case-insensitive

[Array, Var] call findAll – returns all indexes of occurance **Var** in **Array** or [] if nothing found. The function is case-insensitive

[Matrix, ColNumber, Var] call findInColumn – returns index of the first occurance of **Var** in column **ColNumber** of **Matrix**. Returns -1 if nothing found. The function is case-insensitive

[Array, Var] call findInNested – returns index of the first subarray which contains **Var** or [] if no such subarrays. The function is case-sensitive

[NumericArray, Number] call findInSorted – returns index of **Number** in **NumericArray** sorted in ascending order. Much faster then most of find* functions, but array must be prepared with **qSort** or **bSort**

NumArray call max – array maximum

NumArray call min – array minimum

[Array, Database] call selectByKeys – returns subarrays of **Database** with the keys (0-th elements) present in **Array**. The function is case-sensitive

[ExprWHERE, arrayFROM] call sqlDelete - removes records from **arrayFROM** according SQL-like query **exprWHERE**.
ExprWHERE often contains comma-separated field names from **arrayFROM(0)**, reserved name _i (record index) is also available; an expression result must be true or false.
The names mentioned in **exprSELECT** and in the consequent expressions must be present in **arrayFROM** table and starts from underscore.
arrayFROM example with 2 records:

```
[
  ["_Name", "_Armor", "_GunCaliber"],
  ["T55G", 300, 105],
  ["ZSU", 250, 23]
]
```

ExprWHERE sample: **"(_Name == {ZSU}) or (_Armor < 300)"**.
It's also possible to use **true** to remove all records.
sqlDelete returns no result, source table **arrayFROM** will be changed

[exprSELECT, arrayFROM] call sqlSelect
alternative syntax:
[exprSELECT, arrayFROM, exprWHERE] call sqlSelect
alternative syntax:
[exprSELECT, arrayFROM, exprWHERE, exprORDERBY] call sqlSelect - returns records (array of equal-structure arrays) from **arrayFROM** according SQL-like query.
ExprSELECT often contains comma-separated names from **arrayFROM(0)**, name _i – record index is also available.
ExprSELECT example:
"_Name, _Armor+100, _MainCaliber".

It's also possible to use **"*"** to get full records.
The names mentioned in **exprSELECT** and in the consequent expressions must be present in **arrayFROM** table and starts from underscore.
See an example of **arrayFROM** in **sqlDelete** description.
ExprWHERE – expression which must return true or false, for example **{(_Name=="ZSU") or (_Armor<300)}**.
ExprORDERBY – expression which must return number, used to sort the resulting array. By default, ascending sort used, expression **{-_Armor}** will sort the result in descending order of **_Armor** field.
Source table **arrayFROM** will not be changed

Practice: «Examples (part 3).Intro»
Back to subject list

Sort

All functions mentioned below but **merge2sorted** has no result, they reorders original argument(s)

NumArray call bSort – sort array in ascending order. Method – bubble sort.
Very slow on big arrays, but good for arrays of length < 10

NumArray call bSort2 – sort 2 arrays in ascending order of first one. sort array in ascending order. Very slow on big arrays, but good if length of subarrays < 10

NumArray call qSort – sort array in ascending order. Method – quick sort

[Array, Sample] call qSortByExample – sort **Array** elements in order of these match indexes in **Sample**, for example in alphabet.
Method – quick sort

[Array, Expression] call qSortByExpr –
sort **Array** in ascending order of **Expression** applied to each array element.
Method – quick sort

[NumArray, Array1, Array2...ArrayN] call qsortM – sort nested array by ascending order of subarray[0]. Method – quick sort

[Array, I, J] call idxSwap – swap two indexes in array

Array call inverse – inverse the order of elements in **Array**

[Array1, Array2] call merge2sorted – merge 2 arrays sorted in ascending order. The result array is also sorted in ascending order

Practice: «Examples (part 3).Intro»
Back to subject list

System

[AnyVariable, StrComment] call debug – stores debug information, used with **hintDebug**

ListOfGlobalVarNames call deleteGlobals – removes several global variables.
Returns no result

[var1, var2, … varN] call findLostVar – returns index of the first not-defined variable in the list (or first nil element in array). Returns -1, if all vars were defined (no nil elements)

call fw113 – true, if Fwatch 1.13+ on.
Several functions don't work WITHOUT Fwatch, another few good fns works even better WITH it

call getLanguage – return the selected game language.
Available variants: **"Spanish", "Czech", "English", "Italian", "German", "French", "Russian"**. Not recognised language is considered as **"English"**

call hintDebug – shows debugging information stored with the help of **debug**.
There is a delay before and between hints

AddonName call isAddonLoaded – checks if addon/mod **AddonName** is present.
Mods are to be written with «dog»: **"@bwmod" call isAddonLoaded.**
There are about 20 signatures for mods and 210 – for addons in the database.

All addons found are stored in temp. subarray, so the second search of the same addon will be faster. You will see hint message, if addon is not found in database

call isCWA – returns true in CWA and false in OFP

VarName call isNil – returns true, if **VarName** variable is not defined, or destroyed by {**VarName**=nil} expression

Key call optionGet – returns correspondent value from global associative array

[Key, NewValue] call optionSet – sets new value for key of global associative array. Not existing key will be created and initialized with **NewValue**

Var call print – formats **Var** to string and shows it to player with **sideChat** command

[PropOwner, StrPropName] call propGet – returns the value of user-defined property **StrPropName** from **PropOwner**. **PropOwner** can be unit, group or fictitios object defined by string. Returns any value, but if no such **PropOwner** or no such property in his property set - then "undefined" string will be returned

[PropOwner, PropName1, PropName2...] call propsRemove – removes one or more user-defined properties belonged to **PropOwner** (unit, group, string or side). Returns no result

[PropOwner, StrPropName, Value] call propSet – sets user-defined property **StrPropName** belonged to **PropOwner** (unit, group, string or side) to **Value**

[OwnersArray, p1, v1, p2, v2...] call propSetM – gives the same property set for each unit, group or string in **OwnersArray**. Successfully tested on 361 unit with 50 properties of different type, including arrays

Expression call test – prints **Expression** and it's (separated with double line) result with hintC command. Arrays are shown with indents and newlines to demonstrate their stucture, maximal number of strings to show = 32, if there are some more lines – the ellipsis will be used.
If no result returned, fore example, {a=5}, or some of **Expression** arguments was equal to nil – then only **Expression** will be printed.
It doesn't work if an **Expression** returns trigger or camcreated object because of **varTypeSafe** usage. It mentioned to use in scripts, if you want to call it several times – use delay ~2 or at least ~0.01 between calls of **test**

Variable call varType – returns variable type, it can be "number", "string", "array", "bool", "side", "object", "group" or "trigger". Don't use strings with length > 2000 symbols (2Kb) – otherwise you'll see CWA crash. With big arrays it works fine

Variable call varTypeSafe – returns the variable type, can return "number", "string", "array", "bool", "side", "object" or "group". It cannot crash CWA, but «stumbles» over many camcreated/map static objects and (with Fwatch off) on triggers too (error message will be thrown)

call version – returns DASH_Library version number

[Condition, Xpression] call whileDo – works similar to **while Condition do Xpression,** but not limited in number of iterations. Iteration limit for «while» loop = 10 000, you can also use forEach loop to work around this problem

Practice: «Examples (part 1).Intro», «TestCampaign»
Back to subject list

Text

Country call **alphabet** – returns char array with an alphabet of given country.
6 possible variants:**"gb", "us" "ru"** – lover-case symbols, **"GB", "US" "RU" –** uppercase symbols, returns empty array for all other keys

[Array, strDelimiter] call array2Report
alternative syntax:
[Array, strDelimiter, bUseEnds] call array2Report – format array as a report, using **strDelimiter** and (as option) endings to non-unique elements

[Array, FormatSingle, FormatPlural, Delimiter] call array2ReportFMT – format array as a report, using string **FormatSingle** to format unique elements, **FormatPlural** – for non-unique elements and string **Delimiter** to separate elements
In string **FormatSingle**, «%1» means array element,
in string **FormatPlural**, «%1» means element and «%2» means number of such an element in array

Array call array2string
alternative syntax:
[Array, StrDelimiter] call array2string -
format array to strings, elements will be separated with **StrDelimiter** if an array has count == 2 or with "" in other cases

[Array_of_CharArrays, CharArray] call catAfter
alternative syntax:
[Array_of_ Strings, String] call catAfter – add string/char vector to the end of each string/CV array element

[Array_of_CharArrays, CharArray] call catBefore
alternative syntax:
[Array_of_ Strings, String] call catBefore – add string/char vector to the start of each string/CV array element

CharArray call cv2string – join characters/strings to get 1 string.
Created to work with small char vectors (count < 20)

[CV1, CV2] call editDistance – measure of difference between 2 character vectors of any length, shows how difficult to edit one world to get another one. Added or removed char have fixed cost==2. An expression to get character replacement cost can be set globally with **fzTextProps**, default value is {1}, in general case it will be 2-argument function

[CV1, CV2] call endsWith
alternative syntax:
[CV1, CV2, bIgnoreCase] call endsWith – returns true, if char vector **CV1** ends with **CV2.** Optional parameter «ignore case» (true/false) has default value «true»

Unit call firstName – unit's name (without surname). Fwatch 1.13+ required

[CV1, CV2] call cosineCVCVDistance – cosine distance between char vectors, the result lays in range [0...1], it's the result of implementation **cos_dist** for arguments, prepared with **symbolCounts**. Works with char arrays of different lenght.
Advantage: not affected with letter rearrangements – one of the most popular error in man-printed texts.
Disadvantage : anagramms like `["e", "a", "t"]` and `["t", "e", "a"]` are always gets error==0 (complete match)

[String, CV] call FuzzyStrCVCompare – returns true, if there are less than 2 missmatches between given string and char vectors. Mismatch means here "added letter, missed letter or replaced letter". With Fwatch on it's more convinient to use **fzTextMatch** instead

[Text1, Text2] call fzTextMatch
alternative syntax:
[Text1, Text2, MaxError] call fzTextMatch – returns true, if the error of fuzzy text comparison less of or equal to **MaxError**. There are 2 different comparison methods: "edit"- **editDistance** and "cosine"-**cosineCVCVDistance**. **Text1** and **Text2** can be both strings or char vectors or string and char vector.
With Fwatch 1.13 on string arguments will be splited into symbols and than compared like char vectors.
Without Fwatch 2 char arrays comparison is just the same ("edit" or "cosine"), 2 strings will be compared in simplest (==) way and in mixed case
[CV, String] or [String, CV] - **FuzzyStrCVCompare** function will be used – this method is not so useful like "edit" and "cosine" but better than nothing

[Array1, Array2] call inStr – the position in **Array1**,
from there **Array2** starts from, -1 if these array is not the subsequence of **Array1**.
Created to work with char arrays, but works with numeric ones too

Unit call lastName – unit's surname. Need Fwatch 1.13+

CharArray call lCase – character array to lower case

CharArray call lCaseRu – Cirillic version of **lCase**

CharArray call noDoubleLetters – removes double (triple, etc) simbols from character array. Returns reduced char array

Number call nth – numerics:
1 ==> "1st", 12 ==> "12th"

CV call removeOFPECtag: remove tag from the head of class name (weapon, unit, ammo ...) splitted into chars so it will be more useful for reports. Tags with underscore like ["S", "L", "X", "_"] will be always removed, tags without it, ["K", "E", "G"] for example, must be found by this fn in database at first, so sometimes it won't work

[StrArray1, StrArray2] call removeStrings – remove one string/«char» array from another ignoring case

[CV, CV_Old, CV_New] call replaceCV – replace the first found subsequense of characters **CV_Old** in character array **CV** with **CV_New**. Returns new array

[Array, Delimiter] call split – splits **Array** into subarrays, using **Delimiter**.
Aimed to work with char arrays, but must work with other types exept booleans.

[CV1, CV2] call startsFrom
alternative syntax:
[CV1, CV2, bIgnoreCase] call startsFrom – returns true, if character vector **CV1** starts from **CV2**. Optional parameter «ignore case» (true/false) has default value «true»

String call string2CV – split the string into characters (strings of length 1).
Fwatch 1.13 required

[CV, StartIdx, EndIdx] call subStr – returns new array, maked of original array's elements from index **StartIdx** to index **EndIdx**. Works with arrays of any type

Char Vector call symbolCounts – returns numeric vector of length 36, which shows, how many time 26 letters and 10 digits were encountered in character vector

CharArray call uCase – character array to upper case

CharArray call uCaseRu – Cirillic version of **uCase**

Practice: «Examples (part 2).Intro»
Back to subject list

Vehicles

Veh call canFire2 – modification of **canFire** command, it also checks for some weapons and ammo. Binocular, NVG and CarHorn are not mentioned as weapons

Veh call canMove2 – modification of **canMove** command, it also checks fuel level

Veh call cargo – array of units in cargo section of **Veh**

Veh call effectiveCommander – who is in charge in **Veh**?

UnitArray call enableQueryVehicles – creates inner unit array to make **getVehicles** function work. It doesn't usefull with Fwatch 1.13+ on

VehType call getVehicles
alternative syntax:
ArrayofTypes call getVehicles – works better with Fwatch 1.13, returns array of objects (vehicles or empty objects), belonged to class **VehInfo** or to it's child classes or matching to one of classes from **VehInfo**. To work without Fwatch you need big trigger of activation type Any - Present and **enableQueryVehicles** function

Veh call hasCommanderPlace – does the vehicle have commander's place?

Veh call hasDriverPlace – does the vehicle have driver's place?

Veh call hasGunnerPlace – does the vehicle have gunner's place?

[Unit, VehInfo] call knownVehicles
alternative syntax:
[Unit, VehInfo] call knownVehicles
alternative syntax:
[Unit, VehInfo, Knowledge] call knownVehicles
alternative syntax:
[Unit, VehInfo, Knowledge, BlackList] call knownVehicles – find all vehicles/empty objects known by **Unit**, belonged to class **VehInfo** or to it's child classes or matching to one of classes from **VehInfo**, with **Unit** knowsAbout _x value >= **Knowledge** (if not defined – 1.4) and not present in object array **BlackList**, if it was defined.
Returns 2 arrays – array of objects and array of «knowsabout» values, so objects can be sorted by knowledge. To work without Fwatch you need big trigger of activation type Any - Present and **enableQueryVehicles** function.

[PosInfo, VehInfo] call nearbyVehicles
alternative syntax:
[PosInfo, VehInfo] call nearbyVehicles
alternative syntax:
[PosInfo, VehInfo, Radius] call nearbyVehicles
alternative syntax:
[PosInfo, VehInfo, Radius, BlackList] call nearbyVehicles – finds all vehicles/empty objects near **PosInfo** (unit or position) of class **VehInfo** or belonged to it's child classes or matching to one of classes from **VehInfo**, with distance from **PosInfo** not exceeding **Radius** (default value=50 m) and not present in object array **BlackList**, if it was defined. Returns 2 arrays – array of objects and array of distances, so objects can be sorted by distance. To work without Fwatch you need big trigger with activation type Any - Present and **enableQueryVehicles** function

[PosInfo, VehInfo] call nearestVehicle

alternative syntax:
[PosInfo, VehInfo] call nearestVehicle
alternative syntax:
[PosInfo, VehInfo, Radius] call nearestVehicle
alternative syntax:
[PosInfo, VehInfo, Radius, BlackList] call nearestVehicle – find nearest vehicle/empty object near **PosInfo** (unit or position), of class **VehInfo** or belonged to it's child class or matching to one of classes from **VehInfo**, with distance from **PosInfo** not exceeding **Radius** (default value=50 m) and not present in object array **BlackList**, if it was defined. Returns object. To work without Fwatch you need big trigger of activation type Any - Present and **enableQueryVehicles** function.

[Veh, NewSpeed] call setSpeed – immediately set the speed of **Veh** to **NewSpeed**

Unit call vehicleRole – name of unit's position in vehicle.

Possible results: **"Driver", "Gunner", "Commander", "Cargo", "None"**

UnitArray call vehicles – returns array of vehicles occupied by **UnitArray**

Practice: «Examples (part 3).Intro»
Back to subject list

~~STOLEN~~ BORROWED

Some of the functions mentioned above are renamed and/or reworked variants of code created by experienced OFP scripters.
I also implemented a few generic scripting ideas of these and other «OFP jedies».
Respect comes to:

Baddo – **countBuildingPos;**

BINMOD team - **dirOfMove**, **vehicleRole;**

Bn880 – **varType;**

The Chain of Command band - **transpose** and idea of Neural Network for OFP;

DenVdmj – **formatOut, findLostVar**, **varType, varTypeSafe, invQueryCargo, sideUnknown** constant and the example of test campaign;

Dschulle – **grid2pos;**

ECP mod squad – global resource array conception;

Faguss - **qsort**, **qsortM, fwatchIsOn, string2CV,** help with Fwatch command **«:class token»** usage;

FOX2 – **pos2grid;**

Fragor1 – idea of user-defined properties;

General Baron - **findString**, **squadNumber**, **squadOrder;**

Igor Drukov – **relPos2d;**

Invasion44 mod creators – «sqf-database» conception;

KTottE - **pop**, **pushNew;**

LIBMOD gang – **cargo;**

Mandoble – **inFOV;**

MCAR project personnel – an idea to use stringtable.csv as a SQF code storage;

Mr.Peanut - **elevation**, **highestLowest;**

Raptorsaurus - **elevationAngle**, **inForest**, **groundSlope;**

SLX mod developers - **radioState**;

snYpir – **dir2obj;**

Spooner – **getLaserDot;**

Toadlife – **watchDir;**

Vectorboson - **groupIsCargo**, **getGroupSpeed**, **getGroupSpread**, **mostInjured**, **filterGroups**, **nth**, **shuffle**, **array2report**
+ idea of Rnd.sqf module, nowdays added to csv
+ help with **isNil**, **rndFloat** and **varType** improvement;

Special thanks:

DenVdmj - BIS scripting comref translation + detailed scripting reference;

Dschulle – PBOX utility;

Faguss – Fwatch program further development and technical support;

Lone~Wolf – Notepad++ plugin for SQF/SQS code highlighting;

Kegetys – Fwatch program creation;

Notepad++ team – for the best script editor ever;

Razorwings18 – OFPDialogueMaker utility;

Rinzzza – «all-in-one» fundamental OFP manual

CONTACT INFO

Questions, error reports and suggestions about scripts and functions for OFP/CWA please send to **andrey.melekhoff@mail.ru**

Flashpoint.ru and **OFPEC.com** accounts: SoldierEPilot