

Mr-Murray presents

ARMA

ARMED ASSAULT



EDITING GUIDE

Deluxe Edition

The Ultimate Mission Tutorial for the Armed Assault Mission Editor



E D I T I N G G U I D E

by Mr-Murray

Prologue

The contents of this Editing Guide will help you to make the work with the Armed Assault® Editor much more easier. You do not need any knowledge in programming to create interesting and fun full missions for Armed Assault®.

That's true, without any knowledge in programming! But it wouldn't be bad if you have some experience out of the work with the Operation Flashpoint® Editor of course, it would be an advantage but it is not quite needed. This Guide will explain you the parts of the Editor individually and many examples will help you to understand the single operations. Additional to this, the Armed Assault Editing Guide will show you the nearly unlimited possibilities which were offered you by the Editor.

After you worked with this Guide for some time, so you'll be able to create your own exciting Missions. The only thing you really need is creativity and lots of ideas which wants all to be come true. The possibility to create your own movies, which one can compare with Hollywood movies, and further the possibility to add your favorite Sound files into the Missions will pull the player in the ban.

Create dynamic Missions with different weather, time of day and furthermore different mission targets. Units, Objects or the Player himself are located at any other positions every time when the mission begins again. All these possibilities shouldn't actually anymore be a problem for you.

Now it is up to you and your ideas and your creativity to create new good Missions. This Guide doesn't contain an directing Book, no Scenarios or any other Stories for your Missions. That's all up to you. But with this Guide, you have all you need to make your ideas come true. And if anything doesn't work as you want to, so just relax, exit the editor and play some missions, go on with your campaign or enter the Multiplayer lobby to get new ideas for your mission.

Armed Assault® is setting up as Operation Flashpoint® successor with an own created script language from his ancestor. There was many new changes made here and there and lots of new stuff used in ArmA® but the basic concept is still the same. The Editor still enables the user to keep a quite well overview and the missions folder resp. their contents are still the same as well. So read, try and edit yourself with this Guide through the World of Armed-Assault.

Good luck and lot of fun with the Editor is wishing you BIS and Mr-Murray.

Annotation

This guide is meant as an introduction to the Armed Assault® Editor and shall make the work with the Editor much easier, especially for the editing beginners. The scripts, which are all shown here, are completely fictitious and can be further developed, of course.

The Editor offers much more possibilities than explained here in this book. That's why I'd like to insinuate the official Wiki:

<http://community.bistudio.com/wiki>

The Wiki is always up to date about everything related to editing, scripting and all around Armed Assault®. The basics are shown here and enable one to get an impression about what the Editor is able to do. That shall help you to let your ideas come true. All scripts which are shown and explained in the 6th Chapter can be found in the official forum.

www.forum.german-gamers-club.de

I say thank you

With this Guide I would like to say thank you to all Operation Flashpoint® fans who are all still keeping loyal to the game and I also would like to say thank you to Bohemia Interactive Studios, because without those guys, that game - with such a mass of possibilities - would never have been created.

I would further like to say thank you to the whole Mapfact team, BadAss, Chneemann, Flashpoint_K, JörgF, Kriegerdaemon, LockheedMartin\$ch, MCPXXL, OneManGang, Silola, Sniping-Jack, Raedor, Lester, Wüstenfuchs and our helping hands MemphisBelle, Simba, Marco-Polo-IV and Sgt.Ace, who supported me through the last years very much.

A very special thanks is dedicated to Raedor and Chneemann who were always helping me with trouble around ArmA®. Furthermore, I like to say thank you to Andre Scheufeld, Andreas Holzward, Rastatovich and Wolle for their amazing support. Real special thanks to the translator of this guide **MemphisBelle** and his helping hands **Metal0130** and **Matt Rochelle** for their very great revision support.

I also won't forget the people who are the most important to me; My family, my friends and especially my girlfriend, without their support I never could finish the project.

Yours,

Sascha Hoffmann aka Mr-Murray

Community Screenshot Contest

I have decided one way or another to intergrate some community work into this book. Because my guide already existed along with serveral versions and with the newest publication I wanted the community to be apart of it. So I talked with Morphicon and we both decided that a screenshot competition would be the best thing.

I thank you again to everyone who participated in the contest and I also thank the website Armed-Assault.de for realising the importance of this competition.

The results of this competition can be seen in the proceeds of this book. Most of them were placed on the single chapter directories, but many of them in the contents of each chapter as well.

The following images are the ones which has been voted by the community for the first three ranks. The winner are:

Winner 1: Marcus-Ergalla (Aljosha Rall)

Winner 2: Mr. Burns (Andreas Schmitz)

Winner 3: Stoned Boy (Frank Nobis)



Platz 1: Marcus-Ergalla



Platz 2: Mr Burns



Platz 3: Stoned Boy

Directory

Chapter 1: The Beginning

1.1	The User Interface	16
1.2	Adding Units	20
1.3	Adding Groups	26
1.4	Adding Triggers	27
1.5	Adding Waypoints	30
1.6	Synchronize	35
1.7	Adding Markers	36
1.8	Rotating Units And Objects	39
1.9	Merging Units	39
1.10	Edit Units With Allocated Waypoints	40

Chapter 2: The Files

2.1	The Missions Folder	42
2.2	The Mission.sqm	43
2.3	The Description.ext	48
2.4	The Stringtable.csv	51
2.5	The Init.sqs	53
2.6	The Script (.sqs)	54
2.7	The Function (.sqf)	55
2.8	The Paa-Format	55
2.9	The PBO	56
2.10	The Sound Files	56
2.11	The Lip-Files	57
2.12	The Overview	58
2.13	The Briefing	59

Chapter 3: Weapons – Vehicles – Units – Objects

3.1	The Hand Weapons And Static Weapons	64
3.2	The Weapons Class Name List	68
3.3	Arm And Equip Units	70
3.4	The Weapon And Ammo Crates	71
3.5	Load And Unload Vehicles	71
3.6	Weapon Selection In The Briefing	72
3.7	The Vehicle Classes	73
3.8	The Vehicle Weapons	76
3.9	The Unit Classes	77
3.10	The Shell Classes	80
3.11	The Object- and Building Classes	81
3.12	The Plant Classes	88

3.13	The Rock Classes	90
3.14	The Sign Classes	91
3.15	Getting Weapon And Magazine Types Displayed	92
3.16	Getting Fired Type	92
3.17	Does unit have a weapon?	92
3.18	Primary or secondary weapon of a unit	93
3.19	Does unit have ammunition?	93
3.20	Creating mines	93
3.21	Creating weapons and magazines	94
3.22	Getting weapon direction view displayed	95

Chapter 4: The Mission

4.1	The Mission Name	97
4.2	The Mission Start	97
4.3	The Mission Accessories	98
4.4	The Mission Appraisal	99
4.5	The Mission Targets	99
4.6	Finishing a Mission	101
4.7	Saving a Mission	103

Chapter 5: The Mission Accessories

5.1	Empty or locked vehicle	106
5.2	Driver/Passenger of a vehicle	106
5.3	Unit is not allowed to enter a vehicle	106
5.4	Unit in vehicle?	107
5.5	Vehicle only moves when unit has entered	107
5.6	Group already in vehicle when the mission begins	108
5.7	Let a unit get in and get out of a vehicle	108
5.8	Speed of a unit	108
5.9	Display the speed of a unit	108
5.10	Unit keeps standing	109
5.11	Getting a unit started	109
5.12	Unit is moving to its destination	110
5.13	Running patrol, drive or fly	110
5.14	Escape behaviour of a unit or a group	110
5.15	Moving units, objects, triggers and markers	111
5.16	Placing objects higher or lower	111
5.17	The height of a unit	112
5.18	Accurate helicopter landing	112
5.19	Unit is moving into a building	112
5.20	Unit is leaving / joining a group	113

5.21	Assigning a target to a unit	113
5.22	Unit turns to another Unit	114
5.23	Unit is selecting weapon	114
5.24	Inflct damage or heal a unit	114
5.25	Defining a death zone	115
5.26	Checking of an area	115
5.27	Bring about a certain behaviour of a unit in an area	115
5.28	Save or load a unit status	116
5.29	Degree of familiarity of a unit	117
5.30	Friendly enemy	117
5.31	Friendly forces	118
5.32	The alert	119
5.33	Dead as condition	120
5.34	Distance of two units or objects	120
5.35	Allocate a flag to a flagstaff	120
5.36	Burning fire	121
5.37	Add or remove switchable units	121
5.38	Read out and display player side, - name, -type	121
5.39	Opress player input	121
5.40	Force the map on the screen	121
5.41	Adjusting distance of view	122
5.42	Adjust the weather	122
5.43	Adjusting date and time of day	123
5.44	Slow motion or time sprint	123
5.45	Generating units and objects	124
5.46	Generate flares, smoke and explosions	126
5.47	Delete units and objects	127
5.48	Adjusting radio menu	127
5.49	Allocate a call sign to a group	128
5.50	Send a radio message	129
5.51	Creating sound	129
5.52	Using own sounds	130
5.53	Set Identity	134
5.54	Mimics	135
5.55	The action order	136
5.56	The animation command	139
5.57	Disable AI units	144
5.58	SetVelocity	144
5.59	The information text	144
5.60	Units keeps lying or keeps standing	144
5.61	Using ID's	145
5.62	Placing units inside of a building	148
5.63	Unit is moving to desired house position	153

5.64	Getting position displayed	153
5.65	The Eventhandler	155
5.66	Different text displays	157
5.67	Stringtable basic values	158
5.68	Create waypoints	159
5.69	Create trigger	160
5.70	Create marker	162
5.71	All about vehicles	165
5.72	Create a light source	167
5.73	Create dust	167
5.74	Create smoke	168
5.75	Create fire	169
5.76	Assigning ranks	171
5.77	Unit using binoculars	172
5.78	Assigning a unit to a vehicle seat	172
5.79	Allocate a unit to a team	173
5.80	Unit is giving out a command	174
5.81	Has a unit recieved damage?	174
5.82	The air traffic	175
5.83	Decrease grass details	176
5.84	Place sloped objects	176
5.85	Lock or unlock missions	177
5.86	Empty searchlight with light	177

Chapter 6: The Missions Specials

6.1	The paratroopers	179
6.2	The GPS-System	180
6.3	The action menu entry	181
6.4	The backpack	181
6.5	Random positions	185
6.6	The mapclick	187
6.7	The artillery	189
6.8	Deleting killed units and vehicles	194
6.9	Suppressing gaming speed constantly	195
6.10	The bullet mode	196
6.11	Track down enemy units	197
6.12	The air strike	198
6.13	The air vehicle creator	201
6.14	The searchlight	203
6.15	The time counter	204
6.16	The house patrol script	205
6.17	The mine script	208
6.18	The vehicle transport script	209

6.19	The seagull script	213
6.20	The insect script	215
6.21	The saboteur	216
6.22	The spotter	217
6.23	Unit is capitulating itself	218
6.24	The teleport	221
6.25	The persecution script	222

Chapter 7: Multiplayer

7.1	The Multiplayer Mission	224
7.2	The respawn points	224
7.3	Flexible respawn points	225
7.4	The MP-Description.ext	226
7.5	The different ways to respawn	227
7.6	The Deathmatch	227
7.7	Defining the Multiplayer area	228
7.8	Time and Rating	229
7.9	Assigning and displaying scores	231
7.10	Time display	232
7.11	The class header	233
7.12	The respawn dialog	233
7.13	Stringtable MP basic values	234
7.14	The vehicle respawn	235
7.15	Mr-Murray's vehicle respawn	236
7.16	Flag basic informations	238
7.17	Capture the flag	240
7.18	The public variable	246
7.19	Preface information for MP Missions	247
7.20	The controlling commands	249
7.21	The armament within Multiplayer	250
7.22	Text messages for a specific player	251
7.23	Join In Progress (JIP)	252

Chapter 8: Cam Scripting

8.1	Controlling the Camera	255
8.2	The Camera Coordinates	256
8.3	Creating A Camera	257
8.4	The First Scene	258
8.5	Patching the Camera On a Vehicle/Unit	260
8.6	Text and Blending Effects	261
8.7	Camera Effects	262
8.8	Preload Objects and Positions	262

Chapter 9: Scripting

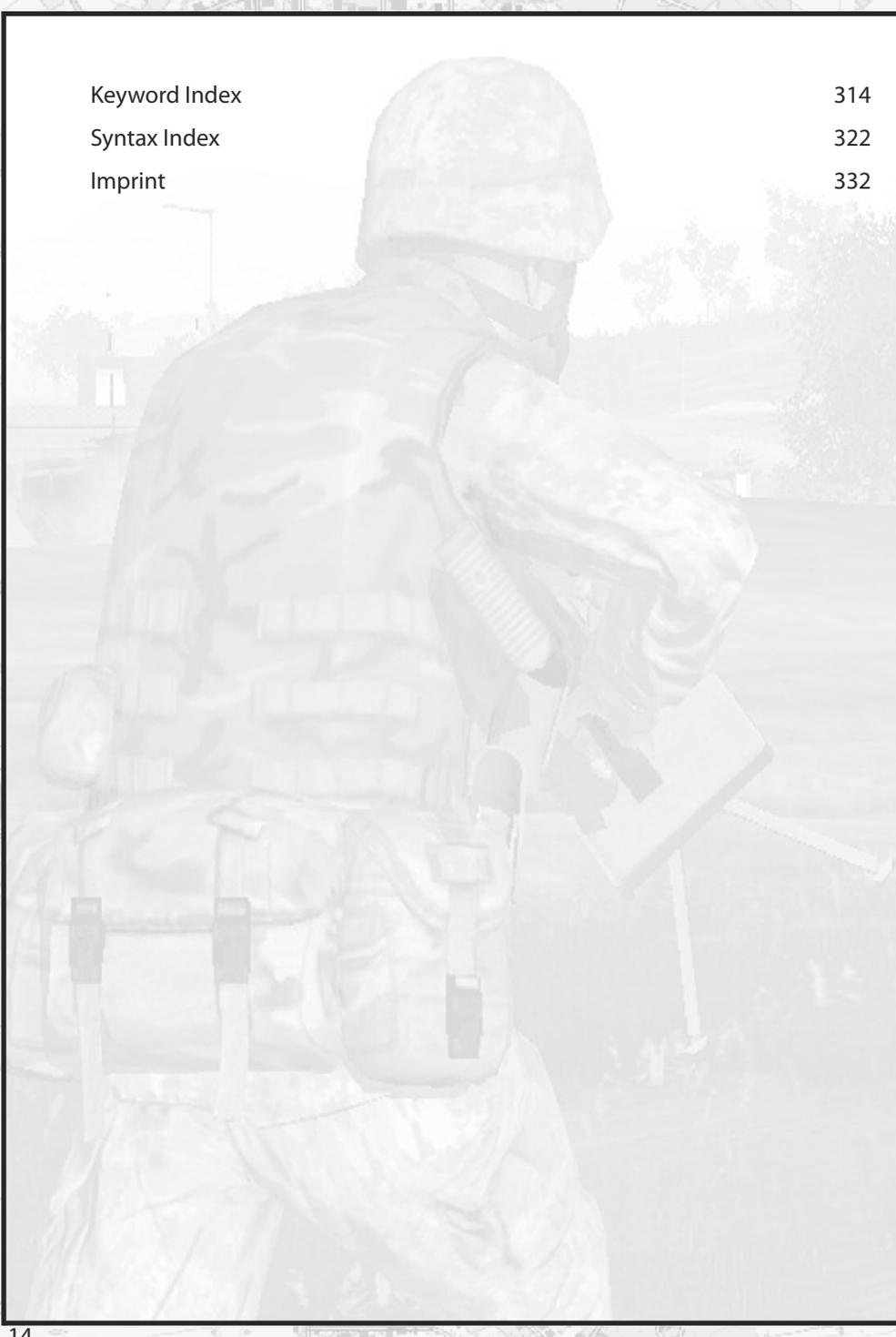
9.1	The variable	265
9.2	Logical values	266
9.3	Logical operators	267
9.4	The While-Do-Loop	268
9.5	The counter	268
9.6	If-Then-Else	268
9.7	The delay	269
9.8	Random	269
9.9	WaitUntil	269
9.10	The brackets	270
9.11	The semicolon	271
9.12	The array	271
9.13	Basic knowledge about functions	274

Chapter 10: Dialogues

10.1	What actually is a dialog?	279
10.2	Base definitions (constants)	280
10.3	Basic classes and subclasses	283
10.4	The font styles	286
10.5	Fading in a graphic	.287
10.6	Fading in a text	288
10.7	Fading scope views	289
10.8	An own tactical map	291
10.9	Defining a button	292
10.10	Defining a Frame	294
10.11	The video sequence	297

Chapter 11: General Informations

11.1	Own profile	299
11.2	The ArmA Cheats	301
11.3	The MOD-Folder	302
11.4	The use of addons	303
11.5	The missions release	304
11.6	The ArmA.rpt	305
11.7	The Nato Alphabet	306
11.8	The ranks and their badges	307
11.9	The Squad.xml	308
11.10	The Start Parameters	311
11.11	Key combinations, tips and tricks	313



Keyword Index

314

Syntax Index

322

Imprint

332

Chapter 1

- The Beginning -

This chapter will serve to provide you with an overview and detailed perspective of the user interface of the editor. It will also get you ready for the following chapters. With the help of this chapter you will receive a detailed explanation of the main functions of the editor to obtain positive results. The main functions of the editor are as follows.

1.1	The user interface	16
1.2	Adding units	20
1.3	Adding groups	26
1.4	Adding triggers	27
1.5	Adding waypoints	30
1.6	Synchronize	35
1.7	Adding markers	36
1.8	Rotating units and objects	39
1.9	Merging units and markers	39
1.10	Edit units with allocated waypoints	40



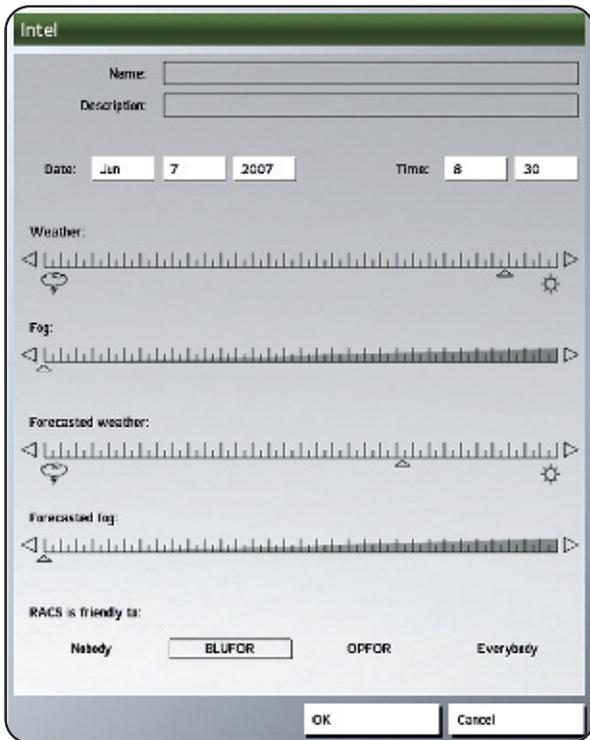
1.1 - The user interface

The user interface of the editor is quite manageable and very user friendly as you can see. You have the possibility to choose and edit the individual areas and sub-menus by using the mouse, arrow keys and F-Keys.

In the Intel box you can define different things like weather, time of day, seasons and on which side the RACS (Resistance) fights. Furthermore there are two input fields at the top of the menu. In the first one you can enter a name for the mission and in the input field below you can define a short description. You can also set the weather at the beginning of the mission and define in which way it will change during the run of the mission. The fog is adjustable, irrespective of the weather. The rain level and the brightness according to the different daytimes will change by adjusting the seasons. The days in the summer are much longer than in the winter, like in real life.



Intel



Name of the mission

Description of the mission

Date and time

Weather forecast

Current weather

Current fog

Forecasted weather

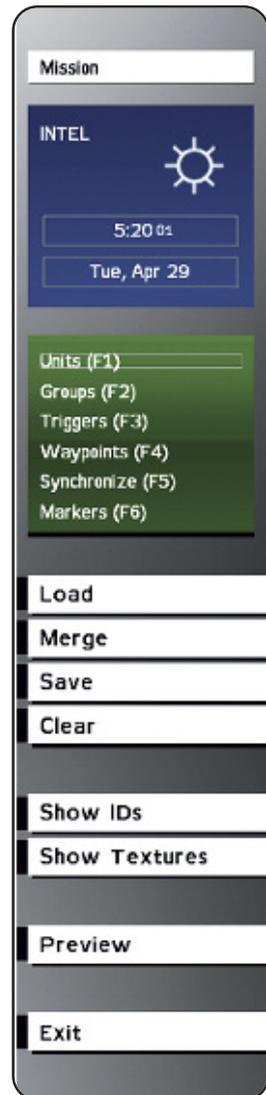
Forecasted fog

Side of the RACS (Resistance)

F-Keys

Pressing the F-Keys (F1 - F6) enables one to enter the sub-menus. This section will give a rough explanation of the different F-Keys, which will be explained individually and more accurately later in this chapter.

- F1** Is needed to place units, vehicles, and objects on the map and adjust them individually.
- F2** Contains two different features. At first it enables you to place entire groups on the map and furthermore it serves as a tool to connect units and triggers with each other.
- F3** Triggers can be placed on the map by using the F3-Key. The Triggers are both powerful and flexible tools, which are needed for a number of different actions. As an example, triggers can be used to define the radio menu.
- F4** The F4-Key creates waypoints that will be followed by groups or individual units. Furthermore they will activate certain actions at predefined places, dependent on the definition.
- F5** Synchronization is a function that can be overlooked quite easily even though it's a useful function. It enables you to synchronize waypoints and triggers with each other. For example, a unit will not move to the next waypoint until the trigger is activated.
- F6** The F6-Key opens the "Markers" sub-menu. The markers provide a tactical view of the map, in order to have a better overview over the mission.



Mission

The first option, called "**Mission**", opens a sub-menu that contains several possibilities to add special features to the mission. Each choice opens a new map. The first one is the mission itself. The second one opens a new map to create the intro, and the third or fourth ones open a new map to create the **Outro - Win**, or the **Outro - Loose**. It's recommended to use new maps for every single choice because it saves performance. And it also enables a much better overview about the main map, because the intro and the outro units are not located on the same map.

There is a further advantage, if the player doesn't like to watch the intro until it ends every time he plays; he just needs to click it away by pressing the space bar. If the intro was produced on the main-map, this would not be possible for him and he would have to see the sequence until the end every time, which can cause him to lose motivation quickly.

Load

With the option "**Load**", the mission will be loaded out of its destination folder in the ArmA® main directory. To do this, the mission needs to be saved into following directory.

C:\Files\ArmA\User\Missions

This folder is empty when the game gets installed for the first time. Every time the player saves a new mission, the game creates a folder with its name in this directory. A final edited mission that has been selected in the ArmA-main menu to play, are not able to load in the editor again. This is because the game converts the mission folder into a PBO-File and so it's not possible to open this mission in the editor again. To do this, a special PBO-Tool is needed.

Merge

Merging is similar to importing. Merging makes it possible to import other missions or units, markers, objects, triggers and waypoints from another map into the current mission. By using the **Merge**-Button everything on the map will get imported, but not the contents of the mission folder.

The merging is quite useful if the player is editing a very complex mission that needs a lot of time to load. If the player wants to add some prefabricated combinations, he just needs to have them saved as their own file – which can be brought into the current scene using the merge function.

Complex sceneries don't need to be rebuilt every time again. This saves on performance and gives some more motivation to the player when he uses several sub-missions as a form of a database.

Save

Pressing the **Save-Button** in the editor menu will save the mission. Here you can decide the method of how you want to save the mission, either as an editor mission or as a final edited Multiplayer or Singleplayer mission. The editor mission will be saved in the directory **C:\Files\ArmA\Missions**. The final edited MP or SP Missions will be saved in the main directory. The Multiplayer missions will be saved in the folder “**MP-Missions**” and the singleplayer missions will be saved in the folder “**Missions**”. You can see an example on the picture below. There you can see the folder missions in your files.



Clear

Pressing the **Clear-Button** clears the map. It will set back into the default status. All things on the map get deleted. Only the missions folder still exists.

Show IDs

By pressing the **Show IDs-Button**, all objects on the map will be displayed. Every single object has a separate ID that it can be contacted with. It enables the user to do different things to an object. You will be able to destroy it or check whether it's still alive or not.

Show Textures

This option enables all textures to be displayed on the map. Every variation, with displayed textures or without, has certain advantages and disadvantages while editing. Finally, every user needs to decide for himself how to best edit a mission.

Preview

By pressing the **Preview-Button** the user can enter the mission to get a first impression of it. He also has the possibility to test several things.

Continue

Pressing the **Continue-Button** enables the user to go back into the last preview. But it only contains the last version of the mission. Current changes are not visible at this time.

Exit

To close the editor and return to the main menu, just click on the **Exit-Button**.

1.2 - Adding units (F1)

The sub-menu of the units is displayed by pressing **F1** or mouse clicking on the button "Units (F1)". To place the unit, just double-click on the map and the unit menu appears. The user has many possibilities to create his favorite unit. It also enables the user place units, vehicles, helicopters, airplanes, objects, and game logic.

Side

East

West

Independent

Civilian

Empty

Logic

Choice of side

East Units

West Units

Resistance Units

Civilian Units

Empty Vehicles

Game Logic

Class

Air

Ammo

Armored

Car

Men

Mines

Objects

Ship

Sounds

Static

Support

Kind of units

Helicopter, Airplane

Weapons, Ammo

Armored Vehicles

Cars, Motorbikes

Soldiers

Mines

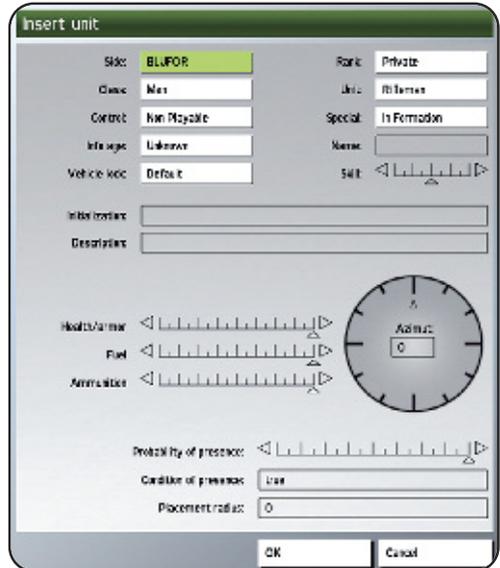
Static Objects

Ships

Sounds

Guns, Machineguns

Support-Trucks



Control - Player or Playable

With this menu the user needs to decide what kind of character he wants to play. Maybe the user wants to play as this unit; or should this unit be playable or not playable (AI).

Playable units are needed while creating multi-player missions, that's important because later in the game, every player must have the ability to make a choice between different units. If the user creates a single player mission with playable units, so they are needed if the gamer wants to use the character switch. The player is able to switch between the units to bring them to different positions.

Player

Playable

Non Playable

The player himself

Playable unit

Not playable unit

The player also has the additional possibility to decide which seat he wants to use in the vehicle and set it immediately.

Player as Driver
Player as Pilot
Player as Gunner

Vehicle lock - Vehicle settings

Default
Locked
Unlocked

To adjust this by an external script or by using Initialization box use the following syntax:

Name lock true - Vehicle locked
Name lock false - Vehicle unlocked

Rank - Rank of the respective unit

The rank of each unit can be set here. The unit with the highest rank will be the leader of the group automatically.

Private
Corporal
Sergeant
Lieutenant
Captain
Major
Colonel

As follows you can see the syntax to set the rank of a unit:

Player setRank "Sergeant "

Unit - Class of unit

After adjusting the units, whether it's a soldier or a vehicle, the specification of the class is possible here. The choice in the sub-menu is always up to the class decision. The user has several choices in the section "**Men**" because he can make a decision between different types of soldiers – from the assault rifle soldier to the grenadier, on up to the sniper; these are only a few of the types available. The same is possible with the vehicles.

Special - Particularities of a unit

This option enables the adjustment of several settings that often get ignored by the user. It also enables the user to start the mission while flying a helicopter or airplane.

- | | |
|---------------------|--|
| None | If the user places a unit on the map and sets the option " None ", then this unit will move to the leader's position even when she's far away from the leader's position. |
| In Formation | If a unit has been placed on the map as a part of a group, with the option " In Formation ", then the game places this unit next to the position of the leader. |
| In Cargo | When the player sets a group on the map, with the option " In Cargo ", and one of its units is a vehicle (this unit must not be the leader) then all units of this group will sit in the vehicle when the mission begins. |
| Flying | All flying units are already in the air (flying) when the mission begins. |

Name - Name of the unit

The name of the unit or the object will be displayed here. This is very important to communicating with this unit while working with scripts, triggers and waypoints.

Skill - Skills of the unit

The abilities of a unit are defined here. This option allows setting the level of skill between **0** and **1**. The level **0** means silly and **1** means very good.

Name1 setSkill 0.8

Name1 setUnitAbility 0.6

It's also possible to set a random skill. To do this, following syntax needs to be defined in the initialization box (Init box) of the unit:

this setSkill (random 0.6)

Now the unit will get a random skill between **0** and **0.6**.

Azimut - Direction of view of an unit

One can adjust a rough direction of view by clicking somewhere in the cycle. So far the direction has been set, so it will be defined as degrees right in the same second. The values of the direction are as same as in real life from **0 – 360** degrees. So far the be wished direction was set, so the unit will be placed on the map with their respective direction.

If the user wants to spin the unit around after pressing **OK** because the units direction is still not the right one, so he can adjust the direction by using the left mouse button and shift-key. To do this, don't double-click on the map to enter the menu, just click the left mouse button to select the respective unit. Then press and hold the Shift-Key. Now move the mouse to change the direction of view.

To spin several units or objects, the same principle is applied. To do this, all units needing direction change need to be selected by the user. To change the direction, the mouse only needs to get moved moved while pressing the left mouse-button and holding shift.

Another possibility to spin a unit in a sequence would be:

Name setDir Value
Name setFormDir Value

The value can be copied out of the sub-menu of "units" by pressing **Ctrl+C**. Then paste it into the script (instead of North).

Initialization - The initialization-line

Every unit and every object has an initialization line. Orders that are located there will be executed by the game immediatly. Scripts can be executed from here or a random skill syntax which can be defined here as well as explained above in Skills. It's generally always recommended to set an additional option called `Init.sqs` in the mission's folder. This script contains the initialization file of the mission. The `Init.sqs` will be executed by the game without a predefined syntax. A closer explanation about the `init.sqs` can be found in Chapter 2.5 – The `Init.sqs`. Make sure that all commands are divided with a `;`

It's also possible to define an entry in the init line of a unit while a mission is running. To do this one only needs to use a `setVehicleInit-order`, and to call it `processInitCommands`.

Player setVehicleInit "Player say 'Sound1'"; processInitCommands;

Description - The information line

Names and descriptions of the unit can be typed into this line. This description will be displayed if the unit is defined as a switchable unit, and the switch menu will open.

Health / Armor - The health status

The status of health and armor of a unit will be set by using the slide control. Injured units can be placed on the map this way. Vehicles can also be used as wrecks if the armor is set to zero. The value can be set between **0** and **1**. **1** is fully damaged and **0** means no damage. There is a further syntax available which will be defined in the **unit submenu**.

Name setdamage 1

The unit, tank, or object would have an amount of damage of **1** and would be dead or destroyed. This value can be reset by using the Syntax:

Name setdamage 0

The unit would be reanimated or repaired again. Of course it's possible to use interim values. By using the value **0.5** the unit would not be repaired completely.

Support:

Support vehicles like fuel, ammunition, and repair trucks can get a value of **0** to **1**. If the value set to **0**, then this vehicle will not to be used again to offer repair, refuel, or ammunition support. The Syntax:

Name setRepairCargo 1

Reassigns a new repair-value to a repair truck.

Fuel - The fuel-status

The quantity of fuel of a vehicle is to set here. It's possible to define it by syntax as well:

Name setfuel 0 - Empty fuel tank

Name setfuel 1 - Fuel tank is full

Support:

Name setFuelCargo 0.3 - Allocates a value of amount of fuel to a vehicle

Ammunition - The ammunition status

Adjust the amount of ammunition that a unit has at the start of a mission.

Support:

Name setAmmoCargo 0.7 - Allocates a value of amount of ammunition to a vehicle.

Probability of presence

This option sets the probability of presence of a unit in percentage. If the slider is moved to the middle, there is a 50% chance that the unit will appear on the map. This makes the mission dynamic, because it's not possible to know whether a unit is displayed on the map or not.

Condition of presence

A unit will only be present on the map when a condition has been executed. This condition is always checked by the game when the mission begins. If **Cadetmode** was set, the unit will be displayed in the **Cadetmode** only.

Placement radius

When the mission begins, the unit will spawn in a random location within this radius.

Info-age - degree of familiarity

Info-age defines the degree of familiarity of a unit. This option tells what the player knows about opposing units and how current this information is. The following selections are available:

**"ACTUAL" "5 MIN" "10 MIN" "15 MIN"
"30 MIN" "60 MIN" "120 MIN" "UNKNOWN"**

It is also possible to define the info-age by using a syntax. This is what it would look like:

Player setTargetAge "10 MIN"



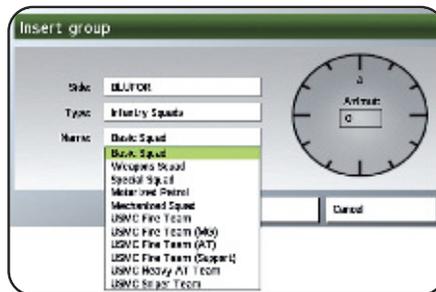
1.3 - Adding groups (F2)

By pressing the (F2) -Key the user will activate the sub-menu groups. It enables the user to place whole groups on the map. After he decides which side to place units for, he can then make a choice between infantry units, tanks or helicopter groups. This possibility saves a lot of time, because not every unit needs to be placed on the map individually. The groups are all predefined, but the user does have the ability to edit the group as he wants. This makes it possible for him to add or remove single units. He can also change the classes of each single unit.

Once you have decided on a side for the units, you have to make some selections. The option "type" offers the type of units, such as infantry, tanks, or helicopter groups. By using this option it is possible to set a whole group quite fast.



At the option **Name**, it's possible to make a choice between 5 different types of groups.



East and West both have 5 different classes of infantry squads available for use.

- | | |
|-------------------------|---------------------------|
| Basic Squad | - Merged infantry squads |
| Weapon Squad | - Smaller Infantry squad |
| Special squad | - Special Forces |
| Motorized Patrol | - Infantry squad with UAC |
| Mechanized Squad | - Infantry squad with APC |

In the same menu as **Infantry Squads**, are **Tank Platoons** and **Helicopter Squadrons**. If an air unit needs to be flying when the mission begins, the user needs to set the formation to **"Flying"**.

The direction of view needs to be adjusted by setting the **Azimut** again.

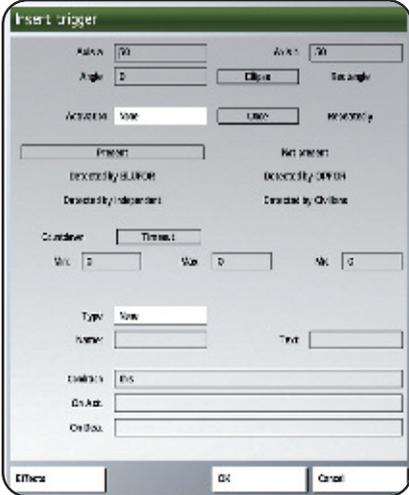
1.4 - Adding triggers (F3)

The trigger can be used as an on/off switch or as a checking-tool. It also enables the user to implement radio menus from Alpha to Juliet. Triggers and waypoints aren't that different from each other. The trigger enables the user to activate or stop certain actions. There is also the possibility to add sounds, music or other resources (such as video clips) by using the sub-menu "Effects" on the trigger menu.

The user has the possibility to activate scripts or similar things when the character has entered trigger area. When the character is leaving this area again, those scripts can be deactivated again.

A trigger can also be used as a ruler when the user wants to place several units next to each other. To do this the axis **A** needs to get the value **100** and the axis **B** needs to get the value **1** for example.

- Axis A /Axis B** Size and area of the trigger
- Angle** Angle of the trigger
- Ellipse/Rectangle** Form of the surface
- Once/Repeatedly** One-or multiple-activations
- Activation by**
 - West
 - East
 - Resistance
 - Civillian
 - Gamelogic
 - Anybody
 - Radio A-J
 - Captured by West
 - Captured by East
 - Captured by Resistance



The way of activation

- Present** A trigger will activate its actions when a unit is entering this area. An example: (Unit=East Activation=East).
- Not present** All actions will be disabled if this east unit is leaving the trigger again.
- Detected by** Detected by west, east or civilians. Trigger will be activated if the unit of the defined side is detected within the area of the trigger.

Countdown/Timeout - Counter

Within this menu the user can set the time distance between a unit entering an activation area, and the activation of that trigger. The possibility to add a minimum, middle, and a maximum value to the timeout function can make the mission much more dynamic. When all three values are defined, the activation of the trigger will happen at a random time between the set values. If the minimum, middle and maximum values are all set to the same value, the trigger will activate when the timer runs out.

Type:

None
Guarded by West
Guarded by East
Guarded by Independent
Switch
End 1 till End 6
Lose

Text - The trigger text

The user can enter a specific description of the trigger. The user needs to write the description into the text box. This possibility enables better organization on the map, especially if several triggers were used. The user mustn't open every single trigger to get the information about what each trigger does. Furthermore, the text for each radio message is to be defined here. If the user creates more radio messages, then he can see the messages on the radio individually.

For example:

Trigger 1:	Activation:	Radio Alpha
	Text:	Request Artillery
Trigger 2:	Activation:	Radio Bravo
	Text:	Request Support

Both of these lines of text would be displayed on the radio now. This enables a better overview and orientation.

Name - The trigger name

To communicate with the triggers the user has to enter a name into the text box next to "Name" - for example, if the user wants to move or delete it later. You can get more information about moving and deleting triggers on the following page.

Condition - Activation condition

A condition is a powerful tool, which enables the trigger to check several things. The trigger would be activated only if a condition was executed. A condition can be used in many ways. It's also possible to use both variants: **Use of a variable** or **checking of a condition**.

Use of a variable

The trigger will be activated if a condition was executed first. If attack would be placed as a variable into the **OnActivation**-Box, the trigger is waiting until this variable has been set to **true**, to activate itself.

To set the variable "**Attack**" to **true**, it has to be defined in a trigger, waypoint or a script. To do this following Syntax is needed:

Attack=true

To activate the syntax, "**Attack**" has to be set to "**true**" in the **OnActivation**-Box of the trigger, waypoint, or written in a script. The action defined will be activated now.

Verifying a condition

Another possibility to check whether a condition was executed or not. A variable is a condition that needs to be checked and executed as well. This means, that the condition is checking the actions of a unit. For example: Is unit **A** still alive, is unit **B** sitting inside **Jeep1**. That would look like this:

Condition: Player is sitting in Jeep1:

Player in Jeep1 or **Vehicle Player == Jeep1**

Condition : Soldier1 not alive:

Not alive Soldier1 or **!(alive Soldier1)**

On Activation

Nearly all actions that can be activated when the trigger gets executed are to be defined here in this box (for example, the starting of a script or setting a condition on true etc.). The user can enter nearly all Syntaxes that are available in ArmA®. But this option does have its borders as well, and so sometimes it is better to use scripts. The use of scripts helps the user stay more organized than having triggers all over the map.

On Deactivation

It's also possible to activate a trigger when it actually gets deactivated. The following example serves as an example of an entry in the action menu, as variant to explain what it means. If the player enters a trigger area, then a further option, called "entry", will be added to the action menu. This entry will be deleted when the player leaves the trigger. To do this just enter:

```
on Activation:      ID=Player addAction ["Entry","script.sqs"]
on Deactivation:   Player removeAction ID
```

Move or delete triggers

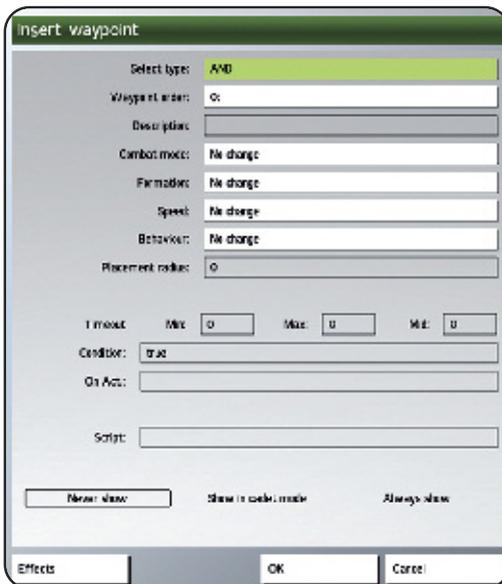
When it's needed, the user can move or delete a trigger. But to make sure that it works, the trigger has to be equipped with a name first. That's important to speak to the trigger directly. The following syntax is needed:

```
TriggerName setPos getPos Name
```

The user can otherwise use coordinates. That syntax would look like this:

```
TriggerName setPos [x,y,z]
```

1.5 - Adding waypoints (F4)



Waypoints are not only needed to set a route where a unit shall move, the user can also use a waypoint to set behavior, formation, Combat mode and speed.

The waypoint can be similar to a trigger in function.

When the unit reaches it's next waypoint, code can be activated and executed upon arrival.

But there are a lot more possibilities, such as having sound effects or music played at a predefined waypoint.

Select type - The actions

This is where you can select the different types of waypoint you wish to use for a unit on the map.

Move
Destroy
Get in
Seek and destroy
Join
Join and lead
Get out
Cycle
Load
Unload
Transport unload
Hold
Sentry
Guard
Talk
Scripted
Support
Get in next
Released

Clarify particularity

If the user allocates a waypoint to a unit with these conditions, then the unit will move to this point and wait for enemy contact before she's moving on to the next one.

Waypoint order

The player can get an overview about all waypoints by clicking this option. It's also possible to change the sequence of waypoints in retrospect.

Description

This is the description of a waypoint. Descriptions will be shown in the **Cadetmode** only. It makes playing for beginners easier, especially playing huge and complex missions.

For example: **Destroy the target**

Combat mode

The combat mode of a unit is defined here:

Never fire	Blue
Hold fire	Green
Hold fire, engage at will	White
Open fire	Yellow
Open fire, engage at will	Red

To define these behaviors by script use following Syntax:

Name setCombatMode "Blue"

Behavior

To define the behaviour of a single unit or a whole group, the following options are available:

Careless
Safe
Aware
Combat
Stealth

This behaviour is definable by script as well. The syntax is:

Name setBehaviour "Careless"

Formation

Special formations are needed on the battlefield while fighting in special situations. These formations will be shown in the examples below. The group leader is always marked in green.

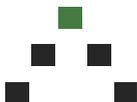
Column



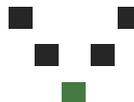
Staggered Column

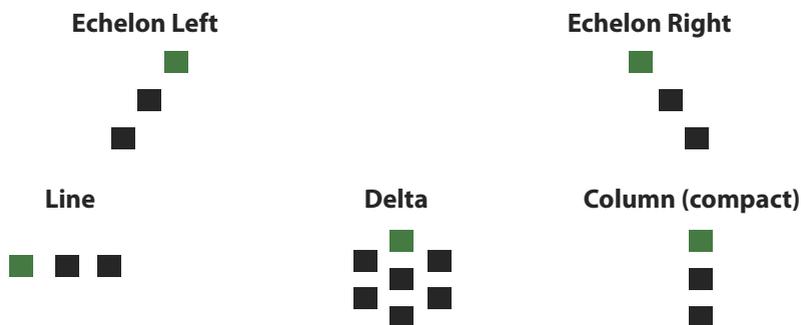


Wedge



Vee





It's also possible to set the formation of a unit by script or trigger. The syntax will be:

Name SetFormation "Line"

Speed

The speed of a single unit will be defined individually at every waypoint. It's possible set a unit to move fast to waypoint 1 and move slow to waypoint 2. There is a choice between 3 variants.

Limited / Normal / Full

And we have the possibility to define this in a script as well. The syntax is:

Name setSpeedMode "Limited"

Placement radius

The placement radius of a waypoint offers more dynamics to the game, because the waypoint will be set at random to a location within this radius. This means, that when the user enters the value 100 for example, the game will set this waypoint within a radius of 100 meters.

Timeout

The delay until the trigger executes its actions, in seconds. If the minimum, middle and maximum values are all set to the same value, the trigger will activate when the timer runs out. But if the values are different from each other then the trigger will activate its actions randomly.

Min	Minimum time until activation
Max	Maximum time until activation
Mid	Middle value

The use of random values adds a lot of excitement to the game, because the user doesn't know when the unit will reach its destination. The user has a much more dynamic mission by combining **placement radius** and **condition of presence**, because the user will no longer be able to say whether the unit exists or not, or which place she's moving to and finally when will she will arrive. Even dynamic missions have a high rate of suspense and this makes the missions much more replayable. ArMA® has the best conditions to create missions as dynamically as possible.

Condition

The use of a condition allows a waypoint to change its status depending on whether the condition is true or not. Using a condition makes it possible to set a waypoint to "stand by" or to let it check something. That waypoint would be activated when the condition is true, but if the condition has not been met, the waypoint will remain inactive. The use of a condition has been explained in sub-section **Chapter 1.4 – Adding Triggers**.

On Activation

In this line it's possible to define everything that shall be executed when the unit has reached a waypoint (e.g. starting scripts or setting variables to true etc.) It enables the user to enter every syntax that is compatible with ArMA®. It has its borders as well, so the user should use scripts sometimes. Scripts just help keep the mission editor clean by keeping the code in external files.

Script

This line enables the user to use code, which would only be used in scripts.

Show Waypoint

It's possible to show or hide waypoints. The user can define here whether they are only visible in Cadet Mode, general, or not visible.

Never Show

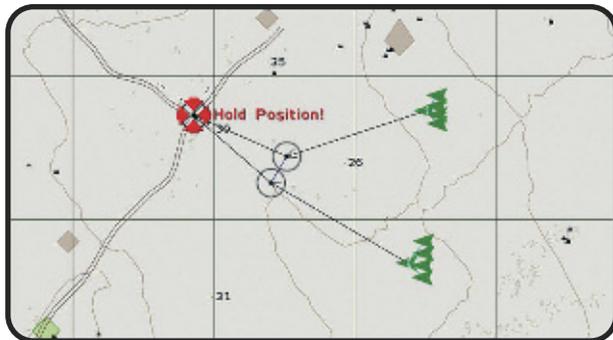
Show in Cadet Mode

Always Show

Join and lead

It's possible to bring several groups together. This is possible at any point on the map as well. Every single unit gets its own waypoint somewhere on the map. The first one should be defined as **Join** and the other one as **Join And Lead**. Both waypoints only need synchronization now. The user has to press the (F5)-Key to do this. Now both waypoints get synchronized with each other. If both units reach their destinations and are still alive,

they will come together as one group now. It will only work if the groups are not too big. ArMA® allows up to **144 units** to a group including their leaders.



1.6 - Synchronize (F5)

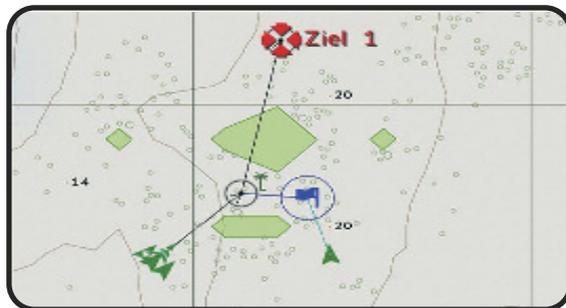
The option **“Synchronize”** offers the possibility to connect waypoints with waypoints or waypoints with triggers. It enables the units to remain in the same formations and the user does not have to use too many variables.

You can see 2 units in the picture below. Both units are coming from different directions but they have the same target. Unit 1 has to wait at its waypoint until unit 2 reaches its waypoint. To do that, the user has to press the **(F5)**-Key to choose synchronize mode. Then, the user needs to click and hold on the waypoint 1 by using the left mouse button; pulling the mouse from waypoint 1 to waypoint 2 creates a blue line. It's important to hold the left mouse button while doing that. When finished, a blue line connects both waypoints with each other. When unit 1 arrives at their waypoint, they will wait there until unit 2 arrives at their destination.

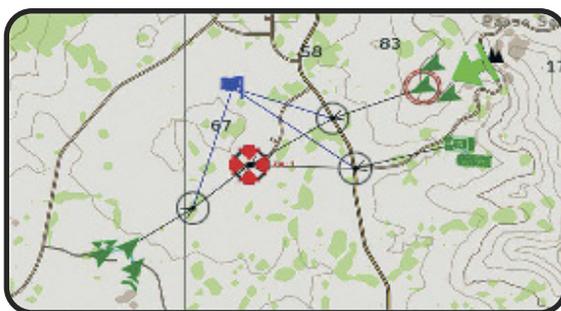


The trigger-waypoint combination works the same way. The groups will only move on when the trigger is executed. It's not necessary to allocate a variable to the unit at the waypoint (like **Grp1go** for example). Enter the command **Grp1go=true** into the OnActivation box of the trigger so that the group runs forward if the trigger is executed.

It doesn't make any difference whether the trigger is executed by an object or radio. The **(F5)**-Variant is much faster and easier to handle.

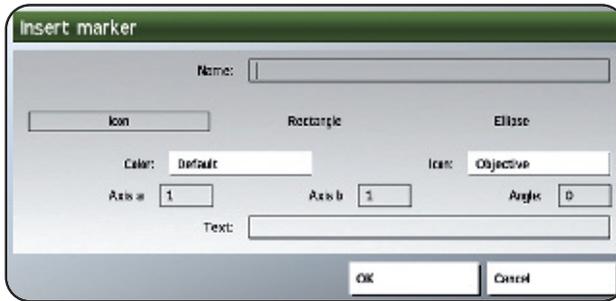


Three groups and a radio trigger will get synchronized with each other in the picture below. If all 3 groups have reached their positions the user only has to give the order to attack by using **Radio 0-0-1**. The units will move to the next position, the target.



1.7 - Adding markers **(F6)**

The markers are necessary to create a tactical view on the map. The markers show the player the course of the mission, the targets, and more information which makes the mission more interesting. It enables the user to get a better overview over the whole mission. It's necessary to give a name to each marker because those markers can be linked into the briefing with their positions on the map. If the player clicks on a link in the briefing the crosshair will move to the position of the marker on the map.



Name

The name of the marker will be entered in this box, which will be needed later for the link with the briefing (to move the crosshair over the map by clicking on a link in the briefing text). It's also needed to move markers from one position to another or change them to another symbol.

The kinds of markers

There are 3 different ways to set a marker on the map. The first one is the **single icon**. The second one is a **rectangle** and the last one is a **circle**. With the last two choices it's possible to mark whole enemy or friendly areas.

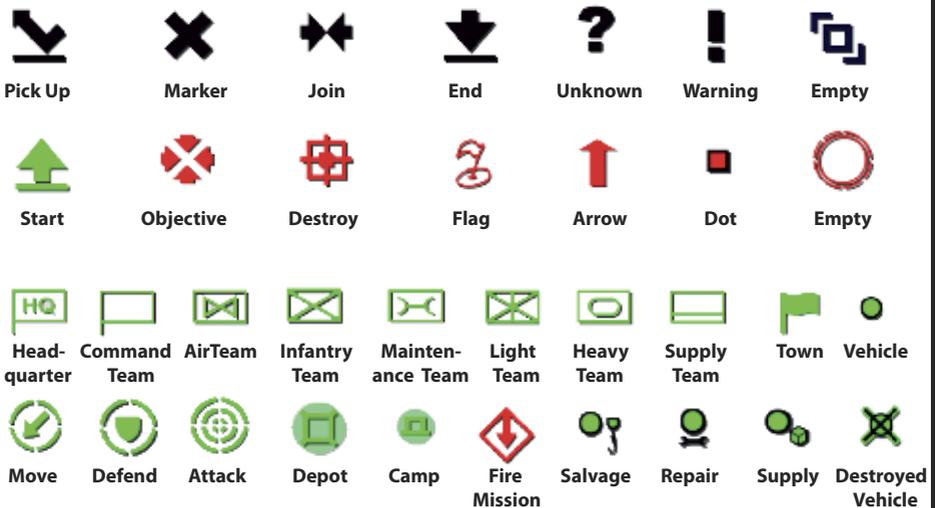
Color

Markers can be several colors. The following colors are available:

Red, black, green, blue, yellow and white

Symbol

Here is an overview of the different tactical signs with their descriptions:



It's possible to create a much more complex map by combining area markers (Ellipse/Rectangle) with tactical signs. Here you can see a list with all markers which are selectable by default in the game. The following marker names are to be used as class names as well with the exception of Objective. The class name of **Objective** was defined as **Flag** by BIS.

Objective (Flag)	Arrow	SalvageVehicle
Flag1	Empty	RepairVehicle
Dot	Select	SupplyVehicle
Destroy	Vehicle	DestroyedVehicle
Start	Defend	MaintenanceTeam
End	Move	CommandTeam
Warning	Attack	SupplyTeam
Join	Headquarters	InfantryTeam
PickUp	Depot	LightTeam
Unknown	Camp	HeavyTeam
Marker	Town	AirTeam
		FireMission

Axis A/Axis B

The user can define the size of the marker here.

Angle

The user can define the angle of the marker here.

Text

To make the description of the marker visible on the map, the text needs to be entered in this box. For example: **Target Alpha**. It's possible to set the marker as the name of the player or the unit by using following syntax:

"S1M" setMarkerText Name S1

The game automatically selects the player name of the unit that is named **S1**.

Dynamic Positions

If you place several markers on the map and connect them with **F2** (see **Chapter 1.9**) with a unit, so this unit will always be spawned on another position each time when the mission gets started.

Move or delete a marker

It's possible to make several changes to the markers. The user can move, delete or change the symbol while the game is running. The execution of a missions target shall serve as an example here. So it would be possible to delete the marker or change the color of the respective marker from red to green.

A marker needs to be named first. To explain the following syntax examples, the marker will get the name **Marker1**. Now there are lots of possibilities to communicate with the marker. One can do this by using waypoints, triggers or even scripts. To do this, the following syntaxes can be used:

Marker will set to position [x,y]:

```
"Marker1" setMarkerPos [x,y]
```

Marker will set to position from **Name**:

```
"Marker1" setMarkerPos getPos Name
```

Marker will be set to position of **Marker2**:

```
"Marker1" setMarkerPos getMarkerPos "Marker2"
```

Defines kind and style of the marker:

```
"Marker1" setMarkerType "Start"
```

Changing the color of the marker:

```
"Marker1" setMarkerColor "ColorBlue"
```

Changing the size of the marker in [height, width]:

```
"Marker1" setMarkerSize [2,4]
```

Deleting the marker:

```
deleteMarker "Marker1"
```

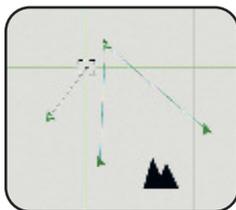
1.8 - Rotating units and objects

To twist whole groups, objects or single units it's necessary to select them first. Then the user needs to press the **Shift-Key**, click and hold the **Left Mouse Button** while moving the crosshair by using the mouse until the object is turned towards the desired direction. It's possible to turn more than one object, unit or group by selecting all units on the map that need to be turned.

1.9 - Merging units and markers

In the editor it's possible to bring single units to whole groups together. It's also possible to divide them from each other again. To do this the user has to select the section "Groups" by pressing the F2-Key. Then the units need to be selected by clicking and holding the left mouse button while moving the mouse. The same method is used to divide a group into its single units. The user only has to click into an empty space on the map.

Furthermore, it's possible to combine a unit or an object with a trigger to make sure that this trigger will only be executed by this unit or object. In the left hand picture you can see how a unit will be combined with a group, and on the right picture one can see that a trigger was placed on the map which was used by a vehicle.



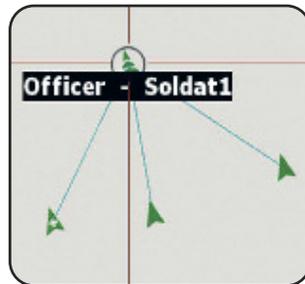
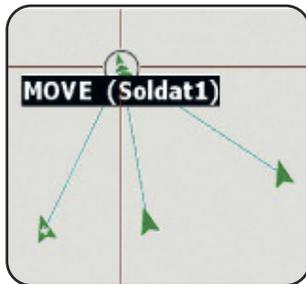
This is possible now with units and markers. The special thing in this case is that one can now assign several random positions to the unit and the object. If the unit is connected now with all 3 markers, the unit will get respawned on one of these markers each time when the mission begins. One doesn't know where he will be respawned. This option will save a script and make a mission much more dynamic.



1.10 - Edit units with allocated waypoints

If a unit already has a waypoint allocated to it, then the user will notice that the waypoint menu always opens after double-click, but not the unit menu.

To edit the unit after it already has a waypoint allocated, the user has to press the **Shift-Key** while double-clicking on the unit. Then the unit menu will appear.



Chapter 2

- The Files -

After being introduced to the user interface in Chapter 1, this chapter will lead you another level down into the system of the game. I will explain to you the individual files which are important when creating a mission. Important information will be saved and configured here.

2.1	The missions folder	42
2.2	The Mission.sqm	43
2.3	The Description.ext	48
2.4	The Stringtable.csv	51
2.5	The Init.sqs	53
2.6	The Script (.sqs)	54
2.7	The Function (.sqf)	54
2.8	The Paa-Format	55
2.9	The PBO	56
2.10	The sound files	56
2.11	The Lip-Files	57
2.12	The Overview	58
2.13	The Briefing	59



2.1 - The missions folder

The missions folder contains all important files which are needed for a mission. It's necessary to create more folders to stay organized. So, it is recommended to create one folder for every file-type individually. For example: music, scripts, sceneries, pictures and so on. It's also important to write the data types in lowercase letters. On the picture below you can see a final edited missions folder:



As one can see in the example above, this folder contains all important files which are needed for a mission. Furthermore you will have a better overview over your files. The filenames in this example are all predefined and not variable! The title image and the folder names are the only ones which can be named differently from the other files. The file-types are explained here:

Mission.sqm	Mission coordinates
Description.ext	Mission configurations (Music, Sound, Weapons, Identity, Resources etc.)
Stringtable.csv	Enables the user to display text by a shortcut only
Briefing.html	Contains the briefing text
Overview.html	Mission info in missions selection menu
Title.paa	Overview image

Different briefing files are needed to display the briefings in several languages. But if one wants to create his mission in one language only, he only has to use the briefing.html. It doesn't matter in which language the briefing file is defined. You can get more information in this chapter, in the sub area **The Briefing.html**. It's up to the player how the following folders will be named

Music	For music files
Sound	For sound files (e.g. Languages, Effects)
Scripts	For scripts (.sqf)
Scenes	For the cut-scenes (e.g. Intro, Outro etc..)
Function	For the functions (.sqf)
Pictures	For the images (e.g. Title.paa)

If one wants to open a file out of the sub-folders, the user only needs to define the respective folder in the syntax, backslash and then the respective filename.
E.g. to run a script:

```
[ ] exec "scripts\script.sqf"
```

2.2 - The Mission.sqm

The Mission.sqm contains all important coordinates which display the objects, units, triggers, waypoints and markers on the map, so it is the most important file in the mission. Furthermore there are several pieces of information located at the top of the Mission.sqm file such as additional add-ons or the mission name, weather and the time of day which are defined in the intel box in the editor.

These explanations might be a little difficult to understand first, but they shouldn't deter one from editing his or her own missions. The player does not have to know every single part of the Mission.sqm, but it's quite useful to know what the options represent.

The first part

On the following page one can see the beginning area of the Mission.sqm in the section "**Class Mission**" which is located in the top of the mission.sqm script. The used add-ons are listed first. These add-ons are original ArmA-Addons.

Attention! When the mission begins, if an external add-on is loaded which has an improperly configured **Config.cpp**, it might happen that this add-on will be registered in this list although it will not be used in the mission. But it's possible to mark and delete it (The Mission.sqm needs to be opened with the Notepad text editor).

The problem begins if another player wants to play this mission and hasn't installed this external add-on. The mission will not be able to start, and the mission download was for nothing. This problem happened many times in Operation Flashpoint®. The player who does not know much about editing will get frustrated quite fast and will download another mission.

Following the **Addons** is the **class Intel** which also contains:

Briefing name
Resistance settings
Starting weather
Forecasted weather
Forecasted fog
Distance of view
Date
Time of day

```
addOns[]=
{
    "cacharacters",
    "sara",
};
addOnsAuto[]=
{
    "cacharacters",
    "sara"
};
randomSeed=8635907;
class Intel
{
    briefingName="@STR_M11_Name";
    resistanceWest=0.000000;
    startWeather=0.000000;
    forecastWeather=0.000000;
    forecastFog=0.375187;
    viewDistance=1000.000000;
    month=6;
    day=2;
    hour=3;
    minute=50;
};
```

Next to the **Class Mission** are the **Class Intro**, **Class OutroWin** and **Class OutroLose**, which are built same as **Class Mission**. The units, waypoints and so on for the respective sequences are defined there.

It's possible to make changes directly in the mission.sqm, but its quite important to make changes correctly. If the player wants to test the mission later in the editor, it needs to be re-loaded to use the updated mission.sqm

If the game is crashing down, there might be a mistake in the script. It is quite useful to make a backup of the original Mission.sqm to make sure that a working version is still available.

Units- and object classes

The sub areas **Class Groups** and **Class Vehicles** are located in the main **Class Mission**. In these sub areas, the related units, objects and waypoints are defined:

```
class Groups
{
    items=22;
    class Item0
    {
        side="WEST";
        class Vehicles
        {
            items=1;
            class Item0
            {
                position[]={7973.895020,4.460081,9351.659180};
                id=0;
                side="WEST";
                vehicle="SoldierWB";
                player="PLAYER COMMANDER";
                leader=1;
                rank="CORPORAL";
                skill=0.200000;
                text="aP";
                init="this addWeapon ""binocular"";
            };
        };
    };
};
```

As you can see all information for the unit named **S1** is defined here. You can see the elucidation of the concept here:

- Items=1** Display the numbers of items of the **Class Groups**.
Number of the total groups of all sides of a map.
- Class Item0** Is the group **0**, or the first group. The next group would be named **Class Item 1**
- Side** The side of the respective group. Even a single unit will be defined as a group!
- Class vehicles** Explains to the user that it is a vehicle
- Items=1** The number of items (units) of the group **Class Item0**

Class Item0	Class Item0 is leader of the group Class Item0 . The subordinated soldier to the leader is Class Item1 , the next one, Class Item2 and so on.
Presence	Probability of presence (Not the player!)
Position	XYZ-Coordinates of the player in order X ZY
Azimuth	Line of vision of the unit (definable value from 0 to 360)
ID	ID of the unit
Side	Respective side
Vehicle	The type of the unit
Player	Himself
Leader	Says whether the unit is leader
Skill	Ability of the unit (definable value from 0 to 1)
Health	Health status (definable value from 0 to 1)
Ammo	Ammunition status (definable value from 0 to 1)
Text	Name of the unit (Variable)
Init	The init-line of the unit (needs a syntax to execute)

Waypoint classes

The waypoints are organized into their respective groups. This class is similar to the units, but different in composition.

```
class Waypoints
{
    items=1;
    class Item0
    {
        position[]={7970.289551,4.731988,9346.483398};
        placement=50.000000;
        CombatMode="RED";
        Speed="FULL";
        combat="COMBAT";
        description="Hold this position!";
        expActiv="" exec""scripts\script.sqs";
        class Effects
        {
            timeoutMin=10.000000;
            timeoutMid=3.000000;
            timeoutMax=30.000000;
        };
        showWP="NEVER";
    };
};
```

As one can see, all of the information for each waypoint is defined here, so each waypoint looks different from another. It is up to the player how to define them. Explanation:

Items=1	Displays the numbers of items of the Class Waypoints, this is the number of waypoints in the group.
Class Item0	Class Item0 is the first waypoint. The second waypoint would be named Class Item1 , the next would be Class Item2 aso.

Position	Coordinates of the waypoint, in order XZY.
Placement	Is the random positioning-radius of the waypoint.
CombatMode	The respective fight-mode of the group of this way-point.
Formation	The respective formation of the group of this waypoint.
Speed	The speed of the group of this waypoint.
Combat	The respective behavior of the group at this waypoint.
Description	The description will be displayed when the waypoint is shown in-game.
ExpActiv	The On Activation field, which will be executed when the trigger is activated. In this example, a script with the name script.sqs will be executed from here.
TimeOutMin	The minimum time to execute the waypoint.
TimeOutMid	The middle time to execute the waypoint.
TimeOutMax	The maximum time to execute the waypoint.
ShowWP	Explains whether the waypoint will be displayed in the game or not

No effects have been defined at this waypoint, these effects are explained in the trigger classes.

Marker Classes

Their classes of the markers are located right behind the group and there respective waypoints. All markers set on the map will be listed here. Here is an example:

```
class Markers
{
    items=28;
    class Item0
    {
        position[]={2452.061035,0.760200,3673.595703};
        name="TargetOne";
        text="Objective Alpha";
        type="Flag";
        a=2.000000
        b=2.000000
        angle=0.100000
    };
};
```

The explanation of the points is shown here individually:

Items = 1	Displays the number of items of the class marker. Therefore the entire number of triggers on the map.
Class Item0	Class Item0 is the first trigger. The second trigger would be called Class Item1 , and the very next trigger would be called Class Item2 and so on.
Position	XYZ-Coordinates of the markers in the order of XYZ.

Name	The name of the marker.
Text	The description of the marker which will be displayed later on the map.
Type	The type of the Marker. In this example, a cross-hair is displayed.
a	The size of the marker in the X - direction.
b	The size of the marker in Y - direction.
Angle	The angle of the marker.

Trigger Classes

The class markers are actually right behind the group and object classes. All markers which have been placed on the map will be displayed within this position of the script. One can see an example below:

```
class Sensors
{
    items=53;
    class Item0
    {
        position[]={8012.703613,6.300000,9301.049805};
        a=100.000000;
        b=100.000000;
        activationBy="WEST";
        timeoutMin=10.000000;
        timeoutMid=3.000000;
        timeoutMax=30.000000;
        age="UNKNOWN";
        name="DetectorOne";
        expCond="Var1";
        expActiv=""[] exec ""scripts\script.sqs"";
        expDesactiv=""[] exec ""scripts\animation-end.sqs"";
    };
};
```

Items=1	Displays the number of items of the Class Sensors , therefore the whole number of the triggers on the map.
Class Item0	Class Item0 is the first trigger. The second one would be named Class Item1 and the very next - Class Item2 and so on.
a	The size of the trigger in X -direction.
b	The size of the trigger in Y -direction.
ActivationBy	Activation by " WEST ".
TimeOutMin	The minimum time to execute the trigger.
TimeOutMid	The middle time to execute the trigger
TimeOutMax	The maximum time to execute the trigger.
Age	Unknown
Name	The name of the trigger.
ExpCond	The condition of the trigger. E.g. the Variable Var1
ExpActiv	The activation field of the trigger, which will be activated when the trigger is executed. In this example, a script named Script.sqs will be activated here.
ExpDesactiv	The deactivation field of the trigger. The trigger can be deactivated here again. In this example, a script named animation-end.sqs will be activated here.

2.3 - The Description.ext

The Description.ext is as important as the Mission.sqm for our mission. The specifications of units and objects are not defined here, but the description will provide a lot of other helpful information. It is important to define important things such as music, sounds, respawn resources, weapons selectable from the briefing, accessories like the compass, and several other things which are needed in the game.

The Description.ext has to be placed in the missions folder of the respective mission. To do this one needs to open a text file and rename it Description.ext. It's quite important to edit this file with Notepad (Text File Editor) or **Notepad++** only. Never use Word or Excell!

The Description.ext will only be explained roughly. If you want to know more about special possibilities of the Description.ext, just use the explanations in the different chapters where these sub points are more thoroughly defined.

Mission Start Text	Chapter 4.2
Distribution of points	Chapter 4.4
Identities	Chapter 5.53
Music	Chapter 5.52
Sound	Chapter 5.52
Respawn	Chapter 7.2
Weapon selection in the briefing	Chapter 3.6

It's not necessary to implement all of these possibilities in the file, but this is up to the mission. One has to use the ones which are needed for the mission, this saves not only a lot of work, but it also enables one to avoid errors in the mission. It's quite unnecessary to use respawn in a single player mission for example.

Furthermore it is very important to make sure that all clasps { which were opened are closed again }, otherwise ArmA® will crash promptly. There are other mistakes that will make the game crash as well, so it's quite important to work carefully.

To hide or make available additional mission accessories like the compass or the watch, one only needs to do this by the parameter **1**, or **true**, for **active/visible** or by using the parameter **0**, or **false**, for **inactive / invisible**.

One has the possibility to define special comments behind **"/"** or a **semicolon**. ArmA® will ignore these marks. These marks are used to create special explanations inside a script to make some things more understandable or to keep a script organized.

If changes were made in a script or the description which one has created, one needs to save, and then restart the mission before the changes will take effect. If the game crashes don't lose your patience, this just requires more troubleshooting. It's quite necessary to define every paragraph individually, so one always knows which paragraph might be the one which contains the error.

In the picture below, one can see how to define a **Description.ext**:

```
// ===== Description.ext =====>
Debriefing = 1;
OnLoadIntro = 1;
OnLoadIntroTime = 1;
OnLoadMissionTime = 1;
Saving = 0;
// == Titlecut =====>
OnLoadIntro=BISTUDIO proudly presents
onLoadMission= ARMED ASSAULT
// == Points =====>
minScore=200
avgScore=2500
maxScore=6000
// == Missions Accessories =====>
ShowCompass = 1;
ShowMap = 1;
ShowGPS = 1;
ShowWatch = 1;
// == Respawn =====>
respawn=3;
respawn_delay=10;
// == Weapons =====>
class Weapons
{
    class M4
    {
        count = 4;
    };
    class Javelin
    {
        count = 2;
    };
};
// == Magazines =====>
class Magazines
{
    class 20Rnd_556x45_Stanag
    {
        count = 10;
    };
    class Javelin
    {
        count = 6;
    };
};
// == Music =====>
class CfgMusic
{tracks[]= {Track1,Track2};
    class Track1
    {
        name = "Track1";
        sound[] = {music\track1.ogg, db+0, 1.0};
    };
    class Track2
    {
        name = "Track2";
        sound[] = {music\track2.ogg, db+0, 1.0};
    };
};
```

```

//=== Sounds =====>
class CfgSounds
{
    sounds[] = {Sound1};
    class Sound1
    {
        name = "Sound1";
        sound[] = {\sounds\sound1.ogg, db+0, 1.0};
    };
};

//=== Radio =====>
class CfgRadio
{
    sounds[] = {};
};

//=== Environment =====>
class CfgSFX
{
    sounds[] = {};
};
class CfgEnvSounds
{
    sounds[] = {};
};

//=== Identities =====>
class CfgIdentities
{
    class MrMurray
    {
        name = "MrMurray";
        face = "Face33";
        glasses = "none";
        speaker = "Dan";
        pitch = 1.00;
    };

    class Memphisbelle
    {
        name = "Memphisbelle";
        face = "Face10";
        glasses = "none";
        speaker = "Howard";
        pitch = 1.00;
    };

    class Dan
    {
        name = "Dan";
        face = "Face22";
        glasses = "none";
        speaker = "Russell";
        pitch = 1.00;
    };
};

// End Of File

```

2.4 - The Stringtable.csv

The Stringtable.csv is needed by the game to display different text variables which were defined by the user. It enables the player to define one or several languages in the mission. This file always should be used by the text editor but Windows always tries to use Excel as default program.

If one wants to edit the Stringtable.csv, the user has to take care for several things. The head needs to be defined first. The head also contains the used languages. It's very important to make sure that the different languages are separated by commas individually.

LANGUAGE,English,German,Czech,Notes

The single languages will be separated by commas and marked with " " in every line individually. You can see an example below:

STR_Titel,"Night Patrol", "Nacht Patrouille", "...", MissionName

One can see a very good example here. The address at the beginning is defined with STR_Title, the languages are following, and at the end of the line is a description which will explain what the current line represents. The Syntax "STR_" is the very first part one has to write. The word Title behind is only a variable which can be freely defined by the user. You can see an example here:

STR_Mission_1,"Hold Position!", "Position halten!", "...", MissionText1

If one wants to make the text displayed at the beginning of the mission the following Syntax is needed:

onLoadMission=\$STR_Title;

The Syntax **STR_Title** can be used as an individual address, which has to be entered if one wants to implement text into the mission. That text would then be displayed in the selected and predefined language. The sign @ in front of the Syntax **STR_** is used in the editor only while editing a trigger or waypoint, but in the config or in the description.ext, the sign \$ has to be used.

Calling text out of the editor:

@STR_Title

Out of the description or the config:

\$STR_Title

Below, one can see an example of a waypoint displayed with text. This text has previously been defined in the stringtable:

STR_Mission_1,"Hold Position!", "Position halten!", "...",MissionText1

The following syntax has been written in the description box of the waypoint:

@STR_Mission_1

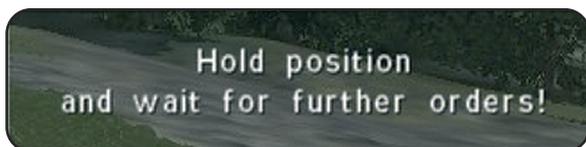
As one can see the text will be displayed in the game:



It's also possible to create a line-change in the text. This gives a better look to the text. To make it, just enter `\n`

STR_Mission_1,"Hold Position\nand wait for orders!""",,..,Missiontext1

In the picture below one can see an example:



In the example below, one can see a fragment of a stringtable out of an original Armed Assault® mission. As one can see, there's only one language defined.

```
LANGUAGE,English,Notes
```

```
STR_M11_Name,"Night Patrol",Mission Name
```

```
STR_M11_OnLoad,"You're on duty tonight",Onload
```

```
STR_M12_OnLoad,"Don't sleep and keep your eyes open!",Onload
```

```
COMMENT,----- Main Mission -----
```

```
STRCAMP_OBJSTART,Guard the military installation,
```

```
STRM_07an01,"Southern sector, Sahrani",prebriefing
```

```
STRM_07an02,"NATO base in La Riviere, Sahrani",prebriefing
```

```
STRM_07an03,"Near Paraiso - One hour later, Sahrani",prebriefing
```

2.5 - The Init.sqs

The Init.sqs is a simple text-file in the "Missions" folder which can be regarded as the init box of the player character. The game runs this file automatically when the mission starts. The Init.sqs enables a better overview for the player because all entries are more clear now. If all of the syntax is written in the initialization box in the unit menu, the player would lose track of information. What should be written in this script? The user can enter everything that he or she wants to run when the mission begins.

As one can see in the example below, the GPS-System, game-acceleration, and the hidden mission targets 1-3 are predefined. The Teleporter.sqs, which is needed while editing, will run at the beginning of the mission as well. It shall make editing easier for the user. It's quite necessary to deactivate or remove the Teleporter.sqs later.

```
;titlecut
titleCut [" ", "BLACK IN"]; titleFadeOut 4

;pre-load a function
SearchLight = compile preprocessFile "Searchlight.sqf";

;Identity
Player setIdentity "Mr-Murray";

;Hide mission tasks
"MZiel1" ObjStatus "Hidden";
"MZiel2" ObjStatus "Hidden";
"MZiel3" ObjStatus "Hidden";

;GPS-System
[] exec "marker.sqs";

;Choke game speed-up
[] exec "time.sqs";

;Edit script
;Important! It's only for editing your mission. Later you have to delete it!!
;Teleport
[] exec "teleport.sqs";
;End
```

Of course it's possible to define a lot of more things than shown in the script above. For example, the behavior of different units, the arming of units, variables, deleting the unit status, loading several functions and so on. The shown Init.sqs should serve as an example only. All written scripts needs to be defined by the user himself. The mission targets need to be predefined and named as well.

Everything is written very clearly here as one can see. To keep the overview, Arma® will ignore everything that has been defined behind a semicolon.

2.6 - The Script

A script is just a text-file in the missions folder which needs to be defined by the user if he wants to execute special things in the mission. This section doesn't explain the scripting, it only explains the file and how it gets activated. For more information, see **Chapter 9**.

Every single script which is used in Arma® has the same file-type as Operation Flashpoint®, the **xxxx.sqs**. To create a script the user only has to create a text-file which just needs to be renamed. Windows will recognize it as unknown file-type, but that is OK. If the user wants to edit the script file, he only needs to open it with Notepad (text file editor).

You will learn more about scripts in the next Chapter and you also will see some examples which will explain the most important things.

The Init.sqs is actually a script, which will be executed by the game automatically at the beginning of the mission. There is no further syntax needed to run the Init.sqs. This is one of the most important advantages of the Init.sqs. The Mission.sqs and the Description.ext are scripts as well. Only the file-type and the function are different from the **SQS-Script**.

To execute a script out of another script with a trigger or waypoint, the following syntax is needed:

```
[ ] exec "scripts\myscript.sqs"
```

or

```
this exec "scripts\myscript.sqs"
```

After the script has been executed the game runs through the script orders individually. The script will end if the word **exit** has been defined at the end of the script.

2.7 - The Function

One can compare a function with a script. In both cases orders were defined, but there are small but fine differences. One can compare this with cars like a racing car and an old car. But when one takes a deeper look into the details then one can certify that the racing car is more modern than the old one.

There is one large difference between the two. The SQS-File has to be read out by the game every time it is executed, while the SQF-File will be saved in the cache only one time when the mission begins. Operation Flashpoint® used mostly SQS-Files, but it's recommended to use SQF-Files within Arma®.

Functions need to be used according to their type of the utilization. They should be a good solution for everyone if they are written clearly and concise. The most important

thing is that functions should be reusable in other missions without the need to edit them individually.

This should give the user the possibility to define a function only one time, e.g. calculating a special vector, or it could be a solution in a very different problem and make editing much easier for the user. Furthermore, it's better to define several small scripts than only one long script. The performance is not the important thing. It's more important to keep the re-usability of the function.

2.8 - The Paa-Format

The Paa-Format is just an image file-type like the more known JPG-File type. ArmA® mostly uses the formats **.paa** and/or **.pac**. Every single texture which is visible on the objects in the game is of course, a texture file.

One can see a graphic named **Title.paa** which is placed in the subtitle "the missions folder" in this chapter. This graphic is meant for the overview only and is defined in the Overview.html. One could see it now if the player would select the mission out of single-player missions.



The name is variable. The user can name it as he wants, but it's quite important to make sure that the image file is named the same as it is used in the Overview.html. You can get more information by reading the subtitle **The Overview** which is located in **Chapter 2.12**.

Arma® also supports the JPG-Format as Operation Flashpoint® does. So it's possible to implement pictures with this format (e.g. Flags). It's necessary to make sure that one is using the correct image size. Arma® only accepts image sizes that are squared, (preferably powers of 4). There're exceptions only in a few sections (e.g. the briefing and/or the overview). Two-potency are values such as: 2, 4,8,16,32,64,128,256,... a.s.o. these formats are shown as examples below:

64x64

128x128

128x64

256x256

To view and edit .paa and .pac files, a special tool is needed which can be found on the Arma® fan-sites or www.mr-murray.de.vu.

2.9 - The PBO

The PBO-File is a special file-type for all OFP®/ArMA® addons. This file contains all folders, scripts, and images etc. which have been previously collected in the "Addon" folder or "Mission" folder. One can compare PBO-Files with Zip- or Rar-Files, without decreasing the file size.

If the player is saving his mission not as user defined mission but as SP or MP-Mission, the game converts all these files to PBO-Files automatically. While the SP-Missions are located in the directory "ArMA/Missions", the MP-Missions are located in "ArMA/MPMissions".

Not only missions are saved as .pbo files, all add-ons which can be found in ArMA/OFP are PBO-Files as well. One can read these files with special tools. The user has the ability to open existing PBO's to learn how the creator has built an add-on for example. But to learn more about unzipping pbo's, there's a lot more information available on ArMA® fan sites where you also can get the necessary tools

2.10 - The sound files

Most of the sound files which are used in OFP®/ArMA® are Wss- or Ogg-Files. But it's also possible to use wave files. The user only has to make sure that these files are not too large, because maybe one day he wants to offer his missions as a download.

The Wss- or the Ogg-Files are exactly the right ones because these files have a minimum file size to their sound length. This gives the possibility to the user to use several sound files without receiving a mission which has too high of a file size. You will find a more accurate explanation about implementing sound files in **Chapter 5.52**.

Sound files without music should be converted as mono files with a frequency of **44.100 kHz** only. It's important to convert the soundtracks as mono to use the distance effects. If the user wants to add sounds to some objects, he can do so by using the syntax:

Name say "Soundname"

It would be audible on the whole map, and that would be quite unrealistic. So notice, stereo sounds always become global.

There some helpful tools available to convert the files from one format into the other. You only have to check the community sites to get such a tool or use the official BIS-Tools.

2.11 - The LIP-Files

The Lip-Files are needed to move the lips of the character. Every single sound file which is made for a language edition can be equipped with a lip file to move the lips of the character while he's talking.

There are only **3** values needed. The Frame rate of each single motion picture needs to be fixed with the value **0.040** first. This value can be seen as time distance of every single move. Each move of the lips now takes **0.040** seconds.

The degrees of opening needs to be set next. There are **4** different values possible from **0** to **3**. The value **0** means closed while the value **3** means wide open.

Shown on the picture below is an example of a lip file for only **1** second. If one divides **1** second with the value **0.040**, one will get the value **25**. But only **20** lines are shown, and that's because the user has the possibility to define the lip file in that way as well.

For example beginning from the time **0.560**, If one sets the Lip value **1** and makes a break until time **0.720** and sets the value up to **2** then, so we have a break of **4** frames.

```
frame = 0.040
0.000, 0
0.040, 1
0.080, 2
0.120, 3
0.160, 2
0.200, 1
0.240, 0
0.280, 1
0.320, 2
0.360, 1

additional -->
```

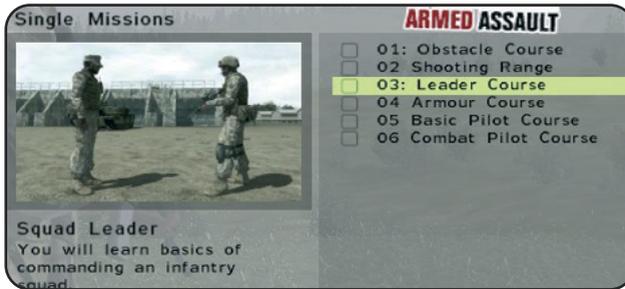
```
0.400, 0
0.440, 1
0.480, 3
0.520, 0
0.560, 1
4 Frame-Pauses
0.720, 2
2 Frame-Pauses
0.800, 1
0.840, 2
0.920, 1
1.000, 2
```

This example displays only one second of lip movement, so one can figure out how long a script would be if one wants to define **10** or **15** seconds.

But there are several tools available which can define those scripts automatically. I will give some links/sources at the end of this guide or check the community sites to get such a tool.

2.12 - The Overview

The overview is a special feature which is shown in single-player missions only. One can see it always as a short description of the mission. The overview and the picture which is shown in the overview, both have to be defined in the Overview.html. One can see an example on the picture below



The mission selection is located at the right side. The description and the image are seen on the left side.

It's quite necessary to have some experience with html, but because you are using this guide such experiences are not really needed. If you really don't have any idea how to create an Overview.html just open an existing one and see how it was created. Just copy the text, make the changes you want to make and add the image.

But if one wants to create his own, he just has to open the text editor and rename and save it as Overview.html into his missions folder. You can use the example below which has been copied from an original ArMA® mission.

```
<html>
<head>
<meta http-equiv="Content-Type"
content="text/html; charset=windows-1250">
<meta name="GENERATOR" content="VB">
<title>Overview</title>
</head>
<body bgcolor="#FFFFFF">
<br>
<!--Night watch-->
<br>
<p align="center"></p>
<BR>
<p>
<!--Mission info>
One more boring night watch. But it's a warm and quiet night...
<!--End of Mission info>
</p></body>
</html>
```

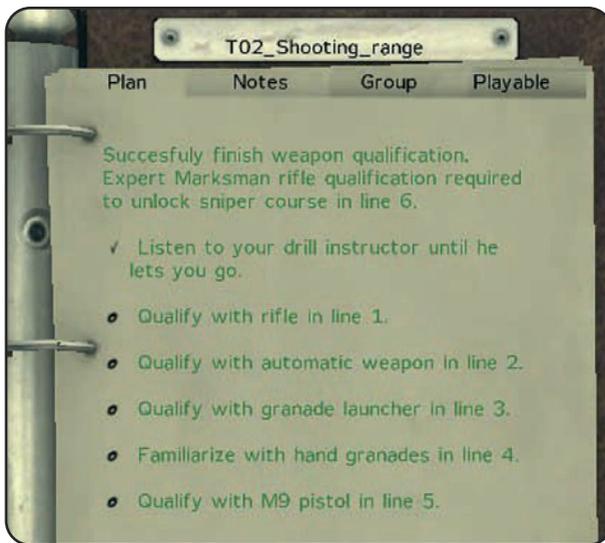
2.13 - The Briefing

The briefing is a quite necessary feature as everyone already knows. To get the briefing to be displayed on the map, there's a Briefing.html needed which is located in the missions folder. The Html-code is much more complex than the one of the overview. It also contains much more information of the mission and the additional mission targets.

One can see the mission's description and the targets in the example below. The first mission objective is already done and has been checked with a green hook. The other objectives are still incomplete and so they remain unchecked. You can get more information in **Chapter 4.5 - The Mission Targets**. First you will learn here how to create a Briefing.html. Later you will understand the briefing much more

The easiest way to create a briefing is to copy an existing one from another mission. One can edit the briefing to his own needs for his mission. Another easy way to create a briefing is to get one of the briefing tools which are located on several Armed-Assault fan sites, this would save the user much performance.

If one wants to open and edit an already existing briefing, the HTML-file only needs to be opened with Notepad.



But I will try to explain to you the way how to create a briefing by yourself. As you can see, there are the sections - plan and notes - located in the briefing. Both sections are actually 2 different pages which have been defined in only one file.

You will need the text editor again as you did while creating the "overview.html". Open the Notepad (text file editor) and write your briefing. Rename the text-file to "Briefing" and save the file as Briefing.html into your missions folder.

There are several briefing files which are all defined in several languages individually as one can see in **Chapter 2.1 - The Missions Folder** located in this chapter. The default Briefing.html will be displayed in English only, if its written in English. It's up to the user to decide which language he uses. If the user wants to define the briefing in several languages he has to keep to some rules and rename each briefing in the correct way.

Briefing.German.html
Overview.German.html
Briefing.France.html
Overview.France.html

The English briefing and overview would be named as follows:

Briefing.html
Overview.html

On the following pages is an example of a briefing.html which coincides with the picture above. Only the notes named in the previous picture have been added. One might be able to create his own briefing if he has a little time and patience. If you also use your creativity and practice a while, you can define a well polished briefing. Beginners can get some Html-Information here.

On the example below one can see a briefing source text:

```
<html>
<head>
<meta http-equiv="Content-Type"
content="text/html; charset=windows-1250">
<meta name="GENERATOR" content="vb">
<title>Briefing</title>
</head>
<body bgcolor="#FFFFFF">
<h2>
<a name="Main"></a>
</h2>
<!-- The notes – Here you can write down your notes.>
<h6>
Damn that's my first shooting lesson.
<br>
</h6>
<!-- End of Notes>
<hr>
<!-- The Mission plan – Here you can put down your mission description.>
<p><a name="Plan"></a>
Successfully finish weapon qualification.<br>
```

```

Expert Marksman rifle qualification required to unlock sniper course in line 6.
</p>
<hr>
<!-- The Mission Tasks-- Here you have to define the mission tasks.>
<p><a name="OBJ_1"></a> Listen to your drill instructor until he lets you go.
</p><hr>
<p><a name="OBJ_2"></a> Qualify with rifle in line 1.
</p><hr>
<p><a name="OBJ_3"></a> Qualify with automatic weapon in line 2.
</p><hr>
<p><a name="OBJ_4"></a> Qualify with grenade launcher in line 3.
</p><hr>
<p><a name="OBJ_5"></a> Familiarize with hand grenades in line 4.
</p><hr>
<p><a name="OBJ_6"></a> Qualify with M9 pistol in line 5.
</p><hr>

<!-- End of mission plan>
<!-- Debriefing - Write down your debriefing>
<hr><br>
<h2><p><a name="Debriefing:End1">Qualified</a></p></h2>
<br><p>
Now I'm a qualified infantryman.
</p><br>
<hr><br>
<h2><p><a name="Debriefing:End2">Title</a></p></h2>
<br><p></p><br>
<hr><br>
<h2><p><a name="Debriefing:End3">Title</a></p></h2>
<br><p></p><br>
<hr><br>
<h2><p><a name="Debriefing:End4">Title</a></p></h2>
<br><p></p><br>
<hr><br>
<h2><p><a name="Debriefing:End5">Title</a></p></h2>
<br><p></p><br>
<hr><br>
<h2><p><a name="Debriefing:End6">Give UP</a></p></h2>
<br><p>
I gave it up. The infantry training is boring.
</p><br>
<!-- END debriefing --->
</body>
</html>

```

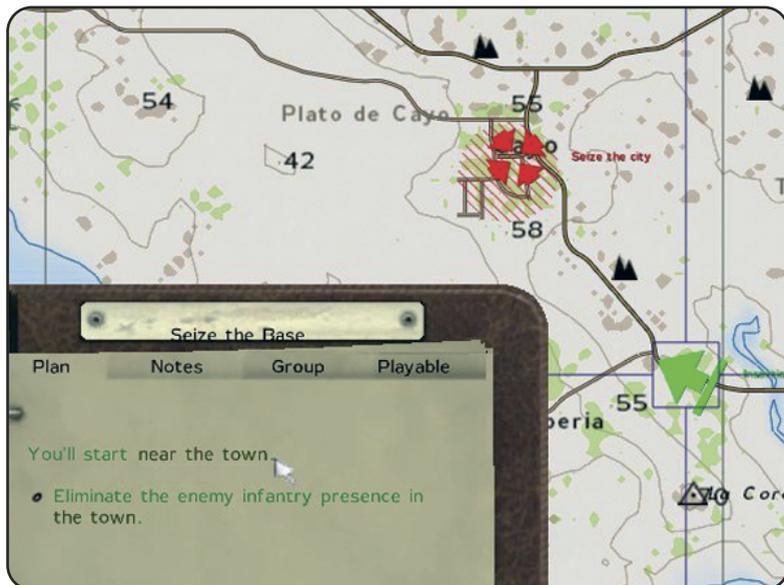
The Html-Document starts with the tags `<html>` and `<head>`. The following tags are not as important right now. The background color has to be defined by the tag `<body bgcolor="#FFFFFF">` although it's already predefined in Armed Assault®. The tag `
` is much more important, it defines a line-break. `<hr>` defines an horizon line which is actually non visible in the briefing. Paragraphs have to be defined by using the `<p>` tag. And last but not least the `<a>` tag, which is needed to define links inside an Html-Document. He who wants to define one of those nice links in the briefing which make the crosshair move to its predefined position can get an example below.

A short example:

If one has set a marker called **Target** on the map, so this one only needs to be linked in the briefing. The sentence in the briefing is called: **Hit and run the target**. The word **Target** has to be linked with its respective position on the map. The order in the Html-Document looks like this

Hit and run the [Target](marker:Target)

The marker called target is defined in the command between `<a>` and ``. If the player clicks on the word **Target** the cross hair would move to the position on the map, as shown in the image below. Commands which are defined with a backslash will end each command.



Chapter 3

- Weapons – Vehicles – Units – Objects –

Now that you have become more familiar with the user interface and the file-types covered in the first two chapters, we now will go to the more specific sections. Also, you should know how to place units on the map and connect them to each other with waypoints. You will now learn all about weapons, vehicles, units and objects in this chapter.

3.1	The hand weapons and static weapons	64
3.2	The weapon classes	68
3.3	Arm and equip units	70
3.4	The weapon and ammo crates	71
3.5	Load and unload vehicles	71
3.6	Weapon selection In the briefing	72
3.7	The vehicle classes	73
3.8	The vehicle weapon classes	76
3.9	The unit classes	77
3.10	The shell classes	80
3.11	The object and building classes	81
3.12	The plant classes	88
3.13	The rock classes	90
3.14	The sign classes	91
3.15	Getting weapon and magazine types displayed	92
3.16	Getting fired type	92
3.17	Does a unit have a weapon?	92
3.18	Primary or secondary weapon of a unit	93
3.19	Does unit have ammunition?	93
3.20	Creating mines	93
3.21	Creating weapons and magazines	94
3.22	Getting weapon view direction displayed	95

Note!

Because of a mistake I have added class names of Low Fly's 1.02 Editor Upgrade, which gives you the possibility to add objects like signs, rocks and plants, here in this chapter. You will need it generally when you want to use the createvehicle command to create a sign for example. I don't wanted to remove the lists and therefore please download it from one of the community websites. Thanks!

3.1 - The hand weapons and static weapons

Here you have a well defined overview of the hand and static weapons, each with a description of the weapon, its magazine, and additional information.

WEST / RESISTANCE – Light Hand Guns



Weapon: M16A2
Magazine: 20Rnd_556x45_Stanag
30Rnd_556x45_Stanag
Grenade:
Flares:



M16A2GL
20Rnd_556x45_Stanag
30Rnd_556x45_Stanag
1Rnd_HE_M203
FlareWhite_M203
FlareGreen_M203
FlareRed_M203
FlareYellow_M203



M4GL - M4A1GL
20Rnd_556x45_Stanag
30Rnd_556x45_Stanag
1Rnd_HE_M203
FlareWhite_M203
FlareGreen_M203
FlareRed_M203
FlareYellow_M203



Weapon: M4
Magazine: 20Rnd_556x45_Stanag
30Rnd_556x45_Stanag
30Rnd_556x45_StanagSD



M4A1SD
20Rnd_556x45_Stanag
30Rnd_556x45_Stanag
30Rnd_556x45_StanagSD



M4AIM
20Rnd_556x45_Stanag
30Rnd_556x45_Stanag
30Rnd_556x45_StanagSD



Weapon: M16A4 - M4A1
Magazine: 20Rnd_556x45_Stanag
30Rnd_556x45_Stanag
Grenade:
Flares:



M16A4_GL
20Rnd_556x45_Stanag
30Rnd_556x45_Stanag
1Rnd_HE_M203
FlareWhite_M203
FlareGreen_M203
FlareRed_M203
FlareYellow_M203



M16A4_ACG_GL
20Rnd_556x45_Stanag
30Rnd_556x45_Stanag
1Rnd_HE_M203
FlareWhite_M203
FlareGreen_M203
FlareRed_M203
FlareYellow_M203



Weapon: M16A4_ACG
Magazine: 20Rnd_556x45_Stanag
30Rnd_556x45_Stanag



MP5A5
30Rnd_9x19_MP5
30Rnd_9x19_MP5SD



MP5SD
30Rnd_9x19_MP5
30Rnd_9x19_MP5SD



Weapon: M4SPR

Magazine: 20Rnd_556x45_Stanag
30Rnd_556x45_Stanag
30Rnd_556x45_StanagSD



M249

20Rnd_556x45_Stanag
30Rnd_556x45_Stanag
30Rnd_556x45_StanagSD
200Rnd_556x45_M249



M240

100Rnd_762x51_M240



Weapon: G36a

Magazine: 30Rnd_556x45_G36



G36C

30Rnd_556x45_G36



G36K

30Rnd_556x45_G36



Weapon: M24

Magazine: 5Rnd_762x51_M24



M107

10Rnd_127x99_M107



Weapon: M9

Magazine: 15Rnd_9x19_M9
15Rnd_9x19_M9SD



M9SD

15Rnd_9x19_M9
15Rnd_9x19_M9SD

WEST / RESISTANCE – Heavy Hand Guns



Weapon: Stinger

Magazine: Stinger



M136

M136



Javelin

Javelin

WEST / RESISTANCE – Static Guns



Weapon: M119

Magazine: 30Rnd_105mmHE_M119



M2StaticMG
M2HD_mini_TriPod

100Rnd_127x99_M2



SearchLight



Weapon: MK19_TriPod
Magazine: 48Rnd_40mm_MK19



Weapon: TOW_TriPod
Magazine: 6Rnd_TOW_Tripod



Weapon: Stinger_Pod
Magazine: 2Rnd_Stinger

EAST – Light Guns



Weapon: AK74
Magazine: 30Rnd_545x39_AK
Grenade:
Flares:



AK74GL
 30Rnd_545x39_AK
 1Rnd_HE_GP25
 FlareWhite_GP25
 FlareGreen_GP25
 FlareRed_GP25
 FlareYellow_GP25



AKS74U
 30Rnd_545x39_AK



Weapon: AKS74UN
Magazine: 30Rnd_545x39_AK
 30Rnd_545x39_AKSD



PK
 100Rnd_762x54_PK



SVD
 10Rnd_762x54_SVD



Weapon: AKS74PSO
Magazine: 30Rnd_545x39_AK



KSVK
 5Rnd_127x108_KSVK



Weapon: Makarov
Magazine: 8Rnd_9x18_Makarov
 8Rnd_9x18_MakarovSD



MakarovSD
 8Rnd_9x18_Makarov
 8Rnd_9x18_MakarovSD

EAST – Heavy Guns

Weapon: 6G30
Magazine: 6Rnd_HE_6G30



RPG7V
 PG7V



Strela
 Strela

EAST – Static Guns

Weapon: D30
Magazine: 30Rnd_122mmHE_D30



DSHKM
 50Rnd_127x107_DSHKM



DSHKM_Mini_TriPod
 50Rnd_127x107_DSHKM



Weapon: TOW_TriPod_East
Magazine: 6Rnd_TOW_Tripod



Stinger_Pod_East
 2Rnd_Stinger



AGS
 29Rnd_30mm_AGS30

Equipment (general)

Weapon: LaserDesignator
Magazine: LaserBatteries



NVGoggles



Binocular



Weapon: Handgrenade
Magazine: Handgrenade



HandGrenadeTimed
 HandGrenadeTimed



Pipebomb
 Pipebomb



Weapon: Mine
Magazine: Mine



MineE
 MineE



SmokeShell
 SmokeShell
 SmokeShellRed
 SmokeShellGreen

3.2 -The weapons class name list

One can take a look at all used weapons and their magazines on the list below. A small description was added to every entry as well. Of course it makes no sense to give out magazines defined with an "SD" for suppressed weapons to a weapon without a suppressor.

West / Resistance

Weapon Class	Description	Ammunition
M16A2	M16A2	Magazine: 20Rnd_556x45_Stanag
M16A4	M16A4	30Rnd_556x45_Stanag
M16A4_ACG	M16A4 - Scope	30Rnd_556x45_StanagSD
M4	M 4 - Standard	
M4A1	M 4 A1 - Standard	
M4A1SD	M 4 - Silencer	
M4AIM	M4 - Aimpoint	
M4SPR	M 4 - Scope	
M16A2GL	Guns with	Magazine: 20Rnd_556x45_Stanag
M16A4_GL	Grenade Launcher	30Rnd_556x45_Stanag
M16A4_ACG_GL		30Rnd_556x45_StanagSD
M4GL		Grenade: 1Rnd_HE_M203
M4A1GL		Flares: FlareWhite_M203 FlareGreen_M203 FlareRed_M203 FlareYellow_M203
M249	M249 SAW	Magazine: 20Rnd_556x45_Stanag 30Rnd_556x45_Stanag 200Rnd_556x45_M249 30Rnd_556x45_StanagSD
M240	M240	Magazine: 100Rnd_762x51_M240
G36a	G 36 – Standard	Magazine: 30Rnd_556x45_G36
G36C	G 36 - Commando	
G36K	G 36 – Commando II	
M24	Sniper Rifle	Magazine: 5Rnd_762x51_M24
M107	Heavy Sniper Rifle	Magazine: 10Rnd_127x99_M107
MP5A5	MP5 - StandardMP5 -	Magazine: 30Rnd_9x19_MP5
MP5SD	Silencer	30Rnd_9x19_MP5SD
M9	Pistol	Magazine: 15Rnd_9x19_M9
M9SD	Pistol - Silencer	15Rnd_9x19_M9SD
M136	AT Rocket Launcher	Magazine: M136
Javelin	AT Rocket Launcher	Magazine: Javelin
Stinger	AA Rocket Launcher	Magazine: Stinger

East

Weapon Class	Description	Ammunition
AK74	AK 74	Magazine: 30Rnd_545x39_AK
AK74GL	AK 74 with Grenade Launcher	Magazine: 30Rnd_545x39_AK Grenade: 1Rnd_HE_GP25 Flares: FlareWhite_GP25 FlareGreen_GP25 FlareRed_GP25 FlareYellow_GP25
AKS74U AKS74UN	AKS 74 U - Standard AKS74UN - Silencer	Magazine 30Rnd_545x39_AK 30Rnd_545x39_AKSD
AKS74PSO	AKS74 - Scope	Magazine: 30Rnd_545x39_AKAK74PSO
PK	MG	Magazine: 100Rnd_762x54_PK
KSVK	Heavy Sniper Rifle	Magazine: 5Rnd_127x108_KSVK
SVD	Sniper Rifle	Magazine: 10Rnd_762x54_SVD
Makarov MakarovSD	Pistol Pistol - Silencer	Magazine: 8Rnd_9x18_Makarov 8Rnd_9x18_MakarovSD
6G30	Grenade Launcher	Magazine: 6Rnd_HE_6G30
RPG7V	AT Rocket Launcher	Magazine: RPG7V
Strela	AA Rocket Launcher	Magazine: Strela

Equipment

Weapon Class	Description	Ammunition
Handgrenade	Handgrenade	Handgrenade
HandGrenadeTimed	Handgrenade (time delay)	HandGrenadeTimed
Grenade	Grenade	Grenade
TimeBomb	Time Bomb	TimeBomb
PipeBomb	Explosive Charge	PipeBomb
SmokeShell	White Smokeshell	SmokeShell
SmokeShellRed	Red Smokeshell	SmokeShellRed
SmokeShellGreen	Green Smokeshell	SmokeShellGreen
Mine	Tank Mine	Mine
MineE	AP Mine	MineE
Binocular	Binoculars	Binocular
NVgoggles	Night Vision Device	NVgoggles
LaserDesignator	Laser Designator	LaserBatteries
CarHorn	Car Horn	CarHorn
SportCarHorn	Sport Car Horn	SportCarHorn
TruckHorn	Truck Horn	TruckHorn
BikeHorn	Bicycle Bell	BikeHorn

3.3 - Arm and equip units

All units which are placeable in Armed Assault® can be armed or unarmed. Its quite well to know that every single unit can carry only one gun (such as a rifle) and only one heavy weapon (such as a LAW or an anti-aircraft weapon). It is possible to add a weapon after the default weapon has been removed first. All weapons can be removed individually and/or completely. A single weapon can be removed by using following syntax:

this removeWeapon "M4" or Name removeWeapon "M4"

The magazines and hand grenades and so on are still in use by the character. If the user adds a weapon to the character which needs the same magazine type, this weapon would be loaded at the beginning. But in the following case it will not appear if the user is using the following syntax:

removeAllWeapons Name

All weapons and all magazines will be removed from the character by using this syntax. If the user wants to rearm that character completely, he has to do it in a special way, because the weapon wouldn't be loaded at the beginning of the mission. The magazines have to be defined first and the weapons have to be defined last to make sure that the weapon will be loaded at the beginning of the mission.

After the weapon (for example the M4) has been removed while using the syntax above, the user can add a new one by using the following syntax:

this addWeapon "M4A1SD" or Name addWeapon "M4A1SD";

That entry can be done in the init. line of the unit or in other places like scripts, triggers or waypoints. If the user wants to remove a magazine only, he only has to use this syntax:

this removeMagazine "30Rnd_556x45_Stanag"

and to rearm with a new magazine just use this syntax:

this addMagazine "30Rnd_556x45_StanagSD"

If all or only some magazines have to be removed, the following syntax can be used as well. All values between **0** and **1** are valid.

Name setVehicleAmmo 0.5

These orders are not meant for the weapons only, they can be used for additional equipment as well. The default unit is not equipped with binoculars or night vision goggles, but those two things are quite useful while traversing the huge landscapes of Sahrani, or at night. To add these weapons to the character the user has to enter following syntax in the init. line of the recipient unit:

To add the binoculars: **this addWeapon "Binocular";**

And for the night vision goggles: **this addWeapon "NVGoggles";**

3.4 - The weapons and ammo crates

All weapons and ammo crates let soldiers equip themselves individually. It makes no difference as to whether one adds weapons and magazines in different ammo crates or only in a single one. To define items for a ammo boxes it's necessary to clear it first. To do this use following syntax:

clearWeaponCargo this or **clearWeaponCargo Name**
clearMagazineCargo this or **clearMagazineCargo Name**

Those entries have to be made in the initialization line of the respective ammo boxes. Its also possible to define them in external areas like scripts and triggers a.s.o. After the ammo box has been cleared, the user can add the weapons and magazines as he'd like to use in the mission. You can **rename** the ammo box or just use the syntax "**this**".

An ammo box will be equipped with **2** suppressed M4A1's, **10** compatible magazines and **6** hand grenades:

```
this addWeaponCargo ["M4A1SD",2];  
this addMagazineCargo ["30Rnd_556x45_StanagSD",10];  
this addMagazineCargo ["Handgrenade",6];
```

Note! Even barrels can be equipped with weapons and ammunition. It's exactly the same way as equipping ammo boxes, the only difference is that the barrels don't have to be cleared beforehand.

3.5 - Load and unload vehicles

Many vehicles in ArMA® are already equipped with weapons, ammunition and similar things. One always has the possibility to rearm himself there. It's also possible to equip vehicles with weapons and ammunition. The same procedure which is done while equipping ammo boxes is done here. To unload vehicles use this syntax:

clearWeaponCargo this
clearMagazineCargo this

To load a vehicle again use this one:

```
this addWeaponCargo ["M4A1SD",2];  
this addMagazineCargo ["30Rnd_556x45_StanagSD",10];
```

In that example, **2** suppressed M4A1's and **10** compatible magazines were loaded into the vehicle.

3.6 - Weapon selection in the briefing

If the player is joining the game as group leader, he has the possibility to edit the weapons and equipment of himself and/or his group members. The only prerequisite is that these features are defined in the description first. Just refer to **Chapter 2.3 - The Description.ext**.

To make sure that it works, the user needs to define the weapon with their similar magazines. One can find 6 suppressed M4A1's with their additional magazines, 2 M136 Rocket launcher's with 6 rockets and 20 hand grenades, in the following example:

```
// Here starts the part of Class Weapons
class Weapons
{
    class M4A1SD
    {
        count = 6;
    };
    class M136
    {
        count = 2;
    };
};
// Here ends the part of Class Weapons

// Here starts the part of Class Magazines
class Magazines
{
    class 30Rnd_556x45_StanagSD
    {
        count = 20;
    };
    class M136
    {
        count = 6;
    };
    class HandGrenade
    {
        count = 20;
    };
};
// Here ends the part of Class Magazines
```

The kind of weapons

The number of weapons

The kind of weapons

The number of weapons

The kind of magazines

The number of magazines

The kind of magazines

The number of magazines

The kind of magazines

The number of magazines

If one wants to add additional weapons he only has to add each class of the weapon and the related magazine to the similar section in the Description.ext.

3.7 - The vehicle classes

WEST

Vehicle	Description	Class Name
Land		
M1Abrams	Tank	M1Abrams
M113	Armored Personnel Tank	M113
M113Ambul	Ambulance Tank	M113Ambul
M113 Mobile HQ	Mobile Headquarter Tank	M113_MHQ
Vulcan	Anti Aircraft Tank	Vulcan
Stryker ICV M2	Light Tank with M2-Machine Gun	Stryker_ICV_M2
Stryker ICV MK19	Light Tank with Grenade Launcher	Stryker_ICV_MK19
Stryker TOW	Light Tank with AT-Launcher	Stryker_TOW
HMMWV	HMMWV	HMMWV
HMMWV M2	HMMWV with M2-Machine Gun	HMMWV50
HMMWV TOW	HMMWV with AT-Launcher	HMMWVTOW
HMMWV MK 19	HMMWV with Grenade Launcher	HMMWVMK
Truck 5 t	Truck 5 Tons	Truck5t
Truck 5 t Open	Truck 5 Tons - open	Truck5tOpen
Truck 5 t MG	Truck 5 Tons with M2-Machine Gun	Truck5tMG
Truck 5 t Repair	Truck 5 Tons - Repair Truck	Truck5tRepair
Truck 5 t Reammo	Truck 5 Tons - Reammo Truck	Truck5tReammo
Truck 5 t Refuel	Truck 5 Tons - Refuel Truck	Truck5tRefuel
Truck 5 t Ammo Truck	Truck 5 Tons - Reammo Truck (Warfare)	WarfareTruck5tReammo
Truck 5 t Supply	Truck 5 Tons - Supply (Warfare)	WarfareWestSupplyTruck
Truck 5 t Salvage	Truck 5 Tons - Salvage (Warfare)	WarfareWestSalvageTruck
Motorcycle	Motorcycle	M1030
Air		
AH 1 Z	Cobra - Helicopter Gunship	AH1W
AH 6	Little Bird Helicopter armed	AH6
AV 8 B	Harrier with Rockets	AV8B2
AV 8 B (GBU)	Harrier with Bombs	AV8B
A10	A10 with Rockets and GAU12-Cannon	A10
MH 6	Little Bird - Helicopter unarmed	MH6
UH 60	Blackhawk - Helicopter with MG	UH60MG
UH 60 (FFAR)	Blackhawk - Helicopter with Rocket Launcher	UH60
Camel	Biplane	Camel
Parachute	Parachute	ParachuteWest
Water		
CRRC	Inflatable Dinghy	Zodiac
RHIB	Patrol Boat with Machine Gun	RHIB
RHIB 2 Turret	Patrol Boat with Machine Gun, Grenade Launcher	RHIB2Turret

EAST

Vehicle	Description	Class Name
Land		
T72	Tank	T72
BMP2	Armoured Personnel Tank	BMP2
BMP2Ambulance	Ambulance Tank	BMP2Ambul
BMP2 Mobile HQ	Mobile Headquarter Tank	BMP2_MHQ
ZSU	Anti Aircraft Tank	ZSU
BRDM2	Light Tank	BRDM2
BRDM2_ATGM	Light Tank with AT-Launcher	BRDM2_ATGM
UAZ	Jeep	UAZ
UAZMG	Jeep with Machine Gun	UAZMG
Ural	Truck	Ural
Ural Open	Truck – open	UralOpen
Ural Repair	Truck – Repair Truck	UralRepair
Ural Reammo	Truck – Reammo Truck	UralReammo
Ural Refuel	Truck – ReFuel Truck	UralRefuel
Ural Ammo	Reammo Truck (Warfare)	WarfareUralReammo
Ural Supply	Lkw - Supply (Warfare)	WarfareEastSupplyTruck
Ural Salvage	Lkw - Salvage (Warfare)	WarfareEastSalvageTruck
Motorcycle	Motorcycle	TT650G
Datsun DshKm - 1	Pick-up with heavy MG	DATSUN_DSHKM1
Datsun DshKm - 2	Pick-up with heavy MG	DATSUN_DSHKM2
Datsun Pk -1	Pick-up with MG	DATSUN_PK1
Datsun Pk -2	Pick-up with MG	DATSUN_PK2
Hilux DshKm - 1	Pick-up with heavy MG	HILUX_DSHKM1
Hilux DshKm - 2	Pick-up with heavy MG	HILUX_DSHKM2
Hilux Pk -1	Pick-up with MG	HILUX_PK1
Hilux Pk -2	Pick-up with MG	HILUX_PK2
Air		
SU 34	SU 34 with Rockets, FFAR, Heavy Cannon	SU34
SU 34B	SU 34 with Rockets and Heavy Cannon	SU34B
Mi 17	Helicopter with Machine Gun	Mi17_MG
Mi 17	Helicopter with Rocket Launcher	Mi17
KA-50	Helicopter Gunship	KA50
Camel E	Biplane	Camel2
Parachute	Parachute	ParachuteEast
Water		
PBX Boat	Inflatable Dinghy	PBX

RESISTANCE

Vehicle	Description	Class Name
Land		
T72 RACS	Tank	T72_RACS
M113 RACS	Armoured Personnel Tank	M113_RACS
Vulkan RACS	Anti Aircraft Tank	Vulkan_RACS
4x4	Jeep (closed)	Landrover
4x4 MG	Jeep with Machine Gun M2	LandroverMG
4x4 Open	Jeep open	Landrover_Closed
Air		
AH6 RACS	Little Bird Helicopter armed	AH6_RACS
MH6 RACS	Little Bird	MH6_RACS
UH60 (FFAR) RACS	UH-60 (FFAR) RACS	UH60RACS
UH-60 RACS	UH-60 RACS	UH60MGRACS
Parachute	Parachute	ParachuteG
Water		
CRRC	Inflatable Dinghy	Zodiac2

CIVILIAN

Vehicle	Description	Class Name
Land		
Pick-Up	Pick-Up blue	Datsun1_civil_1_open
Pick-Up 2	Pick-Up rot (closed)	Datsun1_civil_2_covered
Pick-Up 3	Pick-Up green	Datsun1_civil_3_open
Offroad	Off-Road Vehicle grey (open)	Hilux1_civil_1_open
Offroad2	Off-Road Vehicle red top	Hilux1_civil_3_open
Offroad3	Off-Road Vehicle white (open)	Hilux1_civil_2_covered
Sedan	Car white	Car_sedan
Hatchback	Car red	Car_hatchback
Skoda	Skoda white	Skoda
Skoda (Blue)	Skoda blue	SkodaBlue
Skoda (Red)	Skoda red	SkodaRed
Skoda (Green)	Skoda green	SkodaGreen
Policecar	Police Jeep	Landrover_Police
HMMWV (Civil)	HMMWV (Civil)	HMMWV_civil
Bus	City Bus	Bus_city
UralCivil	Truck yellow (closed)	UralCivil
UralCivil 2	Truck blue (open)	UralCivil2
Traktor	Tractor	Tractor
Motorcycle	Motorcycle	TT650C
Air		
Parachute	Parachute	ParachuteC
Parachute	Empty Parachute	Parachute
DC3	DC3 Civil Plane	DC3

3.8 - The vehicle weapons

Air		
Vehicle	Weapon	Magazine
UH60 Black Hawk MG	M134	200Rnd_762x51_134
UH60 Black Hawk FFAR	FFARLauncher	38Rnd_FFAR
AH 1 Supercobra	HellfireLauncher FFARLauncher	8Rnd_Hellfire 38Rnd_FFAR
AH6 Littlebird FFAR, MG	M197 TwinM134 FFARLauncher	750Rnd_M197_AH1 4000Rnd_762x51_M134 14Rnd_FFAR
AV-8B Harrier	GAU12 SidewinderLauncher	300Rnd_25mm_Gau12 4Rnd_Sidewinder_AV8B
AV-8B Harrier GBU	BombLauncher GAU12	5Rnd_GBU12_AV8B 300Rnd_25mm_Gau12
A10	MaverickLauncher GAU8	5Rnd_Maverick_A10 1350Rnd_30mmAP_A10
Camel	TwinVickers CamelGrenades	500Rnd_TwinVickers 6Rnd_Grenade_Camel
KA-50	2A42 80mmLauncher VikhrLauncher	230Rnd_30mmHE_2A42 230Rnd_30mmAP_2A42 40Rnd_80mm 12Rnd_Vikhr_KA50
MI-17 MG	PKT	2000Rnd_762x54_PKT
MI-17 FFAR	57mmLauncher	96Rnd_57mm
SU 34	R73Launcher S8Launcher GSh301	4Rnd_R73 42Rnd_S8T 180Rnd_30mm_GSh301
SU 34 B	Ch29Launcher GSh301	6Rnd_Ch29 180Rnd_30mm_GSh301
Land/Water		
BMP 2	2A42 PKT AT5Launcher	250Rnd_30mmHE_2A42 250Rnd_30mmAP_2A42 2000Rnd_762x54_PKT 8Rnd_AT5_BMP2
BRDM 2	KPVT PKT	500Rnd_145x115_KPVT 150Rnd_762x54_PKT
BRDM 2 ATGM	AT5Launcher	5Rnd_AT5_BRDM2
M113 MTW	M2	100Rnd_127x99_M2
M113 Vulcan	M168	2100Rnd_20mm_M168
M1A2 Abrahams	M256 M240_Veh	20Rnd_120mmSABOT_M1A2 20Rnd_120mmHE_M1A2 1200Rnd_762x51_M240
T72, T72Racs	D81	23Rnd_125mmSABOT_T72 23Rnd_125mmHE_T72
Shilka	AZP85	2000Rnd_23mm_AZP85
Stryker ATGM	TOWLauncher	2Rnd_TOW
Stryker ICV MG	M2	100Rnd_127x99_M2
Stryker MK19	MK19	48Rnd_40mm_MK19
HMMWV TOW	TOWLauncherSingle	6Rnd_TOW_HMMWV
HMMWV MK19	MK19	48Rnd_40mm_MK19
5to LKW MG	M2	100Rnd_127x99_M2
UAZ AGS 30	AGS30	29Rnd_30mm_AGS30
UAZ MG	DSHKM	50Rnd_127x107_DSHKM
Army 4x4 M2	M2	100Rnd_127x99_M2
RHIB	M2	100Rnd_127x99_M2

3.9 - The unit classes

WEST

Unit	Description	Class Name
AA Specialist	Anti Aircraft Soldier with Stinger	SoldierWAA
AT Specialist	Anti Tank Soldier with M 136	SoldierWAT
Automatic Rifleman	Soldier with Machine Gun M249	SoldierWAR
Camel Pilot	Biplane Pilot with M9 Pistol	BISCamelPilot
Crewman	Vehicle Crew M4A1	SoldierWCrew
Engineer	Engineer with M4 AIM	SoldierWMiner
Grenadier	Grenadier with M4 203	SoldierWG
Machinegunner	Soldier with Machine Gun M240	SoldierWMG
Medic	Corpsman with M4 AIM	SoldierWMedic
Officer	Offizier with M9 Pistol	OfficerW
Pilot	Pilot with M4A1	SoldierWPilot
Rifleman	Soldier with M4AIM	SoldierWB
Rifleman	Soldier with M4AIM	SoldierWNOG
SF Assault	Special Forces with M4A1 GL	SoldierWSaboteurAssault
SF Marksman	Special Forces with M4 SPR	SoldierWSaboteurMarksman
SF Recon	Special Forces with M4 A1 SD	SoldierWSaboteurRecon
SF Saboteur	Special Forces with M4 A1 SD	SoldierWSaboteurPipe
SF Saboteur 2	Special Forces with MP 5 SD	SoldierWSaboteurPipe2
Sniper	Sniper with M24 Sniper Rifle	SoldierWSniper
Squad Leader	Squad Leader with M4 AIM	SquadLeaderW
Team Leader	Team Leader with M4 AIM	TeamLeaderW
Prisoner	Prisoner without Weapon	SoldierWCaptive
USMC Fire Team Leader	Team Leader M16A4 ACOG GL	USMCD_Soldier_TL
USMC Machinegunner	Soldier with Machine Gun M240	USMCD_Soldier_MG
USMC AASpecialist	Anti Aircraft Soldier with Stinger	USMCD_Soldier_AA
USMC Rifleman	Soldier with M16A4	USMCD_Soldier_R
USMC Rifleman (GL)	Soldier with M16A4 ACOG GL	USMCD_Soldier_GL
USMC Rifleman (Mines)	Soldier with M16A4 und Minen	USMCD_Soldier_Engineer
USMC Rifleman (M136)	Anti Tank Soldier with M 136	USMCD_Soldier_AT
USMC Corpsman	Corpsman with M4 AIM	USMCD_Soldier_Med
USMC Squad Leader	Squad Leader with M16A4 ACOG	USMCD_Soldier_SL
USMC Automatic Rifleman	Soldier with Machine Gun M249	USMCD_Soldier_AR
USMC AT Specialist (Javelin)	Anti Tank Soldier with Javeline	USMCD_Soldier_HAT
USMC Sniper	Sniper with M24 Sniper Rifle	USMCD_Soldier_Sniper
USMC Spotter	Soldier with M16A4 ACOG	USMCD_Soldier_Spotter
(WDL) Rifleman	Soldier with M16A4 (WDL)	US_Soldier_WDL
(WDL) Medic	Soldier with M16A4 (WDL)	US_Soldier_WDL_Med
(WDL) Grenadier	Soldier with M16A4 RCO GL (WDL)	US_Soldier_WDL_GL
(WDL) AT Specialist	Anti Tank Soldier with M 136 (WDL)	US_Soldier_WDL_AT
(WDL) AA Specialist	Anti Aircraft Soldier with Stinger (WDL)	US_Soldier_WDL_AA
(WDL) Machinegunner	Soldier with Machine Gun M240 (WDL)	US_Soldier_WDL_MG
(WDL) Engineer	Soldier with M16A4 and Mines (WDL)	US_Soldier_WDL_Engineer
(WDL) Designated Marksman	Sniper with M24 Sniper Rifle (WDL)	US_Soldier_WDL_Sniper
(WDL) Automatic Rifleman	Soldier with Machine Gun M249 (WDL)	US_Soldier_WDL_AR
(WDL) Team Leader	Team Leader with M4 AIM	US_Soldier_WDL_TL
(WDL) Squad Leader	Squad Leader with M16A4 ACOG (WDL)	US_Soldier_WDL_SL
Mercenary Team Leader	Mercenary Team Leader	SoldierMTeamLeader
Mercenary Grenadier	Mercenary Grenadier	SoldierMG
Mercenary Machinegunner	Mercenary Machinegunner	SoldierMMG
Mercenary Engineer	Mercenary Engineer	SoldierMR
Mercenary Sniper	Mercenary Sniper	SoldierMS
Mercenary Saboteur	Mercenary Saboteur	SoldierMD

EAST

Unit	Description	Class Name
AA Specialist	Anti Aircraft Soldier with Strela	SoldierEAA
AT Specialist	Anti Tank Soldier with RPG 7 V	SoldierEAT
Camel Pilot	Biplane Pilot	BISCamelPilot2
Crewman	Vehicle Crew	SoldierECrew
Engineer	Engineer	SoldierEMiner
Especas	Speznaz with AKS 74 U	SoldierESaboteurPipe
Especas Marksman	Speznaz with AKS74PSO	SoldierESaboteurMarksman
Especas Saboteur	Speznaz with AKS 74 UN	SoldierESaboteurBizon
Grenadier	Grenadier	SoldierEG
Machinegunner	Soldier with Machine Gun PK	SoldierEMG
Medic	Corpsman	SoldierEMedic
Rifleman	Soldier with AK-74	SoldierEB
Rifleman	Soldier with AK-74	SoldierENOG
Officer	Officer	OfficerE
Pilot	Pilot	SoldierEPilot
Sniper	Sniper with Dragunov (SVD)	SoldierESniper
Squad Leader	Squad Leader	SquadLeaderE
Team Leader	Team Leader	TeamLeaderE
Prisoner	Prisoner	SoldierECaptive
Partisan Team Leader	Partisan Team Leader	SoldierPTeamLeader
Partisan AT Specialist	Partisan AT Specialist	SoldierPAT
Partisan Saboteur	Partisan Saboteur	SoldierPSaboteur
Partisan Machinegunner	Partisan Machinegunner	SoldierPMG
Partisan Medic	Partisan Medic	SoldierPMedic
Partisan Rifleman	Partisan Rifleman	SoldierPB

RESISTANCE

Unit	Description	Class Name
AA Specialist	Anti Aircraft Soldier with Stinger	SoldierGAA
AT Specialist	Anti Tank Soldier with M136	SoldierGAT
Crewman	Vehicle Crew	SoldierGCrew
Engineer	Engineer	SoldierGMiner
Grenadier	Grenadier	SoldierGG
Machinegunner	Soldier with Machine Gun M240	SoldierGMG
Medic	Corpsman	SoldierGMedic
Officer	Offizier	OfficerG
Pilot	Pilot	SoldierGPilot
Rifleman	Soldier with M16 A2	SoldierGB
Rifleman	Soldier with M16 A2	SoldierGNOG
Royal Commando	Royal Commando with MP 5 SD	SoldierGCommando
Royal Guard	Royal Guard with G36c	SoldierGGuard
Royal Marksman	Royal Sniper with G36a	SoldierGMarksman
Sniper	Sniper with Sniper Rifle M24	SoldierGSniper
Squad Leader	Squad Leader	SquadLeader
Team Leader	Team Leader	TeamLeader
Prisoner	Prisoner	SoldierGCaptive
Spy	Spy	SoldierSpy

CIVILIAN

Unit	Description	Class Name
Civilian to Civilian21	Nearer description is not necessary. Numbered from Civilian to Civilian21.	Civilian
Civilian Man	Civilian Man	D2_RCM03_Civilian1
Civilian Man	Civilian Man	D2_RCM03_Civilian2
King	King	King
War Correspondent	War Correspondent	FieldReporter
Bodyguard	Bodyguard	Anchorman
Prime Minister	Prime Minister	NorthPrimeMinister
Reporter (Female)	Reporter (Female)	MarianQuandt
Reporter (Female)	Reporter (Female)	MarianQuandt02
Reporter (Female)	Reporter (Female)	MarianQuandt03
Reporter (Female)	Reporter (Female)	MarianQuandt04
Zombie	Zombie	Civil_Undead_1
Zombie	Zombie	Civil_Undead_2
Zombie	Zombie	Civil_Undead_3
Zombie	Zombie	Civil_Undead_4
Prisoner 1	Prisoner 1	Prisoner01
Prisoner 2	Prisoner 2	Prisoner02
Prisoner 3	Prisoner 3	Prisoner03
Prisoner 4	Prisoner 4	Prisoner04
Prisoner 5	Prisoner 5	Prisoner05
Prince (civil)	Prince (civil)	Prince_civil
Prince (Army)	Prince (Army)	Prince_army
Prince (Partisan)	Prince (Partisan)	Prince_resistance
Chancellor	Chancellor	Chancellor
Adjutant (Uniform)	Adjutant (Uniform)	AdjutantUniform
Adjutant (Prisoner)	Adjutant (Prisoner)	AdjutantPrisoner
Arms trader	Arms trader	ArmsTrader
SLA President	SLA President	civil_nprem2
SLA President	SLA President	civil_nprem2_NoGeom

INSECTS

Types which are defined with N/A are not available in the Editor. These units can be generated by using the CreateVehicle command as explained in **Chapter 5.45**.

Type	Description	Class Name
N/A	Seagull	Seagull
N/A	Dragonfly	Dragonfly
N/A	HouseFly	HouseFly
N/A	Honeybee	Honeybee
N/A	Mosquito	Mosquito
N/A	Butterfly	Butterfly

3.10 - The shell classes

One can find a selection of the Shell Classes. Shells are the bullets of each single Weapon. It's possible to do everything by using this command, i.e. an explosion or get other fired attack simulated. Those Shells can be created by using the create vehicle command as explained in **Chapter 5.45/5.46**. An example of such a command can be seen below:

Bomb="SH_125_HE" createVehicle [x,y,z]

Bomb="SH_125_HE" createVehicle position Player

Shell Types

B_9x18_Ball	B_762x54_noTracer
B_9x18_SD	B_77x56_Ball
B_9x19_Ball	B_127x99_Ball
B_9x19_SD	B_127x107_Ball
B_127x108_Ball	B_145x115_AP
B_127x99_Ball_noTracer	B_20mm_AP
B_545x39_Ball	B_20mm_AA
B_545x39_SD	B_23mm_AA
B_556x45_Ball	B_25mm_HE
B_556x45_SD	B_30mm_AP
B_762x51_Ball	B_30mm_HE
B_762x54_Ball	B_30mmA10_AP

Rocket and Grenade Types

R_Hydra_HE	M_Sidewinder_AA
R_57mm_HE	M_TOW_AT
R_80mm_HE	M_Hellfire_AT
R_M136_AT	M_Maverick_AT
R_PG7V_AT	M_Vikhr_AT
R_KSVK	BO_GBU12_LGB
R_PG7VR_AT	SH_125_HE
M_Javelin_AT	SH_120_HE
M_Stinger_AA	SH_122_HE
M_Strela_AA	SH_105_HE
M_AT5_AT	G_40mm_HE_6G30

Others

Class Name	Description
FxExploGround1	Rock Fragment
FxExploGround2	Rock Fragment
FxExploArmor1	Shell Splinter
FxExploArmor2	Shell Splinter
FxExploArmor3	Shell Splinter
FxExploArmor4	Shell Splinter
FxCartridge	Cartridge Case
Bomb	Grenade (requires setDamage)
LaserTargetW	Laser Target West
LaserTargetE	Laser Target East
LaserTargetC	Laser Target Civilian

3.11 - The object and building classes

The following list contains a selection of objects which can be placed directly or indirectly somewhere on the map. To place the objects the create-vehicle command is needed again as explained in **Chapter 5.45**.

Type	Description	Class Name
A Camp	A Camp	ACamp
Bmp2 Wreck	Bmp2 Wreck	Bmp2Wreck
Barrels	Barrels	Barrels
Barrel (red)	Barrel (red)	Barrel1
Barrel (brown)	Barrel (brown)	Barrel2
Barrel (yellow)	Barrel (yellow)	Barrel3
Barrel (green rusty)	Barrel (green rusty)	Barrel4
Barrel (green rusty)	Barrel (green rusty)	Barrel5
Barrel (white purple)	Barrel (white purple)	Barrel6
Body	Body	Body
Blackhawk Wreck	Blackhawk Wreck	BlackhawkWreck
Camera	Camera	Camera1
Camp	Camp	Camp
Camp Empty	Camp Empty	CampEmpty
Camp East	Camp East	CampEast
Camp East C	Camp East C	CampEastC
Computer	Computer	Computer
Danger	Danger	Danger
DangerWest	DangerWest	DangerWest
DangerGUE	DangerGUE	DangerGUE
DangerEAST	DangerEAST	DangerEAST
Datsun Wreck 1	Datsun Wreck 1	Datsun01Wreck
Datsun Wreck 2	Datsun Wreck 2	Datsun02Wreck
Fence	Fence	Fence
Fire	Fire	Fire
FireLit	FireLit	FireLit
FlagCarrierWest	FlagCarrierWest	FlagCarrierWest
FlagCarrierNorth	FlagCarrierNorth	FlagCarrierNorth
FlagCarrierSouth	FlagCarrierSouth	FlagCarrierSouth
Fortress1	Fortress1	Fortress1
Fortress2	Fortress2	Fortress2
FenceWood	FenceWood	FenceWood
FenceWoodPalet	FenceWoodPalet	FenceWoodPalet
JeepWreck1	JeepWreck1	JeepWreck1
JeepWreck2	JeepWreck2	JeepWreck2
JeepWreck3	JeepWreck3	JeepWreck3
Grave	Grave	Grave
GraveCross1	GraveCross1	GraveCross1
GraveCross2 Grave	GraveCross2 Grave	GraveCross2
CrossHelmet	CrossHelmet	GraveCrossHelmet
Heli	Heli	Heli
Heli-H Empty	Heli-H Empty	HeliHEmpty
Heli-H Civil	Heli-H Civil	Heli_H_Civil
Heli-H Rescue	Heli-H Rescue	Heli_H_Rescue
Hilux Wreck	Hilux Wreck	HiluxWreck
Land_ladder	Land_ladder	Land_Ladder
Land Radar	Land Radar	Land_Radar
Obstacle	Obstacle	Land_obihacka
Training	Training	Land_podlejzacka
Training 2	Training 2	Land_prolejzacka
Training 3	Training 3	Land_prebehlavka
Carousel	Carousel	Land_kolotoc

Type	Description	Class Name
Carousel small	Carousel small	Land_maly_kolotoc
Swing	Swing	Land_houpacka
Prolezacka	Prolezacka	Land_kulata_prolezacka
SandPit	SandPit	Land_Piskoviste
Obstacle	Obstacle	Land_jezekbeton
Fuel Tank Small	Fuel Tank Small	Land_fuel_tank_small
Fuel Tank Big	Fuel Tank Big	Land_fuel_tank_big
Fuel Tank Letter	Fuel Tank Letter	Land_fuel_tank_stairs
Land Water Tank	Land Water Tank	Land_water_tank
Land Water Tank	Land Water Tank	Land_water_tank2
MASH	MASH	MASH
M113 Wreck	M113 Wreck	M113Wreck
Paleta1	Paleta1	Paleta1
Paleta2	Paleta2	Paleta2
Radio	Radio	Radio
Shed	Shed	Shed
Shed Small	Shed Small	ShedSmall
Shed Big	Shed Big	ShedBig
TargetE	TargetE	TargetE
TargetEpopup	TargetEpopup	TargetEpopup
Wood tank	Wood tank	TargetGrenade
TV Studio	TV Studio	TVStudio
TV Studio (Building)	TV Studio (Building)	Land_Vysilac_budova
TV Radio Tower	TV Radio Tower	Land_Vysilac_vez
TV Radio Tower Gangway	TV Radio Tower Gangway	Land_Vysilac_chodba
Ural Wreck	Ural Wreck	UralWreck
Bale Of Straw	Bale Of Straw	Vec03
Closet	Closet	Land_Toilet
Vysilacka	Vysilacka	Vysilacka
Wall Map	Wall Map	WallMap
RahmadiMap	RahmadiMap	RahmadiMap
Sleeping Bag	Sleeping Bag	Land_SleepingBag
Wire	Wire	Wire
Wire Fence	Wire Fence	WireFence
Barrier	Barrier	ZavoraAnim
AmmoBoxWest	AmmoBoxWest	AmmoBoxWest
SpecialBoxWest	SpecialBoxWest	SpecialBoxWest
WeaponBoxWest	WeaponBoxWest	WeaponBoxWest
AmmoBoxEast	AmmoBoxEast	AmmoBoxEast
SpecialBoxEast	SpecialBoxEast	SpecialBoxEast
WeaponBoxEast	WeaponBoxEast	WeaponBoxEast
AmmoBoxGuer	AmmoBoxGuer	AmmoBoxGuer
SpecialBoxGuer	SpecialBoxGuer	SpecialBoxGuer
WeaponBoxGuer	WeaponBoxGuer	WeaponBoxGuer
Little Church	Little Church	Land_kostel
Big Church	Big Church	Land_kostel2
Middle Church	Middle Church	Land_kostel3
Mexican Church	Mexican Church	Land_kostel_mexico
Church	Church	Land_kostelik
House	House	Land_sara_domek01
House	House	Land_sara_domek02
House	House	Land_sara_domek05
House	House	Land_sara_zluty_statek_in
House	House	Land_sara_Domek_sedy
House	House	Land_dum_mesto2
House	House	Land_sara_domek_sedy_bez
House	House	Land_sara_domek_rosa
House	House	Land_sara_zluty_statek
House	House	Land_sara_domek_zluty

Type	Description	Class Name
House	House	Land_sara_domek_zluty_bez
House	House	Land_OrlHot
Houseblock (small)	Houseblock (small)	Land_Panelak
Houseblock (middle)	Houseblock (middle)	Land_Panelak2
Houseblock (big)	Houseblock (big)	Land_Panelak3
Hotel	Hotel	Land_Hotel
Hotelruin	Hotelruin	Land_Hotel_ruins
Holiday-In	Holiday-In	Land_hotel_riviera1
Holiday-In	Holiday-In	Land_hotel_riviera2
Holiday-In-Ruin	Holiday-In-Ruin	Land_hotel_riviera1_ruins
Holiday-In-Ruin	Holiday-In-Ruin	Land_hotel_riviera2_ruins
Weekend Flat	Weekend Flat	Land_house_y
House (Oriental)	House (Oriental)	Land_dum_olez_istan1
House (Oriental)	House (Oriental)	Land_dum_olez_istan2
House (Oriental)	House (Oriental)	Land_dum_olez_istan2_maly
House (Oriental)	House (Oriental)	Land_dum_olez_istan2_maly2
House (Oriental/Open)	House (Oriental/Open)	Land_dum_istan2
House (Oriental/Open)	House (Oriental/Open)	Land_dum_istan2b
House (Oriental)	House (Oriental)	Land_dum_istan2_01
House (Oriental)	House (Oriental)	Land_dum_istan2_02
House (Oriental)	House (Oriental)	Land_dum_istan2_03
House (Oriental)	House (Oriental)	Land_dum_istan2_03a
House (Oriental)	House (Oriental)	Land_dum_istan2_04a
House (Oriental/Open)	House (Oriental/Open)	Land_dum_istan3
House (Oriental)	House (Oriental)	Land_dum_istan3_hromada
House (Oriental)	House (Oriental)	Land_dum_istan3_hromada2
House (Oriental)	House (Oriental)	Land_dum_istan3_pumpa
House (Oriental)	House (Oriental)	Land_dum_mesto3_istan
Tall House (Oriental)	Tall House (Oriental)	Land_dum_istan4
Tall House (Oriental)	Tall House (Oriental)	Land_dum_istan4_big
Tall House (Oriental)	Tall House (Oriental)	Land_dum_istan4_big_inverse
Tall House (Oriental)	Tall House (Oriental)	Land_dum_istan4_detaily1
Tall House (Oriental)	Tall House (Oriental)	Land_dum_istan4_inverse
House	House	Land_dumruina
House	House	Land_dumruina_mini
House	House	Land_sara_domek_kovarna
House	House	Land_dum_rasovna
House	House	Land_Statek_kulna
House	House	Land_stanice
House	House	Land_ryb_domek
House (Classic)	House (Classic)	Land_statek_hl_bud
Little House	Little House	Land_bouda1
Old House	Old House	Land_sara_domek_ruina
Red House	Red House	Land_cihlovej_dum
Red House	Red House	Land_cihlovej_dum_in
Red House	Red House	Land_cihlovej_dum_mini
City House (yellow)	City House (yellow)	Land_kasarna_rohova
House	House	Land_sara_domek05
House	House	Land_sara_zluty_statek_in
House	House	Land_sara_Domek_sedy
House	House	Land_dum_mesto2
House	House	Land_sara_domek_sedy_bez
House	House	Land_sara_domek_rosa
House	House	Land_sara_zluty_statek
House	House	Land_sara_domek_zluty
House	House	Land_sara_domek_zluty_bez
House	House	Land_OrlHot
Houseblock (small)	Houseblock (small)	Land_Panelak
Houseblock (middle)	Houseblock (middle)	Land_Panelak2

Type	Description	Class Name
Houseblock (big)	Houseblock (big)	Land_Panelak3
Hotel	Hotel	Land_Hotel
Hotelruin	Hotelruin	Land_Hotel_ruins
Holiday-In	Holiday-In	Land_hotel_riviera1
Holiday-In	Holiday-In	Land_hotel_riviera2
Holiday-In-Ruin	Holiday-In-Ruin	Land_hotel_riviera1_ruins
Holiday-In-Ruin	Holiday-In-Ruin	Land_hotel_riviera2_ruins
Weekend Flat	Weekend Flat	Land_house_y
House (Oriental)	House (Oriental)	Land_dum_olez_istan1
House (Oriental)	House (Oriental)	Land_dum_olez_istan2
House (Oriental)	House (Oriental)	Land_dum_olez_istan2_maly
House (Oriental)	House (Oriental)	Land_dum_olez_istan2_maly2
House (Oriental/Open)	House (Oriental/Open)	Land_dum_istan2
House (Oriental/Open)	House (Oriental/Open)	Land_dum_istan2b
House (Oriental)	House (Oriental)	Land_dum_istan2_01
House (Oriental)	House (Oriental)	Land_dum_istan2_02
House (Oriental)	House (Oriental)	Land_dum_istan2_03
House (Oriental)	House (Oriental)	Land_dum_istan2_03a
House (Oriental)	House (Oriental)	Land_dum_istan2_04a
House (Oriental/Open)	House (Oriental/Open)	Land_dum_istan3
House (Oriental)	House (Oriental)	Land_dum_istan3_hromada
House (Oriental)	House (Oriental)	Land_dum_istan3_hromada2
House (Oriental)	House (Oriental)	Land_dum_istan3_pumpa
House (Oriental)	House (Oriental)	Land_dum_mesto3_istan
Tall House (Oriental)	Tall House (Oriental)	Land_dum_istan4
Tall House (Oriental)	Tall House (Oriental)	Land_dum_istan4_big
Tall House (Oriental)	Tall House (Oriental)	Land_dum_istan4_big_inverse
Tall House (Oriental)	Tall House (Oriental)	Land_dum_istan4_detaily1
Tall House (Oriental)	Tall House (Oriental)	Land_dum_istan4_inverse
House	House	Land_dumruina
House	House	Land_dumruina_mini
House	House	Land_sara_domek_kovarna
House	House	Land_dum_rasovna
House	House	Land_Statek_kulna
House	House	Land_stanice
House	House	Land_ryb_domek
House (Classic)	House (Classic)	Land_statek_hl_bud
Little House	Little House	Land_bouda1
Old House	Old House	Land_sara_domek_ruina
Red House	Red House	Land_cihlovej_dum
Red House	Red House	Land_cihlovej_dum_in
Red House	Red House	Land_cihlovej_dum_mini
City House (yellow)	City House (yellow)	Land_kasarna_rohova
City House (yellow)	City House (yellow)	Land_kasarna_brana
City House (yellow)	City House (yellow)	Land_kasarna
City House (yellow)	City House (yellow)	Land_kasarna_prujezd
Pub	Pub	Land_hospoda_mesto
Building	Building	Land_skola
Landhouse	Landhouse	Land_deutshe
Landhouse	Landhouse	Land_deutshe_mini
Landhouse	Landhouse	Land_domek_rosa
Landhouse	Landhouse	Land_dum_m2
City House (Classic)	City House (Classic)	Land_dum_mesto
City House (Classic)	City House (Classic)	Land_dum_mesto_in
House with	House with	Land_dum_mesto2l
House	House	Land_dum_mesto3
House (long)	House (long)	Land_dum_olezlina
House (flat)	House (flat)	Land_dum01
Big House	Big House	Land_dum02

Type	Description	Class Name
House	House	Land_sara_domek_hospoda
House	House	Land_sara_domek_podhradi_1
Villa	Villa	Land_sara_domek_vilka
Convent Building (Corner)	Convent Building (Corner)	Land_sara_dum_podloubi03klaster
Convent Building	Convent Building	Land_sara_dum_podloubi03rovny
Garage	Garage	Land_sara_hasic_zbroj
Garage	Garage	Land_garaz
Garage	Garage	Land_garaz_mala
Beach Hut	Beach Hut	Land_hut01
Beach Hut	Beach Hut	Land_hut02
Beach Hut (open)	Beach Hut (open)	Land_hut03
Beach Hut	Beach Hut	Land_hut04
Beach Hut	Beach Hut	Land_hut06
Holiday Hut	Holiday Hut	Land_ZalChata
Little Hut	Little Hut	Land_psi_bouda
Little Shanty	Little Shanty	Land_bouda2_vnitrek
Shelter (little)	Shelter (little)	Land_kulna
Shanty	Shanty	Land_bouda3
Latrine	Latrine	Land_KBud
Big Barn (closed)	Big Barn (closed)	Land_stodola_old
Big Barn (open)	Big Barn (open)	Land_stodola_old_open
Barn	Barn	Land_strazni_vez
Barn (open)	Barn (open)	Land_sara_stodola
Barn (Closed)	Barn (Closed)	Land_sara_stodola2
Barn (open)	Barn (open)	Land_sara_stodola3
Old Barn	Old Barn	Land_hut_old02
Building (burned)	Building (burned)	Land_afbarabizna
Houseruin (burned)	Houseruin (burned)	Land_afdum_mesto2
Houseruin (burned)	Houseruin (burned)	Land_afdum_mesto2L
Houseruin (burned)	Houseruin (burned)	Land_afdum_mesto3
Pubruin (burned)	Pubruin (burned)	Land_afhospoda_mesto
Houseruin	Houseruin	Land_dulni_bs
Houseruin	Houseruin	Land_dum_zboreny
Houseruin	Houseruin	Land_dum_zboreny_total
Houseruin (open)	Houseruin (open)	Land_hruzdu
Churchruin	Churchruin	Land_kostel_trosky
Bus Stop	Bus Stop	Land_zastavka_jih
Bus Stop 2	Bus Stop 2	Land_zastavka_sever
Market Stall	Market Stall	Land_stanek_1
Market Stall 1	Market Stall 1	Land_stanek_1B
Market Stall 2	Market Stall 2	Land_stanek_1C
Military Shelter	Military Shelter	Land_army_hut_storage
Military Building (open)	Military Building (open)	Land_army_hut_int
Military Building (little)	Military Building (little)	Land_army_hut2
Military Building (open)	Military Building (open)	Land_army_hut2_int
Military Building (closed)	Military Building (closed)	Land_army_hut3_long
Military Building (open)	Military Building (open)	Land_army_hut3_long_int
Bus Stop	Bus Stop	Land_aut_zast
Ammunition Bunker	Ammunition Bunker	Land_garaz_s_tankem
Ammunition Bunker	Ammunition Bunker	Land_garaz_bez_tanku
Ammunition Bunker	Ammunition Bunker	Land_ammostore2
Metal Tower	Metal Tower	Land_hlaska
Wood Tower	Wood Tower	Land_posed
Metal Tower	Metal Tower	Land_vez
Airport Tower	Airport Tower	Land_letistni_hala
Military Building	Military Building	Land_budova1
Military Building	Military Building	Land_budova2
Military Building	Military Building	Land_budova3
Military Building	Military Building	Land_budova4

Type	Description	Class Name
Military Building	Military Building	Land_budova4_in
Guardhouse	Guardhouse	Land_budova5
Military Hospital	Military Hospital	Land_hospital
Repair Center	Repair Center	Land_repair_center
Hangar (green)	Hangar (green)	Land_SS_hangar
Hangar (grey)	Hangar (grey)	Land_SS_hangarD
Hangar	Hangar	Land_hangar_2
Hangar ruin	Hangar ruin	Land_SS_hangar_ruins
Fuelstop (small)	Fuelstop (small)	FuelStation
Fuelstop (small)	Fuelstop (small)	Land_fuelstation
Fuelstop (big)	Fuelstop (big)	Land_benzina_schnell
Fuelstop (Military)	Fuelstop (Military)	Land_fuelstation_army
Factory	Factory	Land_Tovarna1
Factory	Factory	Land_Tovarna2
Tall Tower	Tall Tower	Land_komin
Metall Hut	Metall Hut	Land_Hlidac_budka
Little Metal Hut	Little Metal Hut	Land_bouda_plech
Glass Tower	Glass Tower	Land_strazni_vez
Transformer Station	Transformer Station	Land_trafostanica_velka
Transformer Station	Transformer Station	Land_trafostanica_mala
Radiotower	Radiotower	Land_Vysilac_FM
Radiotower	Radiotower	Land_vysilac_FM2
Radiotower	Radiotower	Land_telek1
Water Tower	Water Tower	Land_watertower1
Silo	Silo	Land_Nasyпка
Lighthouse	Lighthouse	Land_majak
Lighthouse	Lighthouse	Land_majak2
Lighthouse with podest	Lighthouse with podest	Land_majak_v_celku
Lighthousepodest	Lighthousepodest	Land_majak_podesta
Harbour piece	Harbour piece	Land_molo_beton
Harbour piece	Harbour piece	Land_molo_krychle
Harbour piece	Harbour piece	Land_molo_krychle2
Bridge with roof	Bridge with roof	Land_molo_drevo
Bridge	Bridge	Land_molo_drevo_bs
Bridge (end)	Bridge (end)	Land_molo_drevo_end
Fence	Fence	Land_pletivo_dira
Fencegate	Fencegate	Land_plot_zed_drevo1_branka
Wall	Wall	Land_plot_istan1b_hole
Stonegate	Stonegate	Land_plot_istan1_rovny_gate
Archway	Archway	Land_brana02
Archway	Archway	Land_brana02nodoor
Stonefence	Stonefence	Land_plot_zboreny
Woodfence	Woodfence	Land_Plot_Ohrada_Pruchozi
Basefence (camo)	Basefence (camo)	Land_zed_dira
Basefence (grey)	Basefence (grey)	Land_zed_dira_desert
Basefence (Desert)	Basefence (Desert)	Land_zed_dira_civil
Stoplight	Stoplight	Land_Stoplight01
Stoplight 2	Stoplight 2	Land_Stoplight02
Landfield light	Landfield light	Land_Runway_PAPI
Landfield light	Landfield light	Land_Runway_PAPI_2
Landfield light	Landfield light	Land_Runway_PAPI_3
Landfield light	Landfield light	Land_Runway_PAPI_4
Power supply line	Power supply line	Land_trafostanica_velka_draty
Electricity pylon	Electricity pylon	Land_sloup_vn
Power supply line	Power supply line	Land_sloup_vn_dratZ
Power supply line	Power supply line	Land_sloup_vn_drat
Ladder (big)	Ladder (big)	Land_ladder
Scaffold	Scaffold	Land_leseni2x
Scaffold	Scaffold	Land_leseni4x

Type	Description	Class Name
Castletower	Castletower	Land_helfenburk
Castlewall	Castlewall	Land_helfenburk_brana
Castlewall	Castlewall	Land_helfenburk_budova2
Castlewall	Castlewall	Land_helfenburk_cimburi
Castlewall	Castlewall	Land_helfenburk_zed
Wall with door	Wall with door	Land_zidka_branka
Wall ruin	Wall ruin	Wallend
Wall ruin	Wall ruin	Land_zidka03
Wall with gate (closed)	Wall with gate (closed)	Land_statek_brana
Wall with gate (open)	Wall with gate (open)	Land_statek_brana_open
Fountain	Fountain	Land_pumpa
Oil pump	Oil pump	Land_vez_ropa
Citywall	Citywall	Land_podesta_1_cube
Citywall	Citywall	Land_podesta_1_cube_long
Citywall	Citywall	Land_podesta_1_corn1
Citywall	Citywall	Land_podesta_1_cornp
Citywall	Citywall	Land_podesta_1_mid_corn1
Citywall	Citywall	Land_podesta_1_mid_cornp
Citywall	Citywall	Land_podesta_1_mid
Citywall	Citywall	Land_podesta_1_stairs
Citywall	Citywall	Land_podesta_1_stairs2
Citywall	Citywall	Land_podesta_1_stairs3
Citywall	Citywall	Land_podesta_1_stairs4
Stoned area	Stoned area	Land_podesta_5
Stoned area	Stoned area	Land_podesta_10
Scrapheap	Scrapheap	Land_Afbarabizna_ruins
Scrapheap	Scrapheap	Land_AFDum_mesto2_ruins
Scrapheap	Scrapheap	Land_AFDum_mesto2L_ruins
Minaret	Minaret	Land_R_Minaret
Old Silo Bam - Sawmill	Old Silo Bam - Sawmill	Land_Kamenolom_budova
Old Silo Bam - Sawmill	Old Silo Bam - Sawmill	Land_pila
Big Carport	Big Carport	Land_pristresek
Big Carport with camo net	Big Carport with camo net	Land_pristresek_camo
Tower with searchlight	Tower with searchlight	Land_Vez_svetla
Camouflaged water tower	Camouflaged water tower	Land_vodni_vez
Aircraft Factory	Aircraft Factory	WarfareBAircraftFactory
Aircraft Factory (West)	Aircraft Factory (West)	WarfareBWestAircraftFactory
Aircraft Factory (SLA)	Aircraft Factory (SLA)	WarfareBEastAircraftFactory
Airport	Airport	WarfareBAirport
Barracks	Barracks	WarfareBBarracks
Barracks (West)	Barracks (West)	WarfareBWestBarracks
Barracks (East)	Barracks (East)	WarfareBEastBarracks
Camp	Camp	WarfareBCamp
Construction Site (West)	Construction Site (West)	WarfareBWestConstructionSite
Construction Site (West)	Construction Site (West)	WarfareBWestConstructionSite1
Construction Site (East)	Construction Site (East)	WarfareBEastConstructionSite
Construction Site (East)	Construction Site (East)	WarfareBEastConstructionSite1
Crate	Crate	WarfareBCrate
Depot	Depot	WarfareBDepot
Headquarters (West)	Headquarters (West)	WarfareBWestHeadquarters
Headquarters (East)	Headquarters (East)	WarfareBEastHeadquarters
HeavyFactory	HeavyFactory	WarfareBHeavyFactory
HeavyFactory (West)	HeavyFactory (West)	WarfareBWestHeavyFactory
HeavyFactory (East)	HeavyFactory (East)	WarfareBEastHeavyFactory
LightFactory	LightFactory	WarfareBLightFactory
LightFactory (West)	LightFactory (West)	WarfareBWestLightFactory
LightFactory (East)	LightFactory (East)	WarfareBEastLightFactory
Mobile HQ (West) M113	Mobile HQ (West) M113	M113_MHQ_unfolded
Mobile HQ (East) BMP2	Mobile HQ (East) BMP2	BMP2_MHQ_unfolded

Type	Description	Class Name
Hesco Site	Hesco Site	WarfareBHescoSite
Hesco 5x	Hesco 5x	WarfareBHesco5x
Hesco 10x	Hesco 10x	WarfareBHesco10x
Hesco 10x Tall	Hesco 10x Tall	WarfareBHesco10xTall
Sandbag Site	Sandbag Site	WarfareBSandbagSite
Nest	Nest	WarfareBNest
Low Nest	Low Nest	WarfareBLowNest
MG Nest M240 (West)	MG Nest M240 (West)	WarfareBWestMGNest_M240
MG Nest PK (SLA)	MG Nest PK (SLA)	WarfareBEastMGNest_PK
MG Nest_M240 (Resistance)	MG Nest_M240 (Resistance)	WarfareBResistanceMGNest_M240
MG Static M2 (Resistance)	MG Static M2 (Resistance)	WarfareBResistanceM2StaticMG

3.12 - The plant classes

Type	Description	Class Name
Bare tree	Bare tree	AAPL000
Banana tree	Banana tree	AAPL001
Banana tree	Banana tree	AAPL002
Banana tree	Banana tree	AAPL003
Banana tree	Banana tree	AAPL004
Thistles	Thistles	AAPL005
High grasses	High grasses	AAPL006
Cep	Cep	AAPL007
Flowers	Flowers	AAPL008
Brake	Brake	AAPL009
Desert grass	Desert grass	AAPL010
Gras flowers	Gras flowers	AAPL011
Grass	Grass	AAPL012
Long grass	Long grass	AAPL013
White flowers	White flowers	AAPL014
Grass	Grass	AAPL015
Yellow flowers	Yellow flowers	AAPL016
Mushroom	Mushroom	AAPL017
Fly agaric	Fly agaric	AAPL018
Fly agaric	Fly agaric	AAPL019
Mushroom	Mushroom	AAPL020
Flowers	Flowers	AAPL021
Grass	Grass	AAPL022
White flowers	White flowers	AAPL023
Tall conifer	Tall conifer	AAPL024
Tall conifer	Tall conifer	AAPL025
Bush	Bush	AAPL026
Bush	Bush	AAPL027
Desert grass	Desert grass	AAPL028
Mixed grass	Mixed grass	AAPL029
Tree	Tree	AAPL030
Tree	Tree	AAPL031
Leaves	Leaves	AAPL032
Small bush	Small bush	AAPL033
Middle bush	Middle bush	AAPL034
small bush	small bush	AAPL035
Mini bush	Mini bush	AAPL036
Grass bush	Grass bush	AAPL037
Double grass	Double grass	AAPL038

Type	Description	Class Name
Stocks of a trees	Stocks of a trees	AAPL039
Stocks of a trees	Stocks of a trees	AAPL040
Stock of a tree	Stock of a tree	AAPL041
Grass	Grass	AAPL042
Middle bush	Middle bush	AAPL043
Tall bush	Tall bush	AAPL044
Long bush	Long bush	AAPL045
Small bush	Small bush	AAPL046
Long bush	Long bush	AAPL047
Big tree	Big tree	AAPL048
Big tree	Big tree	AAPL049
Big tree	Big tree	AAPL050
Middle tree	Middle tree	AAPL051
Tall conifer	Tall conifer	AAPL052
Tall conifer	Tall conifer	AAPL053
Big leave grass	Big leave grass	AAPL054
Bush tree	Bush tree	AAPL055
Middle palm tree	Middle palm tree	AAPL056
Middle palm tree	Middle palm tree	AAPL057
Middle palm tree	Middle palm tree	AAPL058
Middle palm tree	Middle palm tree	AAPL059
Double palm tree	Double palm tree	AAPL060
Palm bush	Palm bush	AAPL061
Tall palm tree	Tall palm tree	AAPL062
Middle palm tree	Middle palm tree	AAPL063
Tree stump	Tree stump	AAPL064
Tree stump	Tree stump	AAPL065
Conifer bush	Conifer bush	AAPL066
Small conifer	Small conifer	AAPL067
Broad conifer	Broad conifer	AAPL068
Tall conifer	Tall conifer	AAPL069
Birch tree	Birch tree	AAPL070
Tree	Tree	AAPL071
Tree	Tree	AAPL072
Tree	Tree	AAPL073
Small tree	Small tree	AAPL074
Middle decoration tree	Middle decoration tree	AAPL075
Small tree	Small tree	AAPL076
Small tree	Small tree	AAPL077
Tree	Tree	AAPL078
Small tree	Small tree	AAPL079
Tree	Tree	AAPL080
Small tree	Small tree	AAPL081
Small bush	Small bush	AAPL082
Willow	Willow	AAPL083
Birch tree	Birch tree	AAPL084
Birch tree	Birch tree	AAPL085
Bush tree	Bush tree	AAPL086
Tree	Tree	AAPL087
Bush tree	Bush tree	AAPL088
Small bush	Small bush	AAPL089
Tall decoration tree	Tall decoration tree	AAPL090
Avenue tree	Avenue tree	AAPL091
Avenue tree	Avenue tree	AAPL092
Branchwood	Branchwood	AAPL093
Branchwood	Branchwood	AAPL094
Branchwood	Branchwood	AAPL095
Branchwood	Branchwood	AAPL096

3.13 - The rock classes

Type	Description	Class Name
Clutter Stone Small	Clutter Stone Small	AAO000
Granite stone	Granite stone	AAO001
Sandstone (big)	Sandstone (big)	AAO002
Sandstone (big)	Sandstone (big)	AAO003
Limestone (big)	Limestone (big)	AAO004
Limestone (big)	Limestone (big)	AAO005
Sandstone (big)	Sandstone (big)	AAO006
Limestone (middle)	Limestone (middle)	AAO007
Limestone (middle)	Limestone (middle)	AAO008
Sandstone (big)	Sandstone (big)	AAO009
Limestone (middle)	Limestone (middle)	AAO010
Limestone (middle)	Limestone (middle)	AAO011
Sandstone (middle)	Sandstone (middle)	AAO012
Limestone (middle)	Limestone (middle)	AAO013
Limestone (middle)	Limestone (middle)	AAO014
Sandstone (middle)	Sandstone (middle)	AAO015
Limestone (middle)	Limestone (middle)	AAO016
Limestone (middle)	Limestone (middle)	AAO017
Sandstone (little)	Sandstone (little)	AAO018
Granite stones	Granite stones	AAO019
Granite stones	Granite stones	AAO020
Limestone rock (little)	Limestone rock (little)	AAO021
Limestone rock (big)	Limestone rock (big)	AAO022
Limestone rock (big)	Limestone rock (big)	AAO023
Limestone rock (big)	Limestone rock (big)	AAO024
Limestone rock (big)	Limestone rock (big)	AAO025
Sandstone (little)	Sandstone (little)	AAO026
Sandstone (little)	Sandstone (little)	AAO027
Sandstone (little)	Sandstone (little)	AAO028
Sandstone (little)	Sandstone (little)	AAO029
Sandstone (little)	Sandstone (little)	AAO030
Sandstone (little)	Sandstone (little)	AAO031
Sandstone (middle)	Sandstone (middle)	AAO032
Sandstone (little)	Sandstone (little)	AAO033
Sandstone (little)	Sandstone (little)	AAO034
Sandstone (little)	Sandstone (little)	AAO035
Sandstone (little)	Sandstone (little)	AAO036
Sandstone (little)	Sandstone (little)	AAO037
Sandstone (middle)	Sandstone (middle)	AAO038
Sandstone (middle)	Sandstone (middle)	AAO039
Sandstone (middle)	Sandstone (middle)	AAO040
Sandstone (middle)	Sandstone (middle)	AAO041
Sandstone (middle)	Sandstone (middle)	AAO042
Limestone rock (big)	Limestone rock (big)	AAO043
Limestone rock (big)	Limestone rock (big)	AAO044
Sandstone (little)	Sandstone (little)	AAO045
Limestone (middle)	Limestone (middle)	AAO046
Sandstone (little)	Sandstone (little)	AAO047
Sandstone (little)	Sandstone (little)	AAO048
Sandstone (little)	Sandstone (little)	AAO049
Sandstone (little)	Sandstone (little)	AAO050

3.14 - The sign classes

Type	Description	Class Name
Cyclist ahead!	Cyclist ahead!	AASI012
Walkers ahead!	Walkers ahead!	AASI015
Walkers ahead! (old)	Walkers ahead! (old)	AASI016
Double bend L ahead! (old)	Double bend L ahead! (old)	AASI019
Double bend L ahead!	Double bend L ahead!	AASI020
Double bend R ahead! (old)	Double bend R ahead! (old)	AASI021
Double bend R ahead!	Double bend R ahead!	AASI022
Right of Way (old)	Right of Way (old)	AASI025
Right of Way End (old)	Right of Way End (old)	AASI026
Right of Way End	Right of Way End	AASI027
Right of Way	Right of Way	AASI028
Military Sign (SLA/RACS)	Military Sign (SLA/RACS)	AASI169
Military Sign (SLA)	Military Sign (SLA)	AASI181
Military Sign (RACS)	Military Sign (RACS)	AASI186
Picnic area	Picnic area	AASI189
Camping ground	Camping ground	AASI190
Not attached curb ahead!	Not attached curb ahead!	AASI195
Rustle shoot ahead! (old)	Rustle shoot ahead! (old)	AASI196
Rustle shoot ahead!	Rustle shoot ahead!	AASI197
Cross-way ahead! (old)	Cross-way ahead! (old)	AASI198
Right of way ahead! (old)	Right of way ahead! (old)	AASI199
Right of way ahead!	Right of way ahead!	AASI200
Cross-way ahead!	Cross-way ahead!	AASI201
Air traffic ahead! (old)	Air traffic ahead! (old)	AASI204
Air traffic ahead!	Air traffic ahead!	AASI205
Ahead!	Ahead!	AASI259
First aid (old)	First aid (old)	AASI260
First aid	First aid	AASI261
Gravel ahead!	Gravel ahead!	AASI268
Roadway repair service (old)	Roadway repair service (old)	AASI273
Roadway repair service	Roadway repair service	AASI274
Rockfall ahead (old)	Rockfall ahead (old)	AASI275
Rockfall ahead	Rockfall ahead	AASI276
Parking place (old)	Parking place (old)	AASI277
Parking place	Parking place	AASI278
Ahead!	Ahead!	AASI283
Direction panel L	Direction panel L	AASI284
Direction panel R	Direction panel R	AASI285
Road works ahead!	Road works ahead!	AASI286
Crosswalk ahead!	Crosswalk ahead!	AASI287
Right of way ahead! (old)	Right of way ahead! (old)	AASI288
Right of way ahead!	Right of way ahead!	AASI289
Gas station (old)	Gas station (old)	AASI290
Gas station	Gas station	AASI291
Straightforward	Straightforward	AASI292
RoadCone	RoadCone	RoadCone
RoadBarrierlong	RoadBarrierlong	RoadBarrier_long
RampConcrete	RampConcrete	RampConcrete

3.15 - Getting weapon and magazine types displayed

By using the following Syntax, one will have the possibility to get the different weapon and magazine types displayed as an on screen information text. To do this for the weapon types, just use:

```
hint format ["%1", weapons this];  
hint format ["%1", weapons Name];
```

and for the magazine types:

```
hint format ["%1", magazines this];
```

3.16 - Getting fired type

One also has the possibility to get additional information - such as which unit has fired with what kind of ammunition - displayed on the screen. The following details will be displayed:

```
Name of the unit  
The weapon type  
The bullet type  
The way of firing (single fire/burst)
```

Once the unit in the game has fired its weapon, this text will appear on the screen:

```
[Name, "M4AIM", "M4AIM", "Single", "B_556x45_Ball"]
```

To do this it's recommended to use an event handler which will be used to define the syntax in the init. line of a unit as shown in the example below:

```
this addEventHandler ["Fired", {hint format ["%1", _this]}]
```

All of this can be used with names for external scripts as well:

```
Name addEventHandler ["Fired", {hint format ["%1", _this]}]
```

3.17 - Does unit have a weapon?

Some Situation requires the checking whether a special unit has a special Weapon. This information can also be used as condition for further situations or other things.

To do this use the following Syntax:

```
Player hasWeapon "M4" or in a Script ? Player hasWeapon "M4"
```

Two Syntaxes as further examples:

?! (Player hasWeapon "M4") : hint "The Player has lost his Weapon!"

? (Player hasWeapon "M4") : hint "The Player has lost his Weapon!"

The first Syntax will cause a hint which will inform the user that the Player character has lost his weapon while the second Syntax informs about the fact that the Character got his Weapon back again.

3.18 - Primary or secondary weapon of a unit

Even the primary weapon of a unit can always get asked or used as condition for something's as well. Actually the same things can be realized by using the hasWeapon command. Some examples can be seen below:

Getting primary or secondary weapon displayed:

hint format ["%1", primaryWeapon Player];

hint format ["%1", secondaryWeapon Player];

Primary or secondary weapons used as condition:

? (primaryWeapon Player != "M4") : hint "The Player has no M4!"

? (primaryWeapon Player == "M4") : hint "The Player has no M4!"

? (secondaryWeapon Player != "Stinger") : hint "The Player has no M4!"

3.19 - Does unit have ammunition?

The following Syntax will inform the user whether the unit has ammunition left or not. Doesn't he still have a Magazine, so the Variable will be set to true. And this fact can be used as a further condition for something else's.

!(someAmmo Player) : hint "The Player hasn't any ammunition left!"

?(someAmmo Player) : hint "The player does still have ammunition!"

hint format ["%1", someAmmo Player];

This command will prohibit a unit to reload the weapon if the magazine is empty:

Name enableReload false

3.20 - Creating mines

Mines can be simply created at any position on the map by using the following Syntax

Mine = createMine ["MineMine", position player, [], 0]

3.21 - Creating weapons and magazines

It's not possible to create weapons or magazines just for fun. To do this a special option called Weapon holder command is required. A Weapon holder is similar to a Ammo Box, the only difference is that this weapon holder is invisible. This one can be equipped now with Weapons and Magazines by using the same commands which are used to edit a Ammo Box. Once the Weapons and Magazines have been created they will lie on the ground and get picked up by the Player.

Now one is free decide to select a position where the Weapons and or Magazines have to be created. For example on a **XYZ** Position or at the Position of an **Object**.

One can see some examples below:

```
Weapon1 = "weaponHolder" createVehicle getpos Object1
```

```
Weapon1 = "weaponHolder" createVehicle position Player
```

```
Weapon1 = "weaponHolder" createVehicle [x,y,z]
```

Once the Weapon holder has been created so one only needs to allocate him the weapons and magazines. This can be done by using the already known AddCargo command.

```
Weapon1 addMagazineCargo ["10Rnd_127x99_M107",2];
```

```
Weapon1 addWeaponCargo ["m107",1];
```

It's also possible to align the Weapon holder in a special direction or lift him up in a special height. For example if one wants to get a weapon created on a Ammo box or a table or something's else.

To define the directions use the known SetDir-Syntax:

```
Weapon1 setDir Value
```

And as expected use the setpos getpos command to define the height:

```
Weapon1 setPos [getPos Weapon1 select 0,getPos Weapon1 select 1,Value]
```

Ingame it would look like this:



3.22 - Getting weapon direction view displayed

By using the following Syntaxes its possible to get the direction of a weapon displayed. This information can be used for a further condition. This possibility will mostly be used for Vehicles or tanks or something's similar to get the direction of the respective Weapon.

The following Syntaxes will display the XYZ values:

```
hint format ["View Direction: %1", Name weaponDirection "M56"]
```

```
hint format ["%1", Name weaponDirection primaryWeapon Name]
```

The following example will display the height only:

```
hint format ["%1", Name weaponDirection "M56" select 2]
```

```
hint format ["%1", Name weaponDirection primaryWeapon Name select 2]
```

This explains as follows. **X** requires **select 0**, **Y** requires **select 1**, and **Z** requires **select 2**.

If one wants to use this possibility within a script, one needs to define this syntax as follows:

```
_dir = _unit weaponDirection "ClassName" select 0  
? _dir <= Value : hint "Wrong Fire Direction!"
```

Sectoring the XYZ-Values

```
_Direction = Name weaponDirection primaryWeapon Name  
_xDir = _Direction select 0  
_yDir = _Direction select 1  
_zDir = _Direction select 2
```



Chapter 4

- The Mission -

After you have learned the user interface, the files and the weapons in the first 3 chapters, we'll enter a new section now - The mission design. Here you'll learn how to start a mission, define the targets, the fulfilment measures assessed, and finally finishing the mission successfully.

4.1	The mission name	97
4.2	The mission start	97
4.3	The mission accessories	98
4.4	The mission appraisal	99
4.5	The mission targets	99
4.6	Finishing a mission	101
4.7	Saving a mission	



4.1 - The mission name

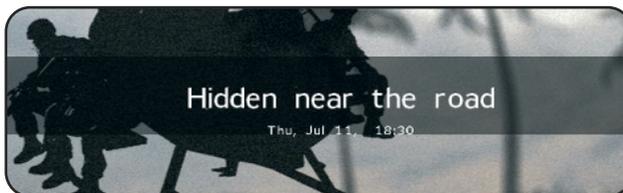
If the user creates a new map, he should name the mission in the Intel menu. This is not quite necessary but useful, because this makes the mission display with its real name. If one does not do that, the mission would be displayed with its island name. The mission name will look like this:



The **example mission** is called **Beispielmission**. The first one is displayed by its real name because it has been defined in **Intel** menu. The second one hasn't been defined, so you can see what happens to the mission name in the mission selection menu.

4.2 - The mission start

The game offers the possibility to display time of day and a headline in an individual style while the mission is loading. The text which shall be displayed is up to the user but it shouldn't be too long. To make the time of day and the desired text lines display, the `description.ext` needs to be edited. If this file doesn't exist in your mission folder, it has to be created. (You'll get more information about this in **Chapter 2.3** - The `Description.ext`.)



To predefine the text and the time of day just enter following syntaxes in the head of the `description.ext`:

```
onLoadIntro = Mr-Murray proudly presents  
onLoadMission = Convoy Attack
```

```
onLoadIntroTime = true respective false or 1 resp. 0  
onLoadMissionTime = false
```

If one doesn't want both things to be displayed, the values just have to be set to **0** or **false** behind the quotes and the text underneath the clock will no longer be displayed.

It's also possible to define the text in the stringtable.csv, that would look like this:

```
onLoadIntro = $STR_Missionstart
```

You can find more information about the stringtable.csv in **Chapter 2.4**.

4.3 - The mission accessories

The user has the possibility to determine whether certain mission accessories are to be enabled in the mission or not. To do this, further things need to be defined in the description.ext as well.

To display each used accessories the number **1** or **0** respective **false** or **true** are needed again. As follows the list of the orders:



- ShowGPS = 1;** - GPS
- ShowCompass = 1;** - Compass
- ShowRadio = 1;** - Radio
- ShowMap = 1;** - Map
- ShowNotePad = 1;** - Briefing
- ShowWatch = 1;** - Clock
- ShowDebriefing = 1;** - Debriefing

Since the Version **1.08** it's possible now to disable these components completely, without an entry within the description.ext. That wasn't able in the earlier Arma® and OFP® Versions. To do this, just set the respective value on **true** or even **false**.

ShowMap true

4.4 - The mission appraisal

As in most games, the player can receive points for reaching targets. This is possible in ArMA® as well. To enable that option just define the necessary commands in the description.ext. The number of the receiving points is variable and can be freely defined by the user. The respective part in the description.ext looks like as follows:

minScore=200	- The least scores
avgScore=3000	- The middle scores
maxScore=6000	- The highest scores

The player will automatically receive points for each enemy unit killed. If one wants the player to receive extra points for completing a special objective at some point in the mission, the following syntax is needed:

Player addRating Value

It's also possible to remove points from the player, for example, if the player destroys a facility which he is actually supposed to protect. To do this use the syntax above and add a - (minus) in front of the value only.

Player addRating -Value

If one wants to receive a point status when the player has received a certain number of points, (for example, to end the mission) then one only has to place a trigger on the map with following conditions: (**Axis a/b = 0**), and write in the condition line:

Rating Player > Value or **Rating Player >= Value**

Select **End1** out of the **Types**. If the player reaches this value, the mission will end and the briefing will display the results.

4.5 - The mission targets

The most important things of a mission are the targets. No targets - no mission; so the targets need to be defined early.

The mission targets can be defined as explained in **Chapter 2.13 - The Briefing.html**, and if one likes, it's also possible to hide them as explained in **Chapter 2.5 - The Init.sqs**.

Hidden targets have to be defined and configured on the map just as the visible targets are. Hiding a target means that the target is not visible for the player in the briefing and on the map, but when the player has accomplished a target, it will appear on the map.

Example mission

The player reads in his orders that he has to "hit-and-run this village". The second order, "destroy the ammunition truck", is still invisible because of the entry in the Init.sqs. If the village has been cleared of all enemy units, the first objective will be marked with a green check mark and the second target will become visible.

To do this just set a trigger on the map right over the respective village. Conditions: **Axis (a/b)** 300, **onActivation** (the side which has to protect the village), and **not present**. The trigger will execute now when the side which protects the village is no longer alive (not present).

The necessary commands which are to be executed when the village is free of enemy units, have to be entered in the **onActivation** line. "**Check target 1**" and "**make target 2 visible again**" are a part of this command. It's recommended to add a hint to the command to give some information to the player. So use following syntax:

"1" ObjStatus "Done"; "2" ObjStatus "Visible"; hint "Missionsplan updated!"

The user has to set a marker directly over the village on the map and name it "**TargetX**". The cross-hair will move to that marker if the player is clicking the link "**This village**" in the briefing.

You can also check **Chapter 2.13** to get further information about the necessary commands which are to be used in the briefing.html.

```
<p>
<a name="OBJ_1"></a>
Capture <a href="marker:TargetX">this village</a>!
</p>
<hr>
<p><a name="OBJ_2"></a>
Destroy the ammunition Truck!
</p>
<hr>
```

The necessary entry in the Init.sqs, which is used to hide the second mission target, needs to be defined as "**2" ObjStatus "Hidden**".

The following are the different commands which are used for each mission status:

- | | |
|----------------|--|
| Hidden | - The mission target will be hidden |
| Visible | - The mission target will be visible again |
| Active | - The mission target is active |
| Done | - The mission target is done |
| Failed | - The mission has been failed. |

4.6 - Ending the mission

When the player has reached all targets then the mission has been finished. There are several possibilities to end a mission of course. Those endings are up to the story, the targets, and of course, the mission. Here you can see some examples of how to end a mission:

Checking the status of a unit within a local view

One can see several units in the image to the right. The APC has to be destroyed for the trigger to be executed. This one has been connected with a trigger by pressing the 2 key. The trigger has been defined as follows:

Axis a/b: 50
Type: End #1
Activation: not present

In local view, the mission will end if the APC is destroyed or if it leaves the trigger area. What if the mission should end if the APC is destroyed, but not when the APC leaves the trigger area? This is when a global view should be used.



Checking the status of a unit within a global view

To do this, one has to set a trigger again but this trigger does not have a bordered area. So we have different settings compared to the local view:

Axis a/b: 0
Type: End #1
Condition: ! (alive Tank1)

Our APC has been renamed to **Tank1** and the trigger was set up to the global view with both **Axis a** and **b** set to **0**. You can set the trigger on any place on the map, because it will now check the whole map to find out if **Tank1** is still alive or not.



The APC can move on the whole map now and the mission will end only if the APC has been destroyed. The local view is variable of course and the value which has to be defined in both Axis boxes is up to the story and what the creator of the mission wants to do. Global view gives more flexibility to the mission. Every trigger which has been defined with a local view generates a circle around itself. If the user is using the global view, no circles will be visible and this gives a better overlook to the user while creating the mission.

Covered on several units out of the global view

The same guidelines used in the **global view**, only with additional commands in the conditions line:

! (alive Tank1) AND ! (alive MG1) AND ! (alive Soldier1)

The **AND** connects every single condition with each other. The mission will be accomplished if the targets named **Tank1**, **MG1**, and **Soldier1** have been destroyed.

The mission is finally accomplished when the trigger area is free of enemy units

One can see several units in the picture below. The objective is to eliminate all enemy units in that village (trigger area). It doesn't make any difference whether the enemy units have been killed/destroyed or have run away, out of the respective trigger area.

Define the trigger as follows:

Axis a/b:	50
Type:	Ende #1
Activation:	EAST Nicht vorhanden



Within ArmA® Version 1.05 its possible to adjust a trigger so that it will execute when all enemy units have been eliminated (resp. are not present) and also one friendly unit is still left in that trigger area. To do this, just define the side which has to conquer that area. The definition has to be done in **Activation**.

Variable covered ending of the mission

The variable covered end is not much more difficult to realize, but up to the size more extensively. The following example shall prove a better explanation. Three different trigger areas have to be cleared of enemy units. The mission shall end only if all three areas, in this example three villages, are free of the enemy. So adjust the triggers as follows:

Trigger 1 (Area 1):

Axis a/b: 100
Activation: EAST (not present)
onActivation: Target1=true

Trigger2 (Area 2):

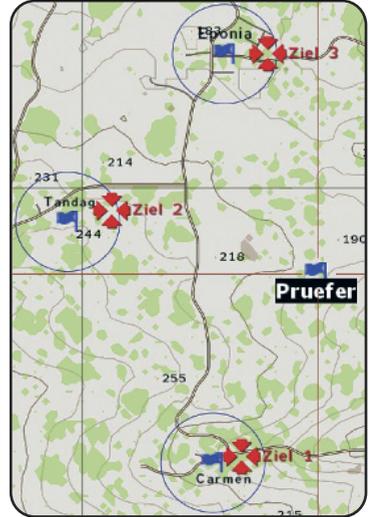
Axis a/b: 150
Activation: EAST (not present)
onActivation: Target2=true

Trigger3 (Area 3):

Axis a/b: 100
Activation: EAST (not present)
onActivation: Target3=true

Trigger 4 (Examiner):

Axis a/b: 0
Type: End #1
Condition: Target1 AND Target2 AND Target3



As one can see, a variable (**Target1, Target2, Target3**) has been set to true for all three triggers. The 4th one contains this as a condition to execute. If this trigger is executed, the mission will be finished. One can also use the timeout function to configure a much more flexible ending.

4.7 - Saving a mission

This command enables the user to save the current mission status while the mission is already running. The user has the possibility to place save-points on the map which will save the mission when the player has reached a specific location or has completed certain mission objectives. To set place such a point on the map just use the following Syntax:

Savegame

If one is selecting "Try again" out of the menu once one has been killed, the player will restart right at the point where the mission was last saved. If one doesn't want to allow the saving of the mission, just define the command **saving = 0** in the Description.ext.

Chapter 5

- Mission Accessories -

This chapter offers lots of additional accessories which can be helpful to the mission. It is the most complex chapter of the whole book because it contains 85 subsections. You may find all the answers to your questions which you've ever searched for, and also lots of other nice information, which might be helpful in your mission.

5.1	Empty or locked vehicle	106
5.2	Driver/Passenger of a vehicle	106
5.3	Unit is not allowed to enter a vehicle	106
5.4	Unit in vehicle?	107
5.5	Vehicle only moves when unit has entered	107
5.6	Group already in vehicle when the mission begins	108
5.7	Let a unit get in and get out of a vehicle	108
5.8	Speed of a unit	108
5.9	Display the speed of a unit	108
5.10	Unit keeps standing	109
5.11	Getting a unit started	109
5.12	Unit is moving to its destination	110
5.13	Running patrol, drive or fly	110
5.14	Escape behaviour of a unit or a group	110
5.15	Moving units, objects, triggers and markers	111
5.16	Placing objects higher or lower	111
5.17	The height of a unit	112
5.18	Accurate helicopter landing	112
5.19	Unit is moving into a building	112
5.20	Unit is leaving / joining group	113
5.21	Assigning a target to a unit	113
5.22	Unit turns to another Unit	114
5.23	Unit is selecting weapon	114
5.24	Inflict damage or heal a unit	114
5.25	Defining a death zone	115
5.26	Checking of an area	115
5.27	Bring about a certain behaviour of a unit in an area	115
5.28	Save or load a unit status	116
5.29	Degree of familiarity of a unit	117
5.30	Friendly Enemy	117
5.31	Friendly Forces	118
5.32	The Alert	119
5.33	Dead as condition	120
5.34	Distance of two units or objects	120
5.35	Allocate a flag to a flag staff	120
5.36	Burning fire	121
5.37	Add or remove switchable units	121
5.38	Read out and display player side, - name, -type	121

5.39	Opress player input	121
5.40	Force the map on the screen	121
5.41	Adjusting distance of view	122
5.42	Adjusting the weather	122
5.43	Adjusting date and time of day	123
5.44	Slow motion or time sprint	123
5.45	Generating units and objects	124
5.46	Generate flares, smoke and explosions	126
5.47	Delete units and objects	127
5.48	Adjusting radio menu	127
5.49	Allocate a call-sign to a group	128
5.50	Send a radio message	129
5.51	Creating sound	129
5.52	Using own sounds	130
5.53	Set identity	134
5.54	Mimics	135
5.55	The action command	136
5.56	The animation command	139
5.57	Disable AI units	144
5.58	SetVelocity	144
5.59	The information text	144
5.60	Units keeps lying or keeps standing	144
5.61	Using ID's	145
5.62	Placing units inside of a building	148
5.63	Unit is moving to desired house position	153
5.64	Getting position displayed	153
5.65	The Eventhandler	155
5.66	Different text displays	157
5.67	Stringtable Basic Values	158
5.68	Create waypoints	159
5.69	Create trigger	160
5.70	Create marker	162
5.71	All about vehicles	165
5.72	Create a light source	167
5.73	Create dust	167
5.74	Create smoke	168
5.75	Create fire	169
5.76	Assigning ranks	171
5.77	Unit using Binoculars	172
5.78	Assigning a unit to a vehicle seat	172
5.79	Allocate a unit to a team	173
5.80	Unit is giving out a command	174
5.81	Has a unit recieved damage?	174
5.82	The air traffic	175
5.83	Decrease grass details	176
5.84	Place sloped objects	176
5.85	Lock or unlock missions	177
5.86	Empty searchlight with light	177

5.1 - Empty or locked vehicle

To place an empty vehicle on the map just press the **[F1]**-Key and double click on the map. When the menu appears one has to select "**Empty**" out of the Side drop-down menu. Then the **unit** and **vehicle type** has to be selected out of the **Class** sub menu. Then press the **OK** button.

If the vehicle needs to be locked or isn't intended to be used, just select **locked** in **vehicle status**. If the user wants to make the vehicle usable later, just use following syntax:

- Name lock true** - The vehicle becomes locked
- Name lock false** - The vehicle becomes unlocked

5.2 - Driver / Passenger of a vehicle

With the following orders it's possible to "beam" a unit to an arbitrary position of a vehicle. Those syntaxes only need to be defined in the init box of each unit or in external scripts as well. If those syntaxes are to be defined in the init box of a unit, the variable **this** can be used instead of a **Name**.

- Name moveInDriver Fhz1** - Driver of the vehicle
- Name moveInCargo Fhz1** - Passenger in a vehicle
- Name moveInCommander Fhz1** - Commander of a vehicle
- Name moveInGunner Fhz1** - Gunner of the vehicle
- Name moveInTurret Fhz1** - Gunner of the vehicle (MG)
- Name moveInTurret [Fhz1,1]** - Second Gunner of the vehicle (MG)
- Name moveInCargo [Fhz1,3]** - Passenger on any cargo position

5.3 - Unit is not allowed to use a vehicle

This syntax has to be used if the user doesn't want to make a vehicle usable for a certain unit. To do this just enter the following syntax into the init box of the vehicle.

[Name1, Name2, Name3] allowGetIn false

Now the units called **Name1**, **Name2** and **Name3** are not allowed to enter the vehicle. If the user wants to make the vehicle usable for these units again, just set the variable on **true** again.

[Name1, Name2, Name3] allowGetIn true

5.4 - Unit in vehicle?

Sometimes it might be useful to check whether a unit is still inside a vehicle or not. This can also be used as condition to activate a trigger or a script.

To check whether **Name** is sitting in a vehicle just use this syntax:

Name in VehicleName
Player in (crew VehicleName)

and define it as condition of a checking trigger. If the unit is getting into the car now, this trigger will be activated.

To use this one in a script, just define it this way:

? Name in VehicleName

To test whether a unit is no longer sitting in a vehicle just use this syntax:

not (Name in Vehiclename)

! can be used instead of **NOT**. The conditions are the same.

5.5 - Vehicle only moves when unit has entered

Maybe you have a vehicle on the map which already has its waypoints allocated and only should start to move when a special unit has entered, for example the player. It doesn't matter which kind of vehicle is used; it works for cars, trucks, ships and also helicopters.

To make it work, just enter following syntax in the condition field of a trigger, or a waypoint:

Name in VehicleName or **Vehicle Name == VehicleName**

Using with groups

That above syntax works for groups as well. The vehicle has to wait until all units of the group have entered the vehicle. In the following syntax the driver is included, which is already sitting in the vehicle. One has to calculate the whole group + the driver.

To make it work just enter following syntax in the condition field:

count crew Truck1 >= 10 or **count crew Truck1 == 10**

5.6 - Group is already in vehicle when the mission begins

If the user wants to create a mission where a group is already sitting in a vehicle, one has to enter the following syntax into the init line of the group leader:

```
{_x moveInCargo Heli1} forEach Units Group this  
or  
{_x moveInCargo Heli1} forEach Units Grp1
```

5.7 - Let a unit get in and out of a vehicle

To do this just place an empty vehicle and a soldier on the map. Then, give a waypoint to the soldier and place it directly on the vehicle. Then select "Get in" of the Type menu. Place another waypoint on the map and select "Get out". The unit will get into the car, move to its destination and will get out.

One has the possibility, of course, to do this by using a syntax as well:

```
unAssignVehicle Vehicle1 - Unit is leaving vehicle  
{unAssignVehicle _x} forEach units Group1 - Units are leaving vehicle
```

The group leader is giving out the command to his group to get out of the vehicle.

5.8 - Speed of a unit

The speed of a unit can be defined in a waypoint in the speed menu or you can also use the following syntax:

```
Name setSpeedMode "Full" - Available in Auto, Limited, Normal, Full  
Name forceSpeed 120 - Value in km/h (imp. Not Miles/h)  
Name limitSpeed 60 - Value in km/h  
Name1 setSpeed getSpeed Name2 - Name1 receives Speed from Name2
```

5.9 - Display the speed of a unit

It's always possible to get the speed of a unit displayed as a hint. One could use this information for further things like conditions i.e. To do this use following Syntax.

```
hint format ["Speed: %1" , speed Name]  
? speed Name > 30 : hint " You are driving to fast!"
```

5.10 - Unit keeps standing

The command **dostop this** in the init line of a unit, will keep a unit in its position where it has been placed. This enables one to avoid the units moving back into the formation close to their leader after the mission has been started.

Using with a team

This option is quite useful if the player is a leader of a group and doesn't want his group following him when he is changing his position. So it is possible to allocate special positions to the units. The units will keep their positions if the **dostop this** command has been defined in the init line of each unit. Furthermore, it's important to set the option "**None**" in "**Special**" for the respective unit..

Using with enemy units

This order is very useful, because it is possible now to spread out enemy units on the terrain. It is quite important to make sure that the option "none" in "special" has been set. If the enemy gets attacked now, they will cover on their positions.

5.11 - Getting a unit started

In this example we have a group which has to move to its predefined position if a trigger or waypoint is executed by the player. Enter the following command in the init line of the group leader.

this stop true

To make it work you also have to enter

Name stop false

in the init line of the player character. The group will now move to its waypoint. The command "this" has been used in the syntax above, because this command has been defined in the init line of the group leader. The second command is using a name, so it's necessary to allocate a name to the group leader.



5.12 - Unit is moving to its destination

In ArmA®, it's also possible to send a unit to a special place on the map without supporting them with waypoints beforehand. There are several possibilities available:

- Using objects : **Name doMove getPos Name**
- Using ID's : **Name doMove getPos (Object ID)**
- Using coordinates: **Name doMove [X,Y,Z]**
- Using markers : **Name doMove getMarkerPos "MarkerName"**

If one wants to make a whole group move to a special position which has been defined with one of the possibilities shown above, you first have to define that order without the **do** of **doMove**. That's needed because the leader might move alone to its predefined position and his group would follow only when he has reached his destination.

An example for a syntax which is used for objects:

Name move getPos Name or **Leader Name move getPos Name**

There are further interesting examples shown in the **Chapter 6.6 - The Map Click** and **Chapter 6.2 - The GPS-System**.

5.13 - Running patrol, drive or fly

If the user wants to make a patrol, running or driving around a base in an unending-loop, place a unit on the map and give it several waypoints. The last waypoint has to be placed directly into the area of the first one, then select **Cycle** out of the **Type** drop-down menu.

5.14 - Escape behaviour of a unit or a group

Maybe the user wants to edit a mission where the enemy units are escaping when a predefined value has been reached. The enemy units will run away and hide somewhere on Sahrani, but be careful, sometimes they reform and attack again from another direction.

All values between **0** and **1** can be used, so even the decimal values will work. The value **0** means no and **1** means maximum escaping behaviour. If the Syntax has to be written in the init line of a squad leader:

this allowFleeing 0.8

The command will effect the whole group. It's much more dynamic if this Syntax is used with the random function.

this allowFleeing (random 0.8)

5.15 - Moving units, objects, triggers and markers

It's possible to move units, objects, triggers and markers while a mission is running. We can say as well, that those parameters will get beamed to another position on the map. To do this, it's recommended that the object which has to get moved has a name, then it's possible to move objects by using following syntaxes:

Using an object:	Name setPos getPos Name
In front, behind, besides an Object	Name2 setpos Name1 modelToWorld [0,3,0]
Using an ID:	Name setPos getPos (Object ID)
Using coordinates:	Name setPos [X,Y,Z]
Using markers:	Name setPos getMarkerPos "Marker1"
From marker to marker:	"M1" setMarkerPos getMarkerPos "M2"
From marker to object:	"Marker1" setMarkerPos getPos Name
Using a vehicle:	Name setPos getPos vehicle Player
Using a vehicle II:	"Marker1" setMarkerPos getPos vehicle Player

A related order which contains the definition of the altitude of an object, in this case is the value **10**.

```
Name1 setPos [(getPos Name2 select 0),(getPos Name2 select 1),10]
```

Name1 will be moved to the position of **Name2** with a height of **10** meters. If one wants to move a whole group from one position to another one, so use the Syntax below:

```
{_x setpos getPos Name} foreach units Group1
```

The following Syntax enables one to move a unit or an Object to a random position. The Syntax consists out of the values of the XYZ – coordinates and the marker positions. The Engine will recalculate the random position by using this information.

```
Name setVehiclePosition [[1000,2000], ["Marker1", "Marker2", "Marker3"], 0]
```

5.16 - Placing objects higher or lower

Nearly all objects which were placed on the map can be set higher or lower. Units and vehicle are not fully supported, so it's not possible to move them down into the terrain, but one can set them higher if it's needed. It's possible to place soldiers in houses or roofs. If a unit gets set on the map a few meters over the ground, that unit will fall to the ground if there's no house or other item below it. That's because of the gravity-conditional, which is simulated very well in ArMA®. The only exception to this are static objects. Static objects, like sandbags, are not subject to the gravity, and would float at the adjusted height. To lift up or lower an object on the map use following syntax:

```
this setPos [(getPos this select 0),(getPos this select 1),10];
```

The object where this syntax has been defined would be displayed now at a height of 10 meters. ArMA® doesn't support this function until version 1.8!

The contents of the Array [] have to be defined as follows. The first () contain the position of the object in **X-direction**. The second () contain the **Y-direction** and the numbers behind the () are similar to the **Z-direction** of the object. If one wants to move **Name1** to **Name2** than both of the **this** have to be replaced with their respective names. Maybe a marker shall move to another position after a target has been destroyed.

Use the following syntax:

```
Name1 setPos [(getPos Name2 select 0),(getPos Name2 select 1),10];
```

Name1 has now been moved to the position of **Name2**, at a height of **10** meters.

Along with the setPos and getPos orders, setPosASL and getPosASL are also available which are used to define the height of an object over the sea level.

5.17 - The height of a unit

It's possible to define several levels of height for a flying unit. One can do this by using the waypoints of the respective unit or even through the use of scripts. The following syntax has to be used:

```
Name flyInHeight 120
```

5.18 - Accurate helicopter landing

If one places a waypoint on the map on the position where the helicopter needs to land, the helicopter will land as close as possible to the waypoint's position. But there's a solution to make a helicopter land precisely at a predefined point on the map. First one has to place a Heli-H on the spot which the helicopter is to land. It doesn't really make any difference whether that H is visible or invisible. The waypoint has to be placed directly on the H, and unload or get out, is to be selected out of the type transport drop-down menu. The helicopter will land now at this exact position. A further possibility is given by using the syntax below:

```
HeliName land "PositionName"
```

5.19 - Unit is moving into a building

To make a unit move into a building, it's necessary to know whether the building allows units to enter or not. To do this just move the cross-hair over a building a wait until the description of the building to appear. If that building is able to be entered by a unit, give the unit a waypoint directly on the building. Now, one can select one of several **positions** inside the building which are selectable **out of the House** option from the waypoint menu. The unit will move into the predefined position after the respective waypoint has been executed.

5.20 - Unit is leaving / joining group

It's possible to make a unit leave or join his / another group. To do this just use a waypoint and go back to **Chapter 1.5 - Adding Waypoints**, and take a look in the subsection, **join and lead**. The other method for a unit to join or leave a group is to use a script. To make a unit leave his group just use following syntax:

[Name1] join grpNull

by using the following syntax one can make a unit join another group:

[Name1] join Name2

At first, **Name1** was allocated to a **non existing** group and then to group **Name2**. For use with several units use following syntax:

[Name1, Name2, Name3] join grpNull

and then:

[Name1, Name2, Name3] join Name4

It's also possible to make an enemy unit join a friendly group, for example, if a Russian unit needs to fight on the American side against his own troops.

5.21 - Assigning a target to a unit

Here, one can define several target possibilities for units. Whether a units designated target is friendly or enemy makes no difference. The syntaxes which has to be used are as follows:

Name1 doTarget Name2

Name1 turns to **Name2**

Name1 commandTarget Name2

Name1 orders an unit to aim on **Name2**

Shooting

Name1 doFire Name2

Name1 is shooting on **Name2**

Name1 doFire ObjNull

Name1 is no longer shooting any target

Name1 fire "Weapontype"

Name1 is shooting blind

Name1 commandFire Name2

Name1 is ordering a unit to shoot at **Name2**

5.22 - Unit turns to another unit

If one wants to make a unit look in a special direction or to another unit, just use following syntax. But there are also more possibilities available. Using the first option, the unit turns with its whole body to the respective direction while the other option enables the unit to beam into the desired direction.

- Name1 setDir 160** - Name will be moved into direction of 160
- Name1 setFormDir 160** - Name turns to direction 160
- Name1 setDir getDir Name2** - Name gets Azimuth of Name2
- Name1 doWatch Name2** - Name1 is looking to Name2
- Name1 lookAt Name2** - Name1 is looking to Name2
- Name1 glanceAt Name2** - Watching Name2 shortly (moves the head only)

Title screen:

```
titleText [format["Direction of view: %1", getDir Name1], "plain down"]
```

5.23 - Unit is selecting weapon

By using the following syntax it's possible to make a unit select his second weapon. It's quite necessary to make sure that the weapon class name has been written the correct way and the unit does have the required weapon in its inventory.

```
Name selectWeapon "Stinger"
```

5.24 - Inflict damage or heal a unit

It's possible to allocate damage to a unit in the form of a value. It's also possible to allocate a predefined damage value from one unit to another. To make it work, values are needed which will give the defined strength of damage to the respective unit. **0** means no damage and **1** means absolute damage...dead. The decimal numbers between **0** and **1** define the intermediate values. Please make sure that the syntax setDamage can be written with one or two **m**. While getDamage will work only by using two **m**.

A damage value gets allocated to **Name1**:

```
Name1 setDamage 1 or Name1 setDamage 1  
Name1 setVehicleArmor 0.5
```

Name1 receives the damage value of **Name2**:

```
Name1 setDamage getDamage Name2
```

If the user wants to use the damage value as a condition, the following syntax is needed:

```
? damage Name1 >= 0.5 or ? getDamage Name1 >= 0.5
```

To destroy objects which are located around another object, these ones need to be defined in a Variable (**Area1**). Then all these Objects will be destroyed by using the following Syntax:

```
Area1 = nearestObjects [Object1,[],300] - Will list Objects around Object1  
{ x setDamage 1 } foreach Area1 - Destroy (i.e. Radio Alpha)
```

5.25 - Defining a death zone

One might have several reasons for setting up a death zone. If the user wants to create a scene where many dead bodies are lying all over the ground or a scene where none of the conflicted forces are to enter a specified area, a death zone is useful. Just enter following Syntax into the init box of the trigger, and make sure that the dimensions are defined to the trigger.

```
{_x setDamage 1} foreach thisList
```

It's only necessary to define the side which shall execute the trigger. If the trigger is to be executed only one time or several times, just define this by using **Repeatedly**.

5.26 - To check an area

It is possible to display the units which are located in a predefined trigger area. It's also possible to display the unit(s) of one or all forces by defining the respective trigger. To do this there are two triggers needed which have to be defined as follows.

The first one is the **checking trigger**. Set **Axis a/b** with the size of the area which has to be checked. Select the **respective forces** which are to be displayed by executing the trigger and rename the trigger to **Area1**.

The second trigger is the radio trigger which displays the radio device later in the game. Set it up with **Axis a/b** both set to **0**, and select repeatedly, then enter the following syntax in the **onActivation** field:

```
hint format ["%1",list Area1] or hint format ["%1",WEST countSide Area1]
```

If one is using the radio now, so all units which are located in **Area1** will be marked. One has now the further possibility to combine several areas with each other. I.e. if a second area has been created which is called **Area2**. So these both areas can be combined with each other and the results of both areas can be used as condition for further things.

```
hint format ["%1", (WEST countSide list Area1)+(WEST countside list Area2)]
```

5.27 - Bring about a certain behaviour of a unit in an area

It is possible to bring out a certain behaviour of a unit by using the following syntax. As already explained in **Chapter 5.26** it's possible to define this option for every side individually or even for all forces. To do this just define the area where the units are to receive the specified order by adding a trigger and renaming it the way you want to. Then, enter the syntax in the **onActivation** field of a trigger, waypoint or even a script:

```
{_x setBehaviour "Stealth"} forEach list Area1
```

All units would take cover now, for example. But the order **setBehaviour "Stealth"** is only one of many possibilities.

5.28 - Save or load a unit status

This command is great to use in campaigns when the player or even the other units have to carry over their last save status to the next mission.

If one wants to use this command in his campaign, one has to take care that the unit is still alive when the mission has ended. That's important if the status has to be used again later.

It's possible to save the status of a unit by using the order **saveStatus**, as this order name already explains itself. But this command only works while used in a Campaign, because the respective value will be saved in the **Objects.sav** of the campaign. So it won't work in multi- or singleplayer missions.

Savestatus

Status1 is variable and can be renamed as you want. The status of **Name1** will be saved now with the variable **Status1** by using following syntax:

```
xy=Name1 saveStatus "Status1"
```

That status now contains several pieces of information about the unit:

- The identity
- The health status
- The weapon status
- The ammunition status

If one wants to give that saved status to another unit, just use the example which is explained in the following section.

LoadStatus

```
xy=Name2 loadStatus "Status1"
```

The unit will now receive the saved status of **Name1**. That means that **Name2** looks like **Name1**, so it's also equipped with the same weapons, the same ammunition and has the same health status. Now one can say that **Name2** is a clone of **Name1**.

DeleteStatus

It's also possible to delete each status. Just use following Syntax:

```
deleteStatus "Status1"
```

5.29 - Degree of familiarity of a unit

This syntax can be used for several useful things. So it's possible, for example, to give a unit some information about another unit or using the degree of familiarity as condition for a further executing action. All values from **0** to **4** have to be used here again.

- 0** **Name1** has no knowledge of **Name2**
- 1** **Name1** has only some knowledge of **Name2**
- 2** **Name1** has enough knowledge of **Name2**
- 3** **Name1** has full knowledge of **Name2**

If one wants to give knowledge of unit to another, one has to use the following Syntax:

Name1 reveal Name2

Name1 now has knowledge of **Name2**

If one wants to use the knowledge as a condition to execute a trigger, one has to enter the following Syntax into the OnActivation field.

Name1 knowsAbout Name2 > 1

The trigger will execute and run its actions when **Name1** has **more** knowledge of **Name2** than value **1**.

The knowledge becomes less over time, so that the value will get back to **0**. The syntax above can be used the other way as well, one only needs to change the **>** to **<**.

Name1 knowsAbout Name2 < 0.8

That's quite useful when the unit always has to stay near to the other units. The trigger will execute if the unit is moving away from its group or gets killed, then the value gets set back, as defined above, to **0.8** and the trigger will execute.

This value can be further displayed as text message. To do this use following Syntax:

hint format ["Degree of familiarity: %1"; Name1 knowsAbout Player]

5.30 - Friendly enemy

If the following Syntax is used for a unit, the unit will no longer get shot or be recognized by the enemy. This gives the user the possibility to simulate a prisoner of war in the mission, who would not get shot immediately as he would if he was not a captive.

this setCaptive true or **Name setCaptive true**

If one wants to reset this, the order **true** just has to be changed again to **false**.

5.31 - Friendly Forces

In ArmA®, the user has the possibility to make the normally hostile forces fight together against each other, so it's possible to make west and east friendly to each other and let them fight against the rest. Furthermore, one can make the civilians become enemy, so that the military forces have to fight against them as well. This function gives the user a huge leeway because this option can be defined with a random value. No one will ever know what will happen when. The friendly side can become enemy the next day again. It adds a dynamic to the mission and promises lots of fun and helps keep the game from becoming boring. The following forces are freely definable:

West - East - Guerrila - Civilian - Enemy - Logic

It's also possible to make one side become friendly to the other one while the other one is still enemy. This means that the enemy side would open fire immediately if the other side becomes "known", but the other side wouldn't shoot back.

The value, which defines the SetFriend function, moves between **0** and **1**. All values above **0.6** means friendly and all values below **0.6** means enemy. Some Syntax examples:

WEST setFriend [EAST,1]

Now WEST would be friendly to EAST but not the other way. To do this, one needs a 2nd syntax like the following:

WEST setFriend [EAST,0.7]; EAST setFriend [WEST,0.7]

Now both sides are friendly to each other, but it may be that one or both of them will become enemy to the other again. One doesn't know if or when that'll happen, so it adds a lot of dynamics to the mission. If the values get set down, both sides will be enemy to each other again.

WEST setFriend [EAST,0]; EAST setFriend [WEST,0]

Random

Of course, everything is possible per the random command. That's one of the biggest advantages of this game, one can define everything using that command, that way, no one knows whether the other side is enemy or friendly against ones self. To do this just use following Syntax:

GUER setFriend [EAST,(random 0.9)]

Random was defined here as well which determines a coincidence-value for 0.9 by it self. This can get released with **0.9** (friendly) but also with **0.3** (enemy). No one will know what happens next.

Using in deathmatch

That function is further explained in **Chapter 7.6**. It's possible to make a single side enemy to itself

EAST setFriend [EAST,0]

5.32 - The Alert

By using the syntaxes which are explained in **Chapter 5.27**, it's possible to realize several types of alerts. It's up to the mission's history whether and how an alert will be caused. Here one can find some examples of the possibilities to use the alerts.

Example 1 - Causing an alert while detecting within a trigger area

To do this, just place a trigger with following settings onto the map.

Activation	BLUFOR Detected by EAST
Effects	Alarm

If a West is detected by an East within the defined trigger area, the trigger will be executed and the alert will be caused.

Example 2 - Causing an alert if a unit is detected

The alert shall be executed if a unit is detected.

Activation	Connect the trigger with the unit or the player character Detected by EAST
Effects	Alarm

Example 3 - Causing an alert if a unit or an object is no longer present (killed/destroyed)

Activation	Connect the trigger with the unit or the object Not present
Effects	Alarm

Example 4 - Alarm triggered if the group named Grp1 is smaller than a predefined value

Activation	None
Axis a/b	0
Condition	Count Units Grp1 < Value
Effects	Alarm

Additional information

It's possible to run nearly all objects by using a syntax in the onActivation line of the trigger. For example, to let all units which are located in the alert area named **Area1** receive knowledge about the unit or the player character, just use following Syntax:

```
{_x reveal Player} foreach list Area1
```

5.33 - Dead as condition

To check whether a unit is still alive, one needs a trigger with following settings. Of course there are several variants available as well. One can see an example of a method often used.

Activation	None
Axis a/b	0
Condition	!(alive Name) or not alive Name

The trigger is now checking globally by (by using **Axis 0/0**) whether **Name** is still alive and would execute its effects at **Activation** if **Name** gets killed.

5.34 - Distance of two units or objects

Sometimes it might be needed that the distance of two units or objects has to be used as condition to run a script or execute a trigger. To do this just use following syntax:

Player distance Jeep1 <= 50 or **Player distance Jeep1 == 50**

Local variable value allocated: `_distance = Player distance Jeep1`

Later in the Text:

`titleText [format["%1 Meters"; Player distance Jeep1], "plain down"]`

5.35 - Allocate a flag to a flagstaff

It's possible in ArmA® to define all flagstaves in the game with individual flags. The flags can be created by yourself or just use one of the flag packs which can be downloaded from the Fan-sites. To make the flag visible in the mission, just copy the image file into the missions folder and enter the respective link into the init field of the flagstaff in the game. Here, you can see an example of such a syntax:

`this setflagtexture "Name.paa"`

The file formats **PAC**, **PAA** and **JPG** are possible using pixel dimensions of **512 x 256**. The following flags can be used in the game by using this path:

`this setFlagTexture "\ca\misc\data\usa_vlajka.paa"`

USA
`usa_vlajka_co.paa`



Sovjet union
`rus_vlajka_co.pac`



Racs
`jih_vlajka.paa`



Russia
`rus_vlajka_co.paa`



SLA
`sever_vlajka.paa`



5.36 - Burning fire

If one wants to extinguish a fire or light it up later in the mission, the fire needs to be named and that name used in the following syntax:

this inFlame true or **Name inFlame false**

5.37 - Add or remove switchable units

By using the following syntax it's possible to allocate or to remove switchable units while a mission is running. A unit will be playable by using this Syntax:

addSwitchableUnit Name - To make the unit playable

removeSwitchableUnit Name - To make the unit unplayable

If one wants to switch to another player, just use this Syntax:

selectPlayer Name or **group Name selectLeader Name**

The ordinary command **Teamswitch** displays the Teamswitch menue.

5.38 - Read out and display player side, - name, -type

To read out the side of a unit in a multiplayer area, use following syntax:

? Side Player == West
Name Player == "Mr-Murray"

Later in the Text:

titleText [format["%1 -Soldier", side Player],"plain down"]

titleText [format["Hey %1", name Player],"plain down"]

titleText [format["Typ: %1", typeOf Player],"plain down"]

5.39 - Oppress player input

Sometimes it might be useful to oppress the input of the player. Certain situations, like sequences or map animations, shall only give some additional information to the player, so it's quite important to make sure that the players input is no longer disabled than strictly necessary. Otherwise the game may become boring to the player or the player starts to get frustrated and exits the game. To activate this function, just use the `disableUserInput` syntax. One only has to switch it to true, or false if one wants to delete it again.

disableUserInput true

5.40 - Force the map on the screen

Its possible to force the map on the screen while the game is running. There's no input of the default key (m) needed. This function is quite necessary while creating animations or sequences which shall give some further tactical information to the player. Its also possible to simulate many different things, like troops movements and more.

forceMap true

Now it would be possible to make markers moveable on the map for example. So it's necessary to oppress the player input while the animation is running.

5.41 - Adjusting distance of view

Sometimes it might be useful to set the distance of view higher or even lower. For example, one may want to create a sequence up to a mountain where a good view is needed. So it's possible in the game to change the distance of view, even if only for a short time. It's also possible to reduce the view of distance again later. This saves performance on the players PC. The view of distance is adjustable from **500** up to **10,000**. Just use following syntax:

setViewDistance 500

5.42 - Adjusting the weather

The weather is predefined in the editor, so one can select a kind of weather which will be active all the time the mission is running. But if one wants to make the mission more interesting because the weather is changing from time to time, one would need a special syntax to make it possible. That syntax makes the weather random, so one can't say which kind of weather will be next. The syntax contains some values again. These values are moving between **0** and **1**. This, interconnected with random-time and random-fog provides a more dynamic mission.

120 setOvercast 0.8

The first value (**120**) displays the time in seconds which is needed before the weather will change again. The second value (**0.8**) displays the kind of weather. **0** means that there are no clouds in the sky and the sun is shining. Every decimal number adds more clouds and even rain to the sky. Within a value of **0.5**, the whole sky is full of clouds and within the value of **1**, heavy thunderstorms are possible.

Random weather:

Its possible to decide this with a syntax which determines the weather-value per chance. If that syntax starts right when the mission begins, the weather will be different every time the mission begins. To do this just use following syntax:

0 setOvercast (Random 0.8)

Rain:

It's also possible to adjust the rain without the SetOvercast command.

0 setRain 0.8

10 setRain (Random 0.8)

Fog:

The syntax to use the same function for fog is similar to the one used for weather or rain. Only the syntax order is different:

10 setFog 0.6

60 setFog (Random 0.8)

5.43 - Adjusting date and time of day

The date and time of day are possible to adjust while the mission is running. One can change these by using fixed values or even with random values.

Year, month, day, hour, minute of day:

By using following syntax one can change the year, the day and the time of day as well. This syntax has to be defined in following sequence.

```
setDate [2006, 11, 30, 9, 0]
```

The 2006-11-30 at 09:00 (am) has been defined here. Remember that the game engine has been written with European standard and so you have to select/enter 21:00 if you'd like to use 09:00 pm. It's possible of course to generate some options with random values. You can see an example right here:

```
setDate [2006, 11, 30, (ceil random 8), (random 60)]
```

Time:

Its further possible to define the time of day individually:

SkipTime 1

This value skips the time by **1** hour

SkipTime -1

This value skips the time **1** hour backwards

5.44 - Slow motion or time sprint

By using the following syntax it's possible to slow the speed of the game down or even up. Especially while creating sequences, it's possible to make quite nice slow motion effects. It's also possible to enable a bullet mode as explained in **Chapter 6.10**. That isn't quite realistic but gives a nice experience to the user later in the game.

Slow motion:

All values which are set below **1** define the slow motion area. But the most effects can be reached by using the decimal values. The smaller the value the more slow the effect.

SetAccTime 1.0

- Normale Zeit

SetAccTime 0.0500

- Zeitlupe

Time sprint:

As the speed can get slowed down, It can be sped up as well. But that isn't quite useful. But if one wants to do it anyway, all values from **1** up to **8** can be used. This ones gives some more speed to the player character and the AI characters foot.

5.45 - Generating units and objects

It's possible to generate units, whole groups and objects while a mission is running. Furthermore, they can get deleted again when they are no longer needed. This offers lots of saved performance to the users PC because the objects will be generated right when they are needed. There are some possible variants available. Here you can get some examples.

Using with objects:

In the following example a D30 gun will be generated at Position XYZ:

```
Ari1="D30" createVehicle [x,y,z]
```

By using the setDir command, it's looking like this:

```
Ari1 setDir 190
```

It might so happen that a vehicle can't be used after it has been generated. To avoid this, the following order should be used as well:

```
Ari1 lock false
```

So it's possible now to use this vehicle. If one wants to generate a(n) **unit/object** at the position of another unit/marker/game logic, just use this syntax:

```
Bomb="SH_120_HE" createVehicle position Player  
Bomb="SH_120_HE" createVehicle getMarkerPos "Marker1"
```

By using the following syntax, a bomb will be created directly over unit **S1** at a height of **50** meters.

```
Bomb="SH_120_HE" createVehicle [(getPos S1 select 0),(getPos S1 select 1),50]
```

Used with units:

Unfortunately, one can not prevent that the syntax line will be very long to generate a unit. In this example, a sniper (WEST) named **Name1** will be generated right on the position of his leader (**S1/Player**). But make sure the leader has been placed on the map first. One can replace the unit (leader) with a game logic as well. The generated soldier has a skill value of **0.4** and his rank is **Corporal**.

```
"SoldierESniper" createUnit [position player, S1, "Name1=this",0.4,"Corporal"]
```

It's also possible to define it another way, so the position would get defined by a marker which has to be placed at the position of the leader. That markers properties have to be: **Axis (a/b) 0**, so that marker is not visible.

```
"SoldierESniper" createUnit [getMarkerPos "Marker1",EGrp1,0.8,"Corporal"]
```

This unit hasn't been allocated a name because a name is only needed if one wants to use this unit later in the mission. The leader, in this case, was named **EGrp1**.

Here is an example with adding a weapon:

```
"SoldierWSniper" createUnit  
[position Player, Player, "this addweapon ""binocular"" ", 0.8, "Corporal"]
```

Using with groups:

Furthermore it's possible to generate whole groups instead of single units. These units can be renamed individually if one wants to define special things to them later. The following script is used as an example only. In this script a group will be generated at the position of the marker "GrpOneM". That marker has already been placed on the map.

```
Grp1 = Creategroup EAST;
_Loader="SquadLeaderE" createUnit [getMarkerPos "GrpM", Grp1, "Grp1=this", 1, "Sergeant "];
_Unit2="SoldierEB" createUnit [getMarkerPos "GrpM", Grp1, "", 1, "Corporal"];
_Unit3="SoldierEB" createUnit [getMarkerPos "GrpM", Grp1, "", 1, "Corporal"];
_Unit4="SoldierEG" createUnit [getMarkerPos "GrpM", Grp1, "", 1, "Corporal"];
_Unit5="SoldierEMG" createUnit [getMarkerPos "GrpM", Grp1, "", 1, "Corporal"];
_Unit6="SoldierEAT" createUnit [getMarkerPos "GrpM", Grp1, "", 1, "Corporal"];
_Unit7="SoldierESniper" createUnit [getMarkerPos "GrpM", Grp1, "", 1, "Corporal"];
exit
```

As one can see, the first unit gets renamed as leader and he also is allocated a higher rank. This unit will be the leader of the group. The name of the whole group is "GrpOne" as well. To make sure that this group works correctly, read the following paragraph carefully.

Notice:

If soldiers of one side are to be generate while the mission is already running and no unit of the respective side has already been placed, it's important to allocate a center to this side to make sure that these units can communicate with each other. Then the setFriend order has to be used and the both sides needs to become enemy's to each other. Otherwise the AI wouldn't start shooting the enemy side. It's necessary to define the setFriend order and the center within the init.sqs script. If one has already placed units from all parties on the map, then these centers will be generated by the engine automatically. In the following example you can get the entries for the example **Init.sqs**:

Init.sqs

```
Createcenter EAST
Createcenter WEST

WEST setFriend [EAST,0]
EAST setFriend [WEST,0]
```

Center

As how Center can be created, it can be deleted again by using deleteCenter SIDE. But that would be unnecessary.

5.46 - Generate flares, smoke and explosions

To generate a flare or a smoke shell over an object or another XYZ Position, one can actually use the same order which is already explained in **Chapter 5.45**. The only different things are the class names, because these are not similar to the magazine names. You can get the necessary one here:

```
Flare1="F_40mm_Green" createVehicle [x,y,z]
```

If you'd like to generate something like this over or near to the XYZ Position:

```
Smoke1="Smokeshell"  
createVehicle [(getPos Name1 select 0),( getPos Name1 select 1), 10]
```

Make sure that these two options are written in one line (but this not really possible on this page of the guide). One can also use a short one:

```
Smoke1="Smokeshell" createVehicle position Name
```

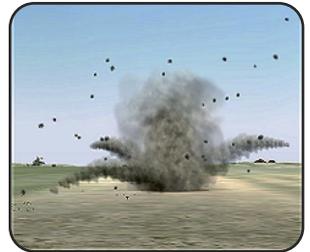
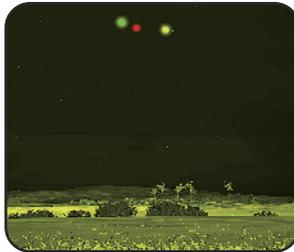
Notice!

Flares have to be generated within a level of 100 metres. Flares, and even smoke- gun grenades can be selected:

F_40mm_White	F_40mm_Red
F_40mm_Green	F_40mm_Yellow
Smokeshell	SmokeshellRed
SmokeshellGreen	B_40mm_HE

One can generate other effects as well. For example an **SH_125_HE** or something similar to generate explosions. So please take a look into **Chapter 3.10**. Several explosive devices are listed there.

Here you can take a look at some example previews.



5.47 - Delete units and objects

While only one syntax is needed to delete objects, there are several different syntaxes needed to delete units. If one wants to delete only the gunner of a Blackhawk - which hasn't been renamed - the syntax example will look like the ones below:

deleteVehicle Name - deletes the vehicle with that name

If one wants to delete the gunner of a helicopter, this unit needs to leave the chopper first. Before this happens, a name needs to be allocated to him. To do this just define the following syntax into the init line of the chopper:

Name=(Gunner Heli1)

Now, one can let him get out of the helicopter. It's also possible to write **Gunner Heli1**.

Name action ["Eject", Heli1] or just **doGetOut Name**

The name is important when it's time to delete the gunner. (**Gunner Heli1**) wouldn't work.

deleteVehicle Name

One can do all this to the **Commander**, the **Driver (Pilot)** or the whole **Crew**.

Delete units in a predefined area

To do this just create a trigger in an arbitrary size and rename it as you want (in this example: **Zone1**). Then select the side which has to be deleted or adjust it with the respective side or the option "Anybody". The syntax to delete:

{deleteVehicle _x} forEach list Zone1

5.48 - Adjusting radio menu

By using the syntax below it's possible to rename the radio menu while the mission is running. These menu options are numbered from **1** up to **10**.

1=Alpha, 2=Bravo, 3=Charlie, ...

Its possible to rename this by using following Syntax:

1 setRadioMsg "Alpha-Team"

The first entry of radio Alpha has been renamed to **Alpha Team**. If one wants to get an empty radio name, just set an empty space between the two quotes.

The radio option does still exist but it's no longer visible for the player and he'll think that its not available. Because instead of text, nothing more would be visible.

That option can be used as a condition if a whole team was killed.



5.49 - Allocate a call-sign to a group

It is possible to allocate different call-signs for several groups. So it is possible to distinguish each group from each other, if someone has spoken or even sent a side-chat. In Armed Assault® it's possible now to allocate names for each group individually. That option was not possible in Operation Flashpoint®. Furthermore it is possible now to define colors to each group. Here you can see some syntax examples.

Default:

```
Name setGroupid ["Alpha";"GroupColor0"]
```

Free Text:

```
Name setGroupid ["Assault-Team-Alpha";"GroupColor2"]
```



Assault-Team-Alpha 1: "WE'RE NOW IN POSITION..."

One can use following values:

Group names:

"Alpha"	"Golf"	"Convoy"
"Bravo"	"Hotel"	"Guardian"
"Charlie"	"Kilo"	"November"
"Delta"	"Yankee"	"Two"
"Echo"	"Zulu"	"Three"
"Foxtrot"	"Buffalo"	

Group colors:

0 – No Color	4 – Blue
1 – Black	5 – Yellow
2 – Red	6 – Orange
3 – Green	7 – Pink

Note:

It may be that the color will not be visible, but this issue has been known since Operation Flashpoint®. One can avoid this by writing both things in one line. The color needs to be defined manually and the last part of the syntax, - where the color has to be defined normally - free. In the following example, the team was named Assault-Team-Alpha and was assigned a red color.

```
Name setGroupid ["Assault-Team-Alpha - Red", ""]
```

5.50 - Send a radio message

There are several possibilities available to send a radio message. The message can be displayed globally, side, group or even only to the units in a single vehicle. In the table below are some examples:

Name sideChat "Test 1-2"	Name talks to his side
Name groupChat "Test 1-2"	Name talks to his group only
Name globalChat "Test 1-2"	Name talks to all parties
Name vehicleChat "Test 1-2"	Name talks to passengers inside a vehicle

If one wants to send a message from the headquarters, use following syntax:

```
[Side,"HQ"] sideChat "Move to your position and wait for further orders!"
```



The command **enableRadio false** is fading the radio text samples out and becomes them invisible.

5.51 - Creating sound

With the release of Armed Assault®, a new function has been enabled which wasn't available in Operation Flashpoint®. The user now has the possibility to define a sound right at the position he wants. In the following example, a sound named Littledog has been placed right on the position of the player.

```
Dog1=createSoundSource ["LittleDog",position Player,[],10]
```

The value **10** defines the range from the source of the sound to the player. By using this syntax, this script has been renamed and became moveable. So one can replace it now on any place on the map. The required syntax is:

```
Dog1 setPos getPos Name
```

So it's possible now to generate the sounds without placing anything on the map. The sounds can be found under empty/sounds or in a trigger or waypoint under the subsection "effects". There are some more sounds available:

Stream, Alarm, BadDog, BirdSinging, Chicken, Cock, Cow, Crow, Crickets1, Crickets2, Crickets3, Crickets4, Dog, Frog, Frogs, LittleDog, Music, Owl, Wolf

If one wants to create his **own sound**, it has to be defined in the description.ext first. This sound can be used by typing the respective name into the syntax. This name should be the one which has been defined in the description.ext before.

```
MySound=createSoundSource ["SoundName",position Player,[],10]
```

5.52 - Using own sounds

If one wants to use his own music in a mission, the files have to be defined in the `description.ext` first. The `description.ext` has already been explained in **Chapter 2.3**. One can find the explicit explanation of this theme here.

First you have to create the folder "sounds" in the missions folder. One can also create another folder named "music". Make sure that those names were written small. Then copy all the desired music files into the respective folder. The `description.ext` offers different sub areas, so that these files are controllable individually. This is quite necessary, because if a player is fading down the music, the radio sound wouldn't get faded down as well. The following example shows the way to define the `description.ext` to make it work as one wants.

Description.ext

```
// === Music =====>
class CfgMusic
{
    tracks[] = { Track1, Track2 };

    class Track1
    {
        name = "Track1";
        sound[] = {\music\track1.ogg, db+0, 1.0};
    };

    class Track2
    {
        name = "Track2";
        sound[] = {\musik\track2.ogg, db+0, 1.0};
    };
};

// === Sounds =====>
class CfgSounds
{
    sounds[] = { Artillerie };
    class Artillerie
    {
        name = " Artillerie ";
        sound[] = {\sounds\artillerie.ogg, db+0, 1.0};
    };
};
```

Continued on next page.

```
//=== Radio/Funk =====>
class CfgRadio
{
    sounds[] = {Funk1};
    class Funk1
    {
        name = "Funk1";
        sound[] = {"sound\funk1.ogg", db+5, 1.0};
        title = "Das ist ein Funkspruck!";
    };
};
//=== Environment =====>
class CfgSFX
{
    sounds[] = {};
};
class CfgEnvSounds
{
    sounds[] = {};
};
```

As one can see in the example above, two additional areas for radio and the surround-sound classes have been defined as well.

```
class CfgMusic
{
    tracks[] = {Track1,Track2};

    class Track1
    {
        name = "Track1";
        sound[] = {\music\track1.ogg, db+0, 1.0};

        titles[] = { };
    };
};
```

- Class CFG music

- Track list

- Class Track1

- Name Track1

- Source,
db = Loudness,
1.0 = Speed of the sound

- Hints to the sounds

By using the syntax **db**, it's possible to define the loudness of the sound. It's possible to adjust the sounds here which are too loud or too silent.

The speed option **1.0** defines the speed which is used while the sound will be played. So it's possible to make some speech samples higher or deeper by playing them slower or faster. But don't adjust it too high or your character will sound like M. Mouse.

If one wants to play a sound in the editor, the respective sound has to be selected out of the effects sub area of a waypoint or a trigger. To make it work one has to save the mission first. Then the mission needs to be loaded again. Now Arma® can read out of the new description.ext. You also have the possibility to run those sounds by using a syntax.

playMusic "Track1"

playMusic ["Track1", 30] (Plays Track1 from Second 30)

playSound "Artillerie"

Name say "Soundname"

As already explained, all these sounds can be faded down by changing the respective syntax.

10 fadeSound 0.5 10 fadeMusic 1 0 fadeRadio 0.1

The first value is needed for the time when this option has to appear. The second one is needed to define the loudness of the sound file. The command preLoadSound enables to load the sound into the RAM while the mission is loading

preLoadSound "Track1"

Connecting sounds with text

It's possible to connect a sound with a text which will be displayed right in the moment when the sound plays. There're two ways to make it work. The first one is to use the Stringtable and define the text as a string. The other way is to enter the text right into the description.ext. Now the text appears when the sound begins to play. This option is actually used for talk and even radio sounds. One can find an example in the script below:

```
// === Radio =====>
class CfgRadio
{
    sounds[] = {RadioMsg1, RadioMsg2};

    class RadioMsg1
    {
        name = "RadioMsg1";
        sound[] = {"\sound\radiosound1.ogg", db+10, 1.0};
        title = "It's done. I am ready for further orders.";
    };

    class RadioMsg2
    {
        name = "RadioMsg2";
        sound[] = {"\sound\radiosound2.ogg", db+10, 1.0};
        title = {$STR_RADIO_2};
    };
};
```

Play a random sound out of an Array

Even with the sounds it's possible to create a lot of things per random. A good method is to define all sounds within an Array to get them selected by the Game automatically and randomly later in the mission.

The following example will explain this by using a script. The used sounds don't need to be defined in the Description.ext because all of them are already defined in the Engine. The used script can get caused just by using this Syntax:

```
[ ] exec "scripts\music.sqs"
```

Each time when the script will get called, a coincidence sound will be selected and played out of the predefined Array.

```
_music = ["ATrack1","ATrack6", "ATrack10", "ATrack11", "ATrack12", "ATrack13",
          "ATrack14", "ATrack15", "ATrack19", "ATrack22"];
playMusic ( _music select floor(random((count _music) - 0.5)));
exit;
```

Call a random sound without a script

Another, quite well method of the "sounddynamic" will be shown with the now following Syntax example. The only disadvantage is that the user is not really able to select the Sounds by him self.

It's quite important, that all sounds does have the same name and are numbered at the end of the Title, i.e: **Track1**, **Track2** aso. Arma[®] own sounds will be used here again, which are defined as the same Principe like **ATrack1** till **Atrack27**. The Queens Gambits Titles are named from **ATrack1** till **ATrack9**.

The now following Syntax contains a **%1** instead of the actual value, which is used as a placeholder only. If this syntax gets caused so the engine will generate a random value out of a pool of numbers from **0** till **27** which will then replace the **%1**.

```
playMusic format ["ATrack%1",ceil random 27];
```

Note: The following entries don't need a definition in the description.ext – Entries:

WithCare_Smile - WithCare_Suicide - WithCare_War - WithCare_What

One can use different sound types of course which will used besides the default music tracks. If these sounds are not yet defined in the engine, so one need to define them into the **Description.ext**.

The following Syntax example explains all again by using the command **Say**.

```
Name1 say format ["Help %1",ceil random 3];
```

Because all generated random values are never be rounded, so they need to be rounded by using the ceil command which will round the number up or off. That's necessary to let the engine know which sound file has to be selected.

5.53 - Set Identity

It's possible to set an identity for every unit individually. But that wouldn't be necessary because it would be a huge effort. So it's more useful to set the identity for the main characters only.

The identity has to be defined in the description.ext before it can be used in the mission. To do this, just take a look at the example below.

```
class CfgIdentities
{
    class MrMurray
    {
        name = "MrMurray";
        face = "Face33";
        glasses = "none";
        speaker = "Dan";
        pitch = 1.00;
    };

    class Memphisbelle
    {
        name = "Memphisbelle";
        face = "Face10";
        glasses = "none";
        speaker = "Howard";
        pitch = 1.00;
    };

    class Dan
    {
        name = "Dan";
        face = "Face22";
        glasses = "none";
        speaker = "Russell";
        pitch = 1.00;
    };
};
```

As one can see, the names are to be freely defined. Only the faces and the voices are predefined. Please make sure that the clasps were set on the correct position in the Description.ext!

If one wants to set the predefined identity to the respective unit, the predefined name has to be written into the init line of the unit.

Name setIdentity "**YourName**" or **this** setIdentity "**Mr-Murray**"

If one wants to set the predefined identity to the respective unit, the predefined name has to be written into the init line of the unit.

Name1 setIdentity "YourName" or **this setIdentity "Mr-Murray"**

This Identity can be saved and loaded again while used in a Campaign. The needed value will be saved directly in the **Objects.sav** of the Campaign and can be deleted again by using **deletelIdentity "XYZ"**.

Name1 saveIdentity "Name1Save" or **Name1 loadIdentity "Name1Save"**

To allocate a face to a unit just use this Syntax:

Name1 setFace "Face33" or **this setFace "Face33"**

Armed-Assault contains **61** different faces which can be selected. These faces are numbered from **Face1** up to **Face57** and **FaceR01** up to **FaceR04**. If the user wants to use his own face file, this file needs to be saved in the Missions- respective User folder. To make it work use following Syntax:

Name1 setFace "MyPicture.jpg"

Possible pixel sizes: **1024*1024; 512*512; 256*256**.

Maximal file size **100 Kb!**



Glasses:

One can allocate glasses to a unit. Take a look at the examples which are listed below:

glasses = "Sunglasses";	Sunglasses
glasses = "Spectacles";	Normal glasses
glasses = "None";	No glasses

Speaker:

The following voices are selectable in ArMA[®] as of today:

Amy	Dan	Howard	Robert	Ryan
Brian	Dusan	Mathew	Russell	

Pitch:

By adjusting the pitch value (pitch = 1.00) the voice pitch can be adjusted higher or deeper.

5.54 - Mimics

It's possible to allocate mimics to the units. This possibility enables the units to become emotional. They can look friendly or even bad. To do this just use following syntax:

Name setMimic "Smile"

Normal - Surprised - Agresive - Hurt - Ironic - Smile - Cynic - Angry - Sad - Default

The following Syntax is changing the mimics by adjusting the values from **0** to **1**.

Name setFaceAnimation 0.5

5.55 - The Action Command

The action commands are used to allocate the various actions to the units. So there are several possibilities available for how to define them. Read the explanations here:

- Object:** The unit (name) which has to execute the action. If one wants to use a vehicle then the Commander will be selected automatically.
- Type:** Name of the respective action (see action orders overview)
- Target:** This option is similar to Object and actually means the name of the unit which has to execute the action.

Syntax:

The necessary syntaxes are listed here:

Name action [<type>]

Name action [<type>, <target>]

Name action [<type>, <target>, additional parameters]

To use additional parameters like weapons and / or magazines, these need to be defined as shown in the lists. **Fire** is next to **Action**, another possibility but this only works for a few orders:

Name fire

["PipebombMuzzle", "PipebombMuzzle", Pipebomb]

- Unit activates a satchel charge

["M203Muzzle", "M203Muzzle", "1Rnd_HE_M203"]

- Unit fires a grenade

["M203Muzzle", "M203Muzzle", "FlareGreen_M203"]

- Unit is firing a flare

["HandGrenadeMuzzle", "HandGrenadeMuzzle", "HandGrenade"]

- Unit is throwing a hand grenade

["SmokeShellRedMuzzle", "SmokeShellRedMuzzle", "SmokeShellRed"]

- Unit is throwing a smoke grenade

["BombLauncher", "BombLauncher", "6Rnd_GBU12_AV8B"]

- Harrier is dropping a bomb

Name action

["TouchOff", Name] - Executes Satchel Charges

["Eject", Heli1] - Gets out

["Hidebody", Name2] - Unit is hiding a corpse

["CancelAction", Name] - Aborting action

Overview of the most often used action commands

The following is a command overview of the most important action commands. Some of them don't work from the beginning and / or haven't been activated yet.

["None", <target>]
["GetInCommander", <target>]
["GetInDriver", <target>]
["GetInGunner", <target>]
["GetInCargo", <target>]
["Heal", <target>]
["Repair", <target>]
["Refuel", <target>]
["Rearm", <target>]
["GetOut", <target>]
["LightOn", <target>]
["LightOff", <target>]
["EngineOn", <target>]
["EngineOff", <target>]
["SwitchWeapon", <target>, <weapon index>]
["UseWeapon", <target>, <weapon index>]
["TakeWeapon", <target>, <weapon name>]
["TakeMagazine", <target>, <magazine type name>]
["TakeFlag", <target>]
["ReturnFlag", <target>]
["TurnIn", <target>]
["TurnOut", <target>]
["WeaponInHand", <target>, <weapon name>]
["WeaponOnBack", <target>, <weapon name>]
["SitDown", <target>]
["Land", <target>]
["CancelLand", <target>]
["Eject", <target>]
["MoveToDriver", <target>]
["MoveToGunner", <target>]
["MoveToCommander", <target>]
["MoveToCargo", <target>]
["HideBody", <target>]
["TouchOff", <target>]
["SetTimer", <target>]

["Deactivate", <target>]
["NVGoggles", <target>]
["ManualFire", <target>]
["AutoHover", <target>]
["StrokeFist", <target>]
["StrokeGun", <target>]
["LadderUp", <target>, <ladder index>, <ladder position>]
["LadderDown", <target>, <ladder index>, <ladder position>]
["LadderOnDown", <target>, <ladder index>, <ladder position>]
["LadderOnUp", <target>, <ladder index>, <ladder position>]
["LadderOff", <target>, <ladder index>]
["FireInflame", <target>]
["FirePutDown", <target>]
["LandGear", <target>]
["FlapsDown", <target>]
["FlapsUp", <target>]
["Salute", <target>]
["ScudLaunch", <target>]
["ScudStart", <target>]
["ScudCancel", <target>]
["User", <target>, <action index>]
["DropWeapon", <target>, <weapon name>]
["DropMagazine", <target>, <magazine type name>]
["UserType", <target>, <action index>]
["HandGunOn", <target>, <weapon name>]
["HandGunOff", <target>, <weapon name>]
["TakeMine", <target>]
["DeactivateMine", <target>]
["UseMagazine", <target>, <magazine creator>, <magazine id>]
["IngameMenu", <target>]
["CancelTakeFlag", <target>]
["CancelAction", <target>]
["MarkEntity", <target>]
["Talk", <target>]
["Diary", <target>]
["LoadMagazine", <target>, <magazine creator>, <magazine id>, <weapon name>, <muzzle name>]

5.56 - The animation command

It's possible to allocate animations to a unit by using the animation commands. One can divide all commands in two main sections. The first section is the command called **switchMove** which will switch the unit into the respective animation. the second one is the **playMove** command, which has to be used if one wants to run an animation. Some commands don't really cooperate with the PlayMove command. If it happens, one has to use the SwitchMove command. Every animation needs its time to get started. To avoid trouble, especially when a second animation follows after the first one, it's recommended to use the delay command (~10). The delay is a small code which pauses the game engine for the defined length of time before going on with the next command.

Animation commands are a nice feature. For example a base where some units are doing some sports, talking to each other or another soldiers salute to one who is passing the entrance to the base. All these animations are to be realized by using the animation commands. The way how to write such a script is explained below:

Name playMove "Animationsbefehl"

Name switchMove "Animationsbefehl"

Using with groups:

Next to the default commands there're different possibilities to select a unit which has to cause an Animation per coincidence or what kind of Animation has to be caused.

To define a unit us following Syntax:

(Units group Group1 select 2) playMove "Animation"

Unit 3 of **Group1** would cause the Animation now. But why has **unit 3** been selected although **select 2** was defined? That is explained as follows.

select 0	-	Leader
select 1	-	Unit 2
select 2	-	Unit 3

Randomly Group Variant

All of this is possible of course per coincidence. That would look as follows:

((Units group Group1) select ceil random 5) playMove "AnimationName"

In this case a random value between **0** and **5** has been created by using the command **ceil random 5** which will cause a Animation for the per coincidence selected unit of **Group1**. It's quite important to make sure that the unit will have a size of **6** units minimum.

Randomanimation

Here are several variants possible, but we will use only 2 of them. At first there's the variant that the Animations are numbered like the Panic-Animation which is numbered from ActsPercMstpSnonWnonDnon_Panicking1 till 7. It's important to make sure that the names are always the same. The numbers are the only thing which are different to each other.

Name playMove format ["Animation%1",ceil random 7];

Getting random animations out of an array

The other Variant is that the engine is selecting an Animation automatically out of an Array. The advantage is that one can predefine some different Animations in this Array. That happens usually by using a script, which looks like this:

[Name] exec "scripts\animation.sqs"

```
_Unit = _this select 0
_Anim = ["Anim1", "Anim2", "Anim3", "Anim4", "Anim5", "Anim6"];
_Unit playMove ( _Anim select floor(random((count _Anim) -0.5 )));
exit;
```

The animation status

The animation status shows the user whether the Animation is still running or already been ended. One can use this information of course for further things like conditions.

? animationState _Unit == "Anim1" : hint "Anim runs"

Ignoring end of Animation

Who doesn't know that, that a unit is disabling the current animated position and is changing back to the default position. That happens right then when the current Animation has been ended. But if one likes that a unit has to keep being in the animated position so the **DisableAI-command** is needed.

To do this one needs to give the syntax for the respective Animation at first and right behind the **DisableAI-command** which will look like this:

Name playMove "Animation"

Name disableAI "Anim"

If its needed now that the unit has to change back to the default position and has to adopt an attitude so one has to use the following Syntax:

Name enableAI "Anim"

A further possibility to get a Animation loop is the using of a Eventhandler.

Name switchMove "Animation";

Name addEventHandler ["AnimDone", {Name switchMove "Animation"}];

A list with the most used commands will be shown on the list below:

Animation	Description
AmovPsitMstpSlowWrflDnon	Sitting down
ActsPercMstpSnonWnonDnon_MarianQ_TVstudioMan_Loop1	Sitting
ActsPercMstpSnonWnonDnon_MarianQ_TVstudioMan_Loop2	Sitting and talking
ActsPercMstpSnonWnonDnon_MarianQ_TVstudioMan_Loop3	Sitting and talking
ActsPercMstpSnonWnonDnon_MarianQ_TVstudioMan_Loop4	Sitting and talking
ActsPercMstpSnonWnonDnon_MarianQ_TVstudioMan_LoopLong	Sitting and talking
ActsPercMstpSnonWnonDnon_MarianQ_WarReporter	Standing around
AwopPpneMstpSgthWnonDnon_throw	Lying and throws a grenade
AwopPercMstpSgthWrflDnon_Throw1	Throws a grenade
AswmPercMrunSnonWnonDf_AswmPercMstpSnonWnonDnon	Unit swims
DeadState	Unit dies
SprintBaseDf (SprintBaseDfL=rennt links / SprintBaseDfR rennt rechts)	Running with weapon
SprintCivilBaseDf (SprintCivilBaseDfL=links / SprintCivilBaseDfR=rechts)	Running without weapon
AmovPercMstpSnonWnonDnon_AwopPercMstpSoptWbinDnon	Taking Binoculars
AmovPercMstpSnonWnonDnon_Ease	If attitude assumes
AmovPercMstpSnonWnonDnon_AmovPknIMstpSnonWnonDnon	Kneels down on one knee
AmovPercMstpSsurWnonDnon	Hands behind the head
AmovPercMstpSnonWnonDnon_AinvPknIMstpSnonWnonDnon	Unit puts a pipebomb
AmovPercMstpSlowWrflDnon_AmovPsitMstpSlowWrflDnon	Sitting down
AinvPknIMstpSnonWnonDnon_AmovPknIMstpSrasWpstDnon	Kneels down and gasps
AinvPknIMstpSlayWrflDnon_AmovPknIMstpSrasWrflDnon	Kneels down and gasps
AinvPknIMstpSlayWrflDnon_healed	Heal Animation
AinvPknIMstpSlayWrflDnon_healed2	Heal Animation
AinvPknIMstpSnonWnonDnon_healed_1	Heal Animation
AinvPknIMstpSnonWnonDnon_healed_2	Heal Animation
AinvPknIMstpSlayWrflDnon_medic	Bandaging a victim
AinvPknIMstpSnonWnonDnon_medic_1	Bandaging a victim
AinvPknIMstpSnonWnonDnon_medic_2	Bandaging a victim
AidIPercMstpSnonWnonDnon08	Shouldered the weapon
AinvPknIMstpSnonWnonDnon_1	Kneeling down (Ammobox)
AinvPknIMstpSnonWnonDnon_2	Kneeling down (Ammobox)
AinvPknIMstpSnonWnonDnon_3	Kneeling down (Ammobox)
AinvPknIMstpSnonWnonDnon_4	Kneeling down (Ammobox)
AmovPercMrunSlowWrflDf_AmovPpneMstpSrasWrflDnon	Takes cover
AmovPercMstpSnonWnonDnon_carCheckPush	Vehicle check
AmovPercMstpSnonWnonDnon_carCheckWheel	Vehicle check
AmovPercMstpSnonWnonDnon_carCheckWash	Washing car
AmovPercMstpSnonWnonDnon_exerciseKata	Martial Arts
AmovPercMstpSnonWnonDnon_exercisekneeBendA	Knee bend slow
AmovPercMstpSnonWnonDnon_exercisekneeBendB	Knee bend fast
AmovPercMstpSnonWnonDnon_exercisePushup	Pushup
AmovPercMwlkSlowWrflDf_ActsPercMstpSlowWrflDnon_HitLeg	Hit leg
AmovPercMstpSlowWrflDnon_ActsPercMstpSlowWrflDr_HideFromFire	Hide from fire
ActsPpneMstpSnonWnonDnon_AmovPercMstpSnonWnonDnon_Injured3	Victim lays on the ground
AsigPercMstpSlowWrflDnon_AmovPercMrunSlowWrflDnon_GoGo	Forward guys!

Animation	Description
AmovPercMstpSlowWrflDnon_Salute	Soldier is saluting
AmovPercMstpSnnonWnonDnon_seeWatch	Watching the clock
AmovPercMstpSnnonWnonDnon_talking	Talking
AmovPercMstpSlowWrflDnon_talking	Talking
ActsPercMstpSnnonWnonDnon_MarianQ_shot1man	Talking
ActsPercMstpSnnonWnonDnon_MarianQ_shot3man	Talking
ActsPercMstpSnnonWnonDnon_MarianQ_shot4man	Patrolling
ActsPercMstpSnnonWnonDnon_MarianQ_shot5man	Patrolling2
ActsPercMstpSnnonWnonDnon_MarianQ_TVstudioMan_Loop1	Sitting
AmovPsitMstpSlowWrflDnon_Smoking	Sitting and smoking
AmovPsitMstpSlowWrflDnon_WeaponCheck1 (Number 1 to 2)	Sitting, checking the weapon
AmovPsitMstpSnnonWnonDnon_ground	Sitting/Hands back
AmovPsitMstpSlowWrflDnon	Sitting / Weapon on ist fold
ActsPercMstpSnnonWpstDnon_InterrogationSoldier	Questioning
ActsPercMstpSnnonWnonDnon_InterrogationVictim	Victim of questioning
ActsPercMstpSnnonWrflDnon_ArrestingSoldier	Arresting a person
ActsPercMstpSnnonWnonDnon_ArrestingMan	Victim of arresting
ActsPercMstpSnnonWnonDnon_ArrestingManLoop	Lying fettered
AmovPlieMstpSnnonWnonDnon	Got hit and lying on the ground
ActsPknIMstpSnnonWnonDnon_TreatingInjured	Convulsed with pain
ActsPpneMstpSnnonWnonDnon_Injured1 (Number 1 to 2)	Convulsed with pain
ActsPknIMstpSnnonWrflDnon_TreatingSoldier	Healing injured within battle
AsigPercMstpSlowWrflDnon_SendMenInAction	Cowering and giving orders
ActsPercMstpSlowWrflDnon_listeningOrdersUnderFire	Cowering and listening to orders
ActsPknIMstpSnnonWnonDnon_ThingPassingStill	Kneeing on the ground and talking
ActsPercMrunSlowWrflDf_FlipFlopPara	Running and rolling forward
AsigPercMstpSlowWrflDnon_GoGo	Go go go
ActsPercMstpSnnonWnonDnon_ThingPassingStill	IDcard controll
ActsPercMstpSlowWrflDnon_ThingPassingMoving	IDcard controll
ActsPercMwikSlowWrflDnon_PatrollingBase1 (Number 1 to 4)	Feeling boring
ActsPercMstpSlowWrflDnon_Lolling	Yawning and stretching
ActsPercMstpSnnonWnonDnon_DancingDuolvan	Happy dancing
ActsPercMstpSnnonWnonDnon_DancingDuoStefan	Happy dancing
ActsPercMstpSnnonWnonDnon_DancingStefan	Happy dancing
ActsPpneMstpSnnonWnonDnon_Panicking	Panic animation
ActsPercMstpSnnonWnonDnon_Panicking1 (Number 1 to 7)	Panic animation
ActsPercMrunSlowWrflDf_TumbleOver	Stumbling but going on
ActsPknIMstpSlowWrflDnon_ThingPassingMoving	Running covered and securing
ActsPercMstpSlowWrflDnon_Listening	Listening to responsiveness
ActsPercMstpSnnonWnonDnon_Listening	Listening to responsiveness
ActsPercMstpSnnonWnonDnon_Talking1 (Number 1 to 2)	Standing without weapon and talking
ActsPercMstpSlowWrflDnon_Talking1 (Number 1 to 2)	Standing with weapon and talking
m2s1kancler	Running / giving order
m2s2kancler	Standing and listening
m2s1pobocnik	Standing and talking
m2s2Zoldak1	Standing and aiming

Animation	Description
m2s2Zoldak2	Standing / aiming from hip
m2s2Orlando_centered	Lifting hands / protecting themselves
m2s2Wicks_centered	Standing with gun in hips
m2s2Lamb_centered	Holding weapon with gun barrel up
m2s2kancler_centered	Running around scared
m2s2bodyguard_centered	Standing with hands up and falling
XOutroZoldak1	Standing inflected forward
XOutroZoldak2	Leaning to someone
XOutroLamb	Weapon in hip and pitching
XOutroOrlando	Standing with folded arms
m1s1Wicks	Holding weapon on board carrier
m1s2Wicks	Holding weapon near be hips and giving orders
m2s1wicks	Holding weapon up on hips
m2s2Wicks	Holding weapon right in front of his chest
m2s3Wicks	Got hit lying on the ground
x01Wicks	Holding weapon at board carrier and giving orders
m5s1orlando	Walking around, weapon shouldered and gesticulating
m2s2Orlando	Throwing hands up and capitulate
m2s3Orlando	Standing with shouldered weapon
m6a51Orlando	Standing with folded arms
m6a52Orlando	Standing with folded arms
x03Orlando	Running around
x04Orlando	Running around and gesticulate
m5s1Gonzales	Running around and gesticulate
m6a51Gonzales	Standing and listening
m6a52Gonzales	Running around and gesticulate
M6bs2Gonzales	Running around and gesticulate
M6bs3Gonzales	Standing, gesticultaing and folding arms
x04Gonzales	Watching the ground and running around
x05s1Gonzales	Standing with hand at the chin
x05s2Gonzales	Standing with hand at the chin and gesticulating
x05s4Gonzales	Running forward and is telling
x05s5bGonzales	Folding Hands and explaining
m1s1Pedros	Crying for help and pointing somethings with hands
x05s1Lamb	Standing with lifted weapon
x05s2Lamb	Standing with lifted weapon
x05s3Lamb	Standing with lifted weapon and gesticulating
x05s4Lamb	Standing with lifted weapon and running
x05s5aLamb	Standing with lifted weapon and gesticulating
x05s5bLamb	Standing with lifted weapon and gesticulating
M6bs2Lamb	Running with weapon under the arm and gesticulating
M6bs3Lamb	Standing with weapon under the arm
x01Lamb	Running with lifted weapon
x03Lamb	Running with weapon in the front of his body
m1s2lamb	Lifting weapon and putting hand on ist back
m5s1Lamb	Lifting weapon with hands on ist magazine

5.57 - Disable AI units

By using the following syntax it's possible to disable the AI units. That means that those units will not fire and not move. The following possibilities are available:

Name disableAI "Move"	- Unit stops moving
Name disableAI "Target"	- Unit is no longer observing enemy units
Name disableAI "Autotarget"	- Unit doesn't watch anything
Name disableAI "Anim"	- AI is no longer able to change any animation
Name disableAI "Watch"	- Unit is no longer looking around

By using enableAI all the effect will get deleted and the unit is behaving normal again.

5.58 - SetVelocity

This order is quite useful to make objects or units move over the map. If one wants to generate an aircraft which has to be in the air when it has been generated, this order will make it move in a pre-defined direction so that the pilot has time to speed up his aircraft. To do this just use following syntax:

Name setVelocity [0,100,100]

5.59 - The Information Text

By using the following syntax it's possible to get some information displayed on the screen. There are several possibilities to do this:

hint "Text"	- Text appears after call
hintC "Text"	- Text appears after call and the game will be paused
hintCadet "Text"	- Appears in Cadet mode only

5.60 - Units keeps lying or keeps standing

If a unit is changing his position and stance when he should be holding still, the following commands can be used to hold the unit in place in a specific stance. The orders kneel and kneelDown are intended on the part of BI but they don't really work up to version 1.14. But those commands shall work in further versions so you can get the needed syntaxes below:

Name setUnitPos "Up"	- Unit keeps standing
Name setUnitPos "Middle"	- Unit is kneeling
Name setUnitPos "Kneel"	- Unit is kneeling
Name setUnitPos "KneelDown"	- unit is kneeling and is changing between lying and kneeling by itself
Name setUnitPos "Down"	- Unit is lying
Name setUnitPos "Auto"	- Unit decides for itself

5.61 – Using ID's

Every Object which is located on the map, owns an own ID which is individual to each single Object, indifferently whether it's a tree, bridge, house or a part of a street. One can get the ID's displayed by clicking the respective button in the Editor which is called **Show ID's**. Now its possible to check the status of each Object, may be to use it for a condition or lots of further reasons which are dedicated to the mission targets.

In Operation Flashpoint® the following command was used:

(Object 12345) setDamage 1

But this command is unfortunately no more running in ArmA®. Now the both commands **NearestObject** and **NearestBuilding** are to be used.

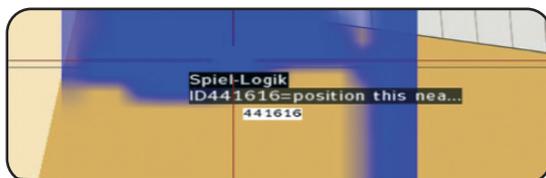
To call the respective object one need to place a game logic or a trigger right onto the object and notes its ID. The following Syntax is calling the respective object. It'll defined into the initline of the logic:

House=position this nearestObject 441616

or **House=nearestObject this** resp. **House=nearestBuilding this**

The word **House** is just a variable and can be free defined. **Object1** would also a possible way to name the object. The rest of the Syntax is the actual important one, because the ID at the end of the line, will call the object. The logic needs to be placed directly onto the object, its also possible to use some an object then a game logic. One should take care that the logic or object will be placed right onto the ID or close to it and that the Syntax will defined accurate. If one is using this command with a direct ID call, so the logic can be set free on the map.

A example how it would look like in the Editor:



Once the Object or ID got a name allocated (as shown above: **House**) so it can be used in different ways. There're some examples shown on the next page about how to use this possibility now.

Allocate a damage to an Object:

Once a object received a name so one can either dammage or heal it again. To do this the already known SetDamage Syntax is needed:

House setDamage 1

Getting damage status of an object:

By using the following Syntax, one has the possibility to receive the damage status of an object.

getDamage House

This information can be used again for further things like conditions or any other things, to check whether the building or the respective object received a damage value which is even higher or less than a predefined one. used in a checking trigger would look like this:

Checking trigger:

Axis a/b: 0
Condition: getdamage House > 0.7
on Activation: hint "The building is heavy damaged!"

Destruction as condition or Mission target:

As explained on the prior side, the user has now the possibility to use the destruction of an object as condition or a mission target. If a condition has to be defined, so it needs to be written in the **onActivation** line of a trigger or waypoint.

Condition : !(alive House) or not alive House

If the destroyed building has to be defined as mission target so a checking trigger is needed which only needs to be placed on the map defined as **End 1**.

Checking trigger:

Type: End 1
Once
Axis a/b: 0
Text: Mission Target
Condition : !(alive House)

Its recommended to use the countdown again, to make sure, that End 1 will not get caused right when the building has been destroyed but a little later. **Chapter 4.6** will explain the way how to end a mission, much more better.



Indestructible Object:

If one wants that some objects are not to be destructible so one can use the following possibility. At first the object needs to be named as explained in **Chapter 5.61** and furthermore a checking trigger has to be placed on the map which has following settings:

Checking trigger:

Type: Repeatedly
Axis a/b: 0
Condition: getDammage House > 0.1
on Activation: House setdamage 0

If the object is receiving a damage which is higher then the value **0.1**, so the damage will reset back to **0**. Because the trigger was set to repeatedly so the building will never be destroyed.

Opening/closing doors of an object:

Some objects have animated doors which can either be opened or even closed by using the action menu. Now its possible to control these doors, if the respective object has been named before as explained in **Chapter 5.61**. The needed Syntax sounds:

House animate ["dvere1",1]

Dvere1 stands for door 1. Each further door does have to be named **Dvere2**, **Dvere3** aso. They will be closed with **0** and get opened again with **1**. Further possibilities are:

Bare animate ["Bargate",1] - To control the barrier

Target animate ["Terc",1] - To control the target

Deactivate lanterns resp. lights:

One can deactivate lanterns or even lights by using the ID's. But before the object needs to get named at first by using the following Syntax:

Light1=nearestObject this

One only needs to place a game logic next to the lantern and enter the Syntax into the initline. Then it can be called from any place on the map (waypoint, trigger etc.). To do this use following command:

Light1 switchLight "Off"

If both command lines are entered into the initline of the logic so the lights will be deactivated so far the mission begins.

Light1=nearestObject this; Light1 switchLight "Off"

5.62 – Placing units inside a building

Many buildings in Armed Assault® are passable as they was in Operation Flashpoint®. But one of the new Arma® features is that the leader of a group can allocate his Soldiers special positions in a building. To do this only a single mouse click is needed, that will take the effect that the resp. soldiers will move to their desired postions.

These positions are fixed defined positions inside a building modell. Each positions owns an own number. The user can now place the units directly on the desired positions inside or even on the top of the building. A perfect example is the Hotel which has around 265 fixed positions, so the Hotel own nearly the most positions of all used buildings in the game. Because of the fact that its quite unclear where all these positions are located so all five levels with their positions in the Hotel, are shown on the next pages. A special feature can be found in **Chapter 6.16**. This script will become units patrolling through the building.

To place units inside a building so one only needs to place a unit directly onto the building and defines the following Syntax into the initline of the unit

this setPos (nearestBuilding this buildingPos PositionNumber)

This command is much less complicate then name **setPos [X,Y,Z]**, because all the needed coordinates doesn't have to get determined before. If the building has been named as explained in **Chapter 5.61**, so following Syntax can be used as well:

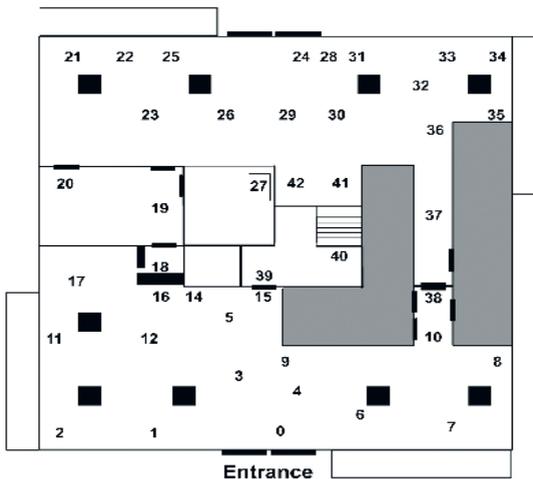
this setPos (HouseName buildingPos PositionNumber)

If one wants to place units randomly inside a building so the random command has to be used.

this setPos (nearestBuilding this buildingPos Random 265)

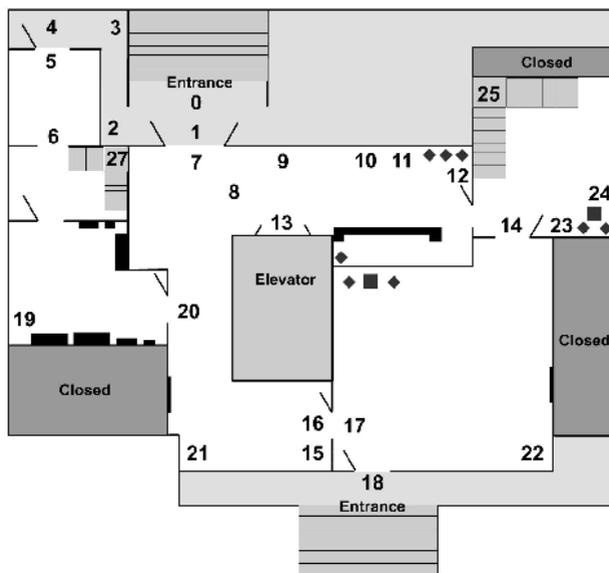
By using this command a random position will be defined now which is located somewhere between **0** and **265**.

Ground Floor (Positios 0 till 42)



Radio station

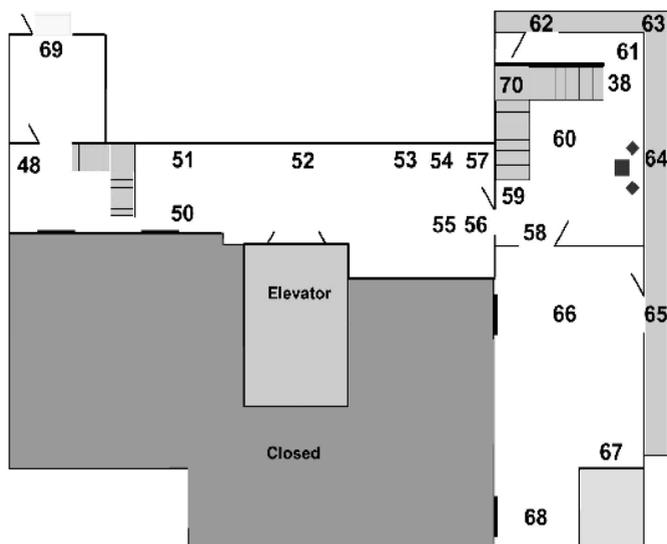
Ground Floor (Positions 0 till 27)



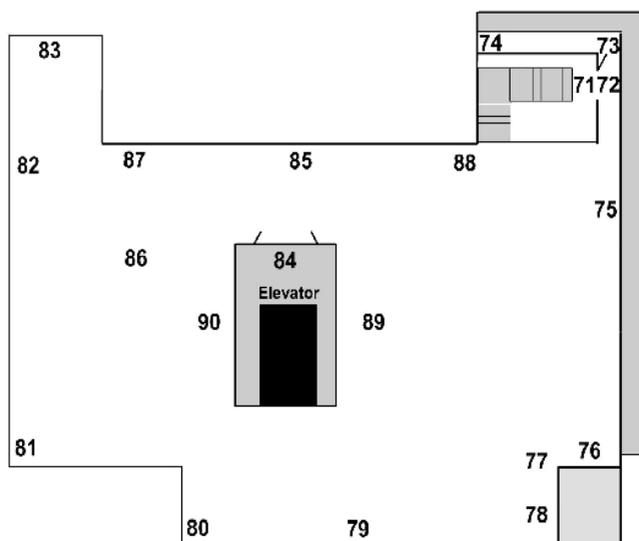
1. Floor (Positions 28 till 47)



2. Floor (Positions 48 till 70)



Roof (Positions 71 till 90)



5.63 – Unit is movin to desired house position

As units can be placed onto a special position, so one can ask them to move to an x – desired position within the respective building. To do this just use following Syntax

Name doMove (nearestBuilding this buildingPos 123)
Name doMove (House1 buildingPos 123)

Or to another random position:

Name doMove (House1 buildingPos random 123)

The way how to name a object or a building has been explained well accurate in **Chapter 5.61**. In example:

House1=nearestBuilding this

5.64 – Getting position displayed

It's possible to get the position of a unit either displayed as text or as condition for somethings else. The already known XYZ- values are to be used as normal. The only different is that the both commands are causing different results:

getPos	-	displayes the height over ground resp. over an object
getPosASL	-	displays the height over the sea level exactly
X-Position	-	getPos Name select 0
Y-Position	-	getPos Name select 1
Z-Position	-	getPosASL Name select 2

Getting the position displayed as text:

To do this the **hint format** and the **titleText** syntaxes are to be used, which can be defined as needed. It's possible to get either one or all three values displayed concurrently. The following command will display the position in XYZ order incl. the exact height over the sealevel. The GetPosASL command is not interesting for the X and Y values..

hint format ["%1", getPosASL Name] or hint format ["%1", position Name]

If one wants one single value only, so the Syntax need to be defined with the respective Select-value. In example the Determination of the height above the sea.

hint format ["%1", getPosASL Name select 2]

hint format ["Your current hight is %1", getPosASL Name select 2]

titleText [format["%1 Meter", getPosASL Name select 2],"plain down"]

One shouldn't embroider to getting positions displayed.



Position as condition:

Now it's possible to use the position of something as a condition for something. I.E. the trigger has to execute so far the helicopter called **Heli1** is flying higher than **100** meters or the player was warned because his Y value became much higher than it actually had to be. That would happen if the player would run/drive/fly to far either in northern or southern direction. By using this way it's possible to border the battlefield very easy. To do the same for the altitude just use the following syntax::

? getPosASL Heli select 2 > 100

Or in example to check whether the unit has already exceeded the Y - value.

? getPosASL Name select 1 > 10600



5.65 - The Eventhandler

One can see the eventhandler as a kind of condition, which can be allocated to one or more units or objects. If the condition has been caused so the predefined action will be executed. So one is allocating an event to a unit which is defined as condition for some things. It's pretty easy to built in a eventhandler (addEventHandler) or to remove it again (removeEventHandler).

Adding an Eventhandler

To add an Eventhandler just use the following Syntax:

Name addEventHandler ["Eventhandler", Code]

Code stands for the action which has to be caused when the condition / event has been executed. One can define nearly everythings what one wants to do. In the following example the Eventhandler **Killed** was given and as code a script which will be caused so far the Event, the death of the respective unit, became true.

Name addEventHandler ["Killed", {this exec "script.sqs"}]

The Eventhandler will be given to several units within the next example. If one of these units will be killed, so the predefined text „**Test**“ will appear on the screen.

{_x addEventHandler ["Killed", {hint "Test"}]} foreach [Name1, Name2,...]

A further great example is the textmessage which displays the information who killed who. This option is a special feature espacially for the Multiplayer games. The Eventhandler needs to be given to all units which are dedicated to. The Syntax looks as follows:

**{_x addEventHandler ["Killed", {hint format["%1 killed by %2",
_this select 0, _this select 1}]} foreach [S1, S2, S3, S4]**

If the unit **S1** is been killed by unit **S2**, so the name **S1** will be displayed within a **hint**, as defined in the Editor. **S1** killed by **S2**. The hint format Syntax individually:

hint format ["%1 killed by %2",_this select 0, _this select 1]

Defining an Eventhandler for a unit within a Area

It's possible to give an Eventhandler for several units within a trigger area. In example the **Fired** eventhandler, which is causing the **Alarm.sqs**.

{_x addEventHandler ["Fired",{this exec "Alarm.sqs"}]} foreach thisList

Removing an Eventhandler again:

They can be removed as simple as the were implanted. Just use the following Syntax:

Name removeEventHandler ["Killed", 0]

Overview about most of the Eventhandlers:

The table below is explaining most of the existing Eventhandlers including a syntax example.

Eventhandler	Example
AnimChanged	Syntax: Name addEventHandler ["AnimChanged", {{Einheit, Animation}}] Will always be caused when the predefined animation is getting started.
AnimDone	Syntax: Name addEventHandler ["AnimDone", {{Einheit, Animation}}] Will always be caused when the predefined animation is getting ended.
Dammaged	Syntax: Name addEventHandler ["Dammaged", {hint str (_ this)}] Will be caused when the respective unit was damaged, also displays the kind of damage.
Engine	Syntax: Name addEventHandler ["Engine", {hint str (_ this)}] Will be caused when the engine of the vehicle gets started.
Fired	Syntax: Name addEventHandler ["Fired", {hint "Hold fire!"}] This Eventhandler will be caused if the unit is firing its weapon.
Fuel	Syntax: Name addEventHandler ["Fuel", {hint str (_ this)}] Will be caused when the fuelstatus has been changed (1=full/0=empty)
Gear	Syntax: Name addEventHandler ["Gear", GearState] GearState: TRUE – Gear down/ FALSE – Gear up
GetIn	Syntax: Name addEventHandler [Name, VehiclePosition, VehicleName] Will be caused if the unit is getting into the vehicle Positions: Driver, Gunner, Commander, Cargo
GetOut	Syntax: Name addEventHandler [Name, VehiclePosition, VehicleName] Will be caused if the unit is getting again out of the vehicle.
Hit	Syntax: Name addEventHandler ["Hit", {hint "This unit were hit!"}] This action is unfortunately not always causing.
Init	Syntax: Name addEventHandler [] Only to be used in a config! Will be caused atumatically when the mission is started, not really important.
IncomingMissile	Syntax: Name addEventHandler ["IncomingMissile", {hint "Warning!"}] Will be caused if the unit is getting shot by a rocket.
Killed	Syntax: Name addEventHandler ["Killed", {_ this exec "Script.sqs"}] Will be caused if the unit has been killed.

5.66 - Different text displays

Text Displays are, as already known, not only important for a mission but also for editing or scripting. One has the possibility to get certain information displayed on screen to work with later in the editing or scripting process or to use for other things. The following contents will explain the different Text Displays in Armed Assault®. This will also explain how to implement Text Displays and Text variables.

The Hint-Variant

At first there's the hint variant, which will create text in the upper left corner on the screen. This text can be adjusted to the user's requirements.

Default: **hint "My Text"**

The information about the coordinates of the player over sea level:

```
hint format ["Coordinates: %1", getPosASL Player]
```

```
hint format ["Your Altitude is %1", getPosASL Player select 2]
```

Or the information about the weapons, resp. the kind of ammunition or the side which this unit belongs to:

```
hint format ["%1", weapons Name];
```

```
hint format ["%1", magazines this];
```

```
hint format ["%1", side Name];
```

```
hint format ["%1 versus %2", side Name1, side Name2];
```

Used with an Event handler:

```
Name addEventHandler ["Fired", {hint format ["%1", _this]}]
```

Used with a Stringtable value:

```
hint localize "STR_MP_04"
```

The Title Text-Variant

The title text variant works as well as the hint variant does. The only difference is that the text will not appear in the upper left corner of the screen, but at a position defined by the user. For example, right in the middle of the screen, or even on the bottom.

Default: **titleText ["Paraiso\nOne day later...", "Plain",4]**

Variant to get the distance between two units or the Player displayed:

```
titleText [format["%1 Meter", Name1 distance Name2],"Plain Down"]
```

```
titleText [format["%1 Meter", getPosASL Player select 2],"Plain Down"]
```

Used with a Stringtable value:

```
titleText [format [localize "STR_ART_H7"], "Plain Down"];
```

5.67 - Stringtable Basic Values

There are lots of text variables already predefined in Armed Assault® which are used as text displays all over in the game and in the menus. Some of these entries are well fit to be used with the user's mission without defining a new Stringtable.csv. The table below will show some of the most important entries which are already predefined in ArmaA®. They can easily be used just by using the hint or the Title-Text command.

Source	Text
STR_ARMEDASSAULT_CAPITAL	Armed Assault
STR_ARMEDASSAULT	ARMED ASSAULT
STR_ARMA_SPLASH_1	Präsentiert
STR_SAHHRANI_ISLAND	Sahrani Island
STR_SAHHRANI	Sahrani
STR_SAHHRANI_SOUTH	The Kingdom South-Sahrani
STR_SAHHRANI_NORTH	Democratic Republic Sahrani
MISSION_COMPLETED_CA	Mission accomplished
MISSION_FAILED_CA	Mission failed
MISSION_SUCCESFULL	Mission successful
MISSION_DEFEATED	Enemy forces defeated
STR_MISSION_OBEY	Mission failed, you haven't followed your orders.
STR_MP_GAME_DESC_OBJECTIVES	Destroy the enemy targets
STR_MP_GAME_DESC_PILOTDOWN	Find and rescue the Pilot.
STR_MP_GAME_DESC_CSWEAP	Come home alive!
STR_MP_GAME_DESC_ESCAPE	Escape from the Island with the Helicopter!
STR_MP_GAME_DESC_WARCRY	A massive planed Operation: Set the Town free.
STR_MP_GAME_DESC_CITYCONFLICT	Conquer and defend the Town.
STR_ART_H1	Click on the map
STR_ART_H8	ENTER VALID COORDINATES FIRST!
STR_ART_H7	TARGET COORDINATES ACCEPTED
STR_ART_H3	FIREMISSION WILL GET STARTED
STR_ART_H6	OUT OF RANGE SELECT NEW TARGET POSITION!
STR_ART_H9	FIREMISSION OVER
STR_ART_H10	CAN NOT FIRE
STR_MARKER_START	Infiltration POINT
STR_WP_HOLD	Keep your position
STR_WP_BASE	Base
STR_WP_ENEMYBASE	Enemy Base
STR_WP_MEETINGPOINT	Meeting Point
STR_WP_PICKUP	Pick up position
STR_WP_SAD	Seek and destroy
STR_WP_ATTACK	Attack
STR_WP_DESTROY	Destroy
STR_WP_DESTROYTARGET	Destroy target
STR_WP_SEIZEANDHOLD	Conquer and defend
STR_WP_EXTRACT	Exfiltration Point
STR_MP_GAME_DESC_HUNTING	Transport strategic resources under fire

5.68 - Create Waypoints

Arma® offers the possibility to create or delete Waypoints for units or groups while running a mission. These waypoints can also be equipped with certain types and functions.

Syntax:

WP1 = Name addWaypoint [position, Radius]

The following example will create a waypoint called WP1 for Group 1, defined with a radius of 100 meters right on the position of the Player.

WP1 = Group1 addWaypoint [position Player, 100]

This waypoint will also get a function allocated, which is listed in **Chapter 1.5**. In this case this waypoint with name **WP1** will receive the function "Sentry".

WP1 setWaypointType "SENTRY"

[Group1, 2] setWaypointType "SENTRY"

Additionally to this there are a lot of further possibilities to adjust this waypoint. The basic definitions to define a waypoint can be found in **Chapter 1.5** within the sub-item - Insert Waypoint. Additionally to this, the following commands are available.

Command	Example
	[Name, Waypoint Number] setWaypoint... ..
setWaypointBehaviour	[Group1, 2] setWaypointBehaviour "AWARE" Waypoint name setWaypointBehaviour "AWARE"
setWaypointCombatMode	[Group1, 2] setWaypointCombatMode "RED"
setWaypointDescription	[Group1, 2] setWaypointDescription "Go to Position!"
setWaypointFormation	[Group1, 2] setWaypointFormation "LINE"
setWaypointHousePosition	[Group1, 2] setWaypointHousePosition 1
setWaypointPosition	[Group1, 2] setWaypointPosition [position player, 0]
setWaypointScript	[Group1, 2] setWaypointScript "Target.sqs player"
setWaypointSpeed	[Group1, 2] setWaypointSpeed "FULL"
setWaypointStatements	[Group1, 2] setWaypointStatements ["true", ""]
setWaypointTimeout	[Group1, 2] setWaypointTimeout [5, 10, 6]
setWaypointType	[Group1, 2] setWaypointType "HOLD"
showWaypoint	[Group1, 2] showWaypoint "ALWAYS"
setWpPos	[Group1, 2] setWpPos [x, y, z]
getWpPos	_Pos = getWpPos [Group1,1]

Remove Waypoint

One can remove waypoints as easily as they can be created. To do this just use following syntax:

deleteWaypoint [Name, Waypoint Number]

deleteWaypoint Waypoint Name

5.69 - Create Trigger

Triggers are easily created as units, objects and waypoints, which can give many new editing possibilities to the user. This part will explain how to create triggers and also how to adjust them by using syntaxes.

Because a trigger needs, up to the user's plans, a little more than just a short syntax, I suggest to do this by using scripts or functions.

Here is an overview with the respective explanations.

CreateTrigger

This command will create a default trigger first. This one can basically be defined with a name. Although it is already defined with a name, it's nevertheless still rough and needs to be adjusted by using the following commands. But first the creation:

```
ASL1 = createTrigger ["EmptyDetector", position Player]
```

```
ASL1 = createTrigger ["EmptyDetector", [X,Y,Z]]
```

The trigger with the name **ASL1** will be created right on the position of the player or even on given coordinates.

SetTriggerArea

The trigger **ASL1** still needs to get a size, angle and conformation which will be defined as follows. While setting these options, without the conformation, will be defined in numbers. The conformation, so circle or rectangle are to be defined by using true or false, where true stands for rectangle and false for circle.

```
ASL1 setTriggerArea [X, Y, Angle, Form]
```

SetTriggerText

Now one has the possibility to define a text to the trigger. This is basically sensible by using Radio Triggers. That's why one would already have the respective text displayed in the Radio on the map.

```
ASL1 setTriggerText "Artillery-Support"
```

SetTriggerTimeout

This option will define the Timeout of the trigger and whether it shall work as a countdown or even timeout. The values for the times have to be defined as numbers again.

```
ASL1 setTriggerTimeout [Min, Mid, Max, False]
```

SetTriggerType

Additionally to all the options explained above, a Type can also be defined. These are also predefined in the Editor.

ASL1 setTriggerType "WIN"

Overview of the Types:

"NONE"	None
"WEST G"	Guarded by West
"EAST G"	Guarded by East
"GUER G"	Guarded by Resistance
"SWITCH"	switch
"END1"	End 1
"END2"	End 2
"END3"	End 3
"END4"	End 4
"END5"	End 5
"END6"	End 6
"LOOSE"	Loose
"WIN"	Win
"WEST SEIZED"	Conquered by West
"EAST SEIZED"	Conquered by East
"GUER SEIZED"	Conquered by Resistance

SetTriggerActivation

This option will define the kind of activation, whether activated by West, East, by Radio or whatever. The syntax:

ASL1 setTriggerActivation ["WEST", "EAST D", true]

This trigger was defined as follows:

Activation by West // Detected by East // Repeatedly

First part of the array:

Side: "NONE", "EAST", "WEST", "GUER", "CIV", "LOGIC", "ANY"
Radio: "ALPHA", "BRAVO", "CHARLIE", "DELTA", "ECHO",
"FOXTROT", "GOLF", "HOTEL", "INDIA", "JULIET",
Others: "STATIC", "VEHICLE", "GROUP", "LEADER", "MEMBER".

Second part of the array:

"PRESENT", "NOT PRESENT", "WEST D", "EAST D", "GUER D", "CIV D".

Third part of the array:

In this case **true** means repeatedly and **false** means only a single activation of the trigger.

SetTriggerStatements

The Condition line, the OnActivation line and the OnDeactivation line, of the newly created trigger, will be defined with this option. Please take care about the correct diction within the Array!

ASL1 setTriggerStatements [Condition, Activation, Deactivation]

ASL1 setTriggerStatements ["this", "Value = true", "Value = False"]

SetMusicEffect

This option will set the Music effects of the trigger.

ASL1 setMusicEffect "ATrack1"

The default or original music tracks don't need a dedicated entry within the Description.ext and are listed from **ATrack1** to **ATrack27**. The names of the Queens Gambit Music tracks are listed from **QGTrack1** till **QGTrack9**.

SetSoundEffect

This option will add a sound effect to the trigger.

ASL1 setSoundEffect [Anonymous, Voice, Environment, Trigger]

ASL1 setSoundEffect ["Alert", "", "", ""]

Following is a little list of some available sounds:

Stream, Alarm, BadDog, BirdSinging, Chicken, Cock, Cow, Crow, Crickets1, Crickets2, Crickets3, Crickets4, Dog, Frog, Frogs, LittleDog, Music, Owl, Wolf

5.70 - Create Marker

One has the possibility to create markers globally or locally. This takes the advantage to the Local ones that not everyone can see them, but also only those who are dedicated to. It's possible to create markers even for a single Side or even for a special player only.

CreateMarker

By using this command a marker will be created and will also receive a name and a position. Note: Normally the marker will be set in quotes while calling. This is not really necessary for the created marker. But it's not a bad practice to define all markers, whether they are used normally or belately created, in quotes "".

_Marker1 = createMarker ["M1", position Player]

_Marker1 = createMarker ["M1", [X,Y]]

SetMarkerType

The type of the marker will be defined by using the parameter SetMarkerType.

"M1" setMarkerType "Arrow"

The list below will give an overview of the available icon types for the markers:

Objective (Flag)	Arrow	SalvageVehicle
Flag1	Empty	RepairVehicle
Dot	Select	SupplyVehicle
Destroy	Vehicle	DestroyedVehicle
Start	Defend	MaintenanceTeam
End	Move	CommandTeam
Warning	Attack	SupplyTeam
Join	Headquarters	InfantryTeam
PickUp	Depot	LightTeam
Unknown	Camp	HeavyTeam
Marker	Town	AirTeam
		FireMission

SetMarkerText

Thy text of the marker will be defined by using this syntax:

"M1" setMarkerText "Hold Position"

SetMarkerShape

The shape of the marker will be defined by using this syntax. Whether it should be an icon, rectangle or even a circle. It's not really necessary to define the size for icons because they will always be displayed by default size, but not the rectangle or the circle, so it's important to define a size for those.

"M1" setMarkerShape "ELLIPSE"

One will have the following selections:

"RECTANGLE" - "ELLIPSE" - "ICON"

SetMarkerBrush

One can define the look of a marker by using the MarkerShape Rectangle or MarkerShape Ellipse command

"M1" setMarkerBrush "FDiagonal"

The following designs can be selected:

Horizontal	- Horizontal Lines
Vertical	- Vertical Lines
Grid	- Horizontal Grid
DiagGrid	- Sloping Grid
FDiagonal	- Sloping Grid
BDiagonal	- Sloping Grid
Cross	- Cross Grid

SetMarkerColor

The color of the marker will be defined by using this syntax. The markers will always be displayed in red by default. If one wants to keep that color, this syntax is not needed.

"M1" setMarkerColor "ColorBlue"

The following colors can be selected

"ColorRed"	"ColorGreen"	"ColorBlue"
"ColorRedAlpha"	"ColorGreenAlpha"	"ColorYellow"
"ColorBlack"	"ColorWhite"	"Default"

SetMarkerSize

The size of the marker can be defined by this Syntax. This one needs to be defined as follows:

"M1" setMarkerSize [100, 200]

SetMarkerDir

The direction of the marker will be defined by using this Syntax. The definition has to be made in degrees (1-360).

"M1" setMarkerDir 90

Delete Marker

If a marker has to be deleted later in the mission, just use this already known command::

deleteMarker "M1"

Create a local marker

The creation and definition of the local marker works as same as for the global one, but the only difference is that this definition is related to the global markers only. The commands are always the same but only the expansion Local.

I will nevertheless show the collection of the locally related commands:

CreateMarkerLocal	SetMarkerTypeLocal	SetMarkerColorLocal
SetMarkerSizeLocal	SetMarkerShapeLocal	SetMarkerBrushLocal
SetMarkerTextLocal	SetMarkerDirLocal	SetMarkerPosLocal
DeleteMarkerLocal		

Further and important information can be found in the **Chapter 1.7 - Adding Markers**.

5.71 - All about vehicles

This subchapter will explain a few further things about vehicles. It's possible to become lots of different information which can be used or just called. That's why this really important conclusion is written here again.

Ask for or allocate a damage

It's possible to become a vehicle directly damaged, or check whether a vehicle is damaged within a trigger area. It's also possible to check whether the vehicle is generally still able to drive.

- ? getDamage Vehicle1 > 0.5** - If damage is bigger than 0.5, then...
- ?!(canMove Vehicle1)** - If vehicle is no more able to drive, then...
- ?!(alive Vehicle1)** - If vehicle is nor more alive, then...
- Vehicle1 setDamage 0.5** - Set a damage of 0.5
- {_x setDamage 1} forEach crew Vehicle1** - Whole crew gets damaged

The first three syntaxes can be entered as usual within the condition line of a checking trigger. Please remove the question mark right there, but leave the question mark while using this syntax within a script.

Unit in vehicle

By using this syntax, one can check whether a special unit is still inside a vehicle:

Name in Vehicle
Player in (crew Vehicle)

Typ des Vehikels abfragen

Some things shall get caused so far a certain unit is entering a vehicle. To do this just use the following syntax and enter the class-name of the respective vehicle between the quotes.

typeof vehicle Player == "M1030"
typeof vehicle Name == "AH1W"

Become vehicles locked for certain units

One can check the type of a unit by using the command **TypeOf** and also lock a certain type of unit out of a vehicle. That's necessary to make sure that just the unitType Pilot is able to fly an aircraft. It's possible realize this either by using a script or a checking trigger.

Checking trigger

Activation: Repeatedly
Axis a/b: 0
Condition: typeof driver Vehicle1 != "SoldierWPilot"
onActivation: Driver Vehicle1 action ["Eject",Vehicle1]

If another player equipped with a different unit type will enter the vehicle, The trigger will execute causing the unit to be ejected from the vehicle.

Lock vehicle for certain Name

If a certain unit or the player shouldn't have access to a special vehicle or even a special vehicle position (i.e. Gunner), one can do this by using the following syntax. In this case a Checking Trigger and a script example are used.

Checking trigger:

Activation: Repeatedly
Axis a/b: 0
Condition: Vehicle **Name1** == Vehicle1
on Activation: Name1 action ["Eject", Vehicle1]

Script Syntaxes:

? vehicle **Name1** == **Vehicle1** : **Name1** action ["Eject", **Vehicle1**]
? driver **Vehicle1** == **Name1** : **Name1** action ["Eject", **Vehicle1**]
? gunner **Vehicle1** == **Name1** : **Name1** action ["Eject", **Vehicle1**]
? commander **Vehicle1** == **Name1** : **Name1** action ["Eject", **Vehicle1**]

Ask/Set speed of a special vehicle

To use a speed of a vehicle for a condition for something else, just use this syntax:

speed Vehicle1 > 30

And to set the speed of a certain vehicle (Full, Normal, Limited, Auto):

Vehicle1 setSpeedMode "Full"

Vehicle1 forceSpeed 120 - Value as km/h

Vehicle1 limitSpeed 60 - Value as km/h

Driver or Gunner present?

The command **isNull** will check whether a special position within a vehicle is still empty.

isNull driver Vehicle1 - If driver position is empty, then...

! isNull driver Vehicle1 - If driver position is empty, then...

Vehicle with still running machine

The command **isEngineOn** is checking whether the engine of a certain vehicle is still running or not.

isEngineOn Vehicle1 - If machine of Vehicle1 is still running, then...

Fuel Status

The following orders will be used to check the Fuel status of a vehicle:

Fuel Vehicle1 == 0.5 - Is fuel status is as same as value, then...

Vehicle1 setFuel 0.8 - Is setting a fixed fuel status

Vehicle1 setFuelCargo 1 - Is setting fuel status for refuel truck

5.72 - Create a light source

It's possible to create light sources just by using scripts or functions. The following example will explain how to create a light source at the position of an object. This one will get started by using the following syntax:

```
[ObjectName] exec "scripts\light.sqs"
```

Light.sqs

```
_object = _this select 0;
_light = "#lightpoint" createVehicle position _object;
_light setLightColor [0, 0, 8];
_light setLightAmbient [0.9, 0.9, 0.9];
_light setLightBrightness (0.1 / 0.1);
exit;
```

One has the possibility to adjust the color of the light, so it might be that the most incredible results will appear. One only would need to change the respective value within the array. Right when the change has been saved, it will immediately be visible in the preview. The use of the decimal values becomes the results effective.

- _light setLightColor [0, 0, 1]** - Defining the light color
- _light setLightAmbient [0, 0, 1]** - Defining of the environment light
- _light setLightBrightness (0.1 / 0.1)** - Defining the brightness of the light

Such a light can also be created locally. One would have to use the local create command, which looks as follows:

```
_light = "#lightpoint" createVehicleLocal _pos
```

5.73 - Create Dust

The following example will create a small dust cloud. This one can be adjusted in the size just by changing the values. It's also possible to adjust the color and the shape.

```
[ObjectName] exec "scripts\dust.sqs"
```

Dust.sqs

```
_object = _this select 0
_dust = "#particlesource" createVehicle position _object;
_dust setParticleParams [["\Ca\Data\Cl_basic.p3d", 1, 0, 1], "", "Billboard", 1, 0.7, [0, 0, 0],
                        [0,0,0], 5, 0.2, 0.2, 0.0, [0.8, 1], [[0.7,0.6,0.5,0.25],[0.7,0.6,0.5,0.0]],
                        [0, 1], 1, 0, "", "", _object];
_dust setParticleRandom [0, [0.5, 1.5, 0], [1, 1, 0], 0, 0, [0, 0, 0, 0], 0, 0];
_dust setParticleCircle [0, [0, 0, 0]];
_dust setDropInterval 0.02;
~2
deleteVehicle _dust
exit;
```

5.74 - Create Smoke

A little example about how to create a column of smoke in Armed Assault®. It should be clear that this feature is still very expandable, but this small example is much more than just a good beginning about this feature.

To create smoke or fire, a special command is needed which is called Drop-command. As one can see in the picture below, those command lines can be very long, because both of these lines, beginning with drop, are representing actually one single line. It's not possible here to write them in one line, otherwise one wouldn't be able to see anything. The most different things like color, shape, behavior and speed will be defined within this array. This example below represents a loop. Each time the script is running through it, a value of **1** is added to the local value `_i`. If the local Value `_i` will reach the predefined value of **800**, the script will end and the smoke disappears.

The values are variable and can be adjusted by the user. The script will be run by using following syntax.

```
[ObjectName] exec "scripts\smoke.sqs"
```

Smoke.sqs

```
_vehicle = _this select 0
_position = [0, 0, 0]
_i = 0

#Loop
_i = _i + 1
drop ["\ca\data\cl_basic", "", "Billboard", 6, 6, _position, [0,0,0], 1, 1, 1.5, 0.02,
      [1,30], [[0,0,0],[0,0,0,0.7],[0,0,0,0]], [0], 1.2, 1, "", "", _vehicle]
~0.1
? _i == 800 : exit
goto "Loop"
```

This script is well to be used when a refuel truck or something similar has been destroyed for example. Here is a checking trigger example:

Checking trigger:

Activation: Once
Axis a/b: 0
Condition: not alive `Vehicle1`
onActivation: `[Vehicle1] exec "scripts\smoke.sqs"`

It's furthermore possible to define the position around the object or the vehicle. To do this the coordinates (XYZ-values) are to be used again, which are defined within the array.

```
_position = [0, 0, 0]
```

5.75 - Creating Fire

The following example shows how to create fire at any given object position. Even here there is a lot of scope for making modifications. In addition, the internal script values can be adjusted at any time to suit your requirements. The initial start array offers many free definition possibilities.

[Objects,60,0] exec "scripts\fire.sqs"

Object refers to the name of the object, the **second** value refers to the length of time that the object should burn and the **third** value determines the height position above the object that the fire will be placed.

Fire.sqs

```
_object=_ this select 0;
_time=_ this select 1;
_zpos=_ this select 2;
_delay=0.15;
_timecheck=0;
_i=0
;The following line represents a line of code that is not possible here:
_light = "#lightpoint" createVehicle [(getPos _object select 0),
                                     (getPos _object select 1),_zpos];
_light setLightBrightness 0.03;
_light setLightAmbient[0.03, 0.028, 0.0];
_light setLightColor[1.0, 0.9, 0.0];
#Loop
_pos=getPos _object
_x=(_pos select 0)
_y=(_pos select 1)
_z=(_pos select 2)
_vx=0.2-(random 0.4)
_vy=0.2-(random 0.4)
_vz=random 0.15
_m=(2.1-(random 0.1))
_Fire=[2,6] select (random 1.5)
;The following line represents a line of code that is not possible here:
drop [["\ca\data\ParticleEffects\FireAndSmokeAnim\FireAnim",8,_Fire,32],
      "", "Billboard", 9, (0.7+(random 0.3)), [_x,_y,_z+_zpos+(random 1)],
      [_vx,_vy,_vz], 0, _m, 2, 1, [0.5,1,2],[[1,1,1,0.5],[1,1,1,1],[1,1,1,0]], [0],
      0,0,"",,logic]
```

Continued on the following page...!

```

? _i<=3: goto "Loop2"
_Smoke=3;
_s=(2.2-(random 0.1));

; The following line represents a line of code that is not possible here:
drop [{"ca\data\ParticleEffects\FireAndSmokeAnim\SmokeAnim",8,
      _Smoke,32}, "", "Billboard", 9, (2.5+(random 0.5)), [_x,_y,_z],
      [_vx,_vy,_vz], 0, _s, 2, 2, [0.2,5],[[0,0,0,0],[1,1,1,0.8],[1,1,1,0]], [0],
      0,0,"",,logic]; _i=0;

#Loop2
~_delay
_i=_i+1

_timecheck=_timecheck+_delay
? _time>=_timecheck: goto "Loop"
deleteVehicle _light
exit

```

This script contains, among other things, a light source, a fire-drop command and additionally a smoke-drop command, combining together the **Sub-Chapters 5.73, 5.74** and **5.75** into one action. The drop-commands have to be naturally in a line of code!

The following picture shows an engine on fire:



5.76 - Assigning ranks

It's possible to become a soldier ranked up while the Mission is running. That means that the user has the possibility to rank the unit up or even degrade for good or bad achievements.

So following Ranks are possible to use:

Private
Corporal
Sergeant
Lieutenant
Captain
Major
Colonel

These ranks can be assigned by using the following syntax:

Name1 setRank "Major" or **Player setRank "Major"**

Example of usage

The player should be promoted if a certain amount of points have been exceeded that's possible to become true on several ways. The variant with using a checking Trigger shall serve as example here

Checking Trigger:

Activation: Einfach
Axis a/b: 0/0
Condition: Rating Player >= Value
onActivation: Player setRank "Major";
hint format ["You've just been ranked up to %1!",rank Player]

The Player will be ranked up now.

As Condition

The rank of a unit can now be used as a condition for something else. For example, when assigning weapons or the use of vehicles. At first the use of vehicles or certain types of weapons could be disabled for a player. Due to the fact that a strong performance is rewarded accordingly, the player will make more of an effort and try to collect points to achieve promotion as quickly as possible.

? rank player == "Major" : exit

? rank Name1 != "Major" : exit

5.77 - Unit using Binoculars

This possibility is not quite useful for a mission but a nice feature for Camera sequences. But before the unit will use its binoculars so it has to be assigned at first. One can do this by using following syntax:

Name1 addWeapon "Binocular";

Once the Unit has been equipped with the binoculars, the unit has to make use of the binoculars. The best way to realize this is the SelectWeapon-command. It's only needed now to order the unit to use its Binoculars by using a script or a trigger. To do this just use following Syntax.

Name1 selectWeapon "Binocular"; Name1 setBehaviour "safe";

Of course it is also possible to implement this with an animation command, as described thoroughly in **Chapter 5.56**.

5.78 - Assigning a unit to a vehicle seat

It is possible to assign a unit to a vehicle seat. However, after the assignation the unit will not automatically sit in the seat. So what is the point of this setting? It allows the unit to be positioned next to the vehicle and then, at the sound of an alarm for example, to be given an order to get into the vehicle. This is more akin to a real-life situation as no soldier would sit constantly in the firing position if there's not yet a reason to do so. In this way MG's and searchlights too could also be unoccupied in the first instance and then at the sound of an alarm the units would storm to their assigned vehicles. Additionally, the player also has the possibility to switch off the occupying of troops of a tank before it's been entered. To do this the following commands has to be used:

- | | |
|-------------------------------------|---|
| Name assignAsCargo Truck1 | - Assigns a unit to a passenger seat |
| Name assignAsDriver Tank1 | - Assigns a unit to the driver seat |
| Name assignAsGunner Tank1 | - Assigns a unit to the gunner position |
| Name assignAsCommander Tank1 | - Assigns a unit to the commander seat |

Use **unAssignVehicleName** to remove a unit of a vehicle again. The unit now knows that the unassigned vehicle no longer belongs to it and won't automatically get back into the vehicle after receiving the command to get out of the vehicle

After a unit has been assigned to a vehicle, it can be given the command to get in by using the following syntax:

- | | |
|--------------------------------|-----------------------------|
| [Name] orderGetIn true | - Unit is getting in |
| [Name] orderGetIn false | - Unit is getting out again |

If several units have to be getting in a vehicle, just add their names right into the respective Field of the Syntax, like this:

[Name1, Name2, Name3] orderGetIn true

One has the further possibility to get the vehicle displayed which was assigned to the unit or which place it has received inside the car. Especially a player can use more or all positions of a vehicle. That fact can be used as condition for somethings else as you can see in **Chapter 5.71**.

Displaying allocated Vehicle:

hint format ["**Allocated vehicle: %1**", assignedVehicle **Player**]

Displaying current Position inside the Vehicle:

hint format ["**Position in vehicle: %1**", assignedVehicleRole **Player**]

5.79 - Allocate a unit to a team

If the Player has to control a Group leader so he has the Possibility to divide its Group members in several smaller Teams. These ones would receive different colours then to differ them from each other. So one can allocate a single Unit to an already existing Team:

Name assignTeam "BLUE"

To un-divide a Unit and Team again just use:

unassignTeam Name

To delete the whole Team again just use following Syntax

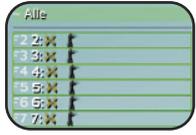
dissolveTeam "BLUE"

Following Team colours are to be used:

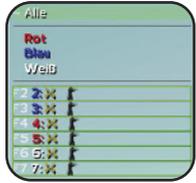
- MAIN
- RED
- GREEN
- BLUE
- YELLOW

Later in the Editor/Mission the result will be loo like this:

No Team



Blue and red Team



Blue, red and Green Team



5.80 - Unit is giving out a command

By using following Syntax one can enable a Unit to give out a command which will be executed even by herself or by another unit. The Example below explains some possibilities. In following examples you will see some possibilities of this commands.

- Name1 commandFire Name2** - Name1 is firing on Name2
- Name1 commandTarget Name2** - Name1 gets Name2 as target
- Name1 commandMove Name2** - Name1 moves to Name2
- Name1 commandStop Name2** - Name1 keeps standing

It's also possible to do this with using Groups. And at this point the sense of the whole command comes out much more. If the Leader of a Group is giving out a command, so this one will be spoken out listenable and the group members are all executing this command.

(units group Name1) commandFire Name2

5.81 - Has a unit recieved damage?

It's basically possible to request the damage of any unit or any Object for using them i.e. as condition for somethings else. Their different possibilities available to become it realized. I.e. whether a unit is still able to walk/run or to shoot. The following commands can be used.

canStand - canMove - canFire - handsHit - getDammage

Following a few syntax examples used in a script:

- ?! (canStand Name) : Name sideChat "Aah... my legs!"**
- ?! (canMove Name) : Name sideChat "My car gas been damaged!"**
- ?! (canFire Name) : hint "Name can't fire!"**
- ? (handsHit Name == 1) : Name sideChat "Aah... my hands!"**
- ? (getDammage Player) > 0.5 : hint "You are hurt!"**

Bedenke:

Never use a questionmark (?) in the conditionline of a trigger, cause its already intigrated. So following a small trigger example:

Axis a/b: 0
Condition: (handsHit Name == 1)
on Activation: Name sideChat "Aah... my hands!"

5.82 - The air traffic

Even the Air traffic has its System and its order. So there are some things as well, where one has to take care about. At first it's possible to keep in mind that the Start Airfield is always the Home Airfield. An aircraft would always get the value of its Home Airport allocated automatically. But if the mission begins in the air so the aircraft will get the very next Airfield allocated.

So every single Airfield has its own value which can get allocated to any aircraft. The List below shows the current Airfields and their Parameters:

- 0** - **Paraiso**
- 1** - **Rahmadi**
- 2** - **Pita**
- 3** - **Antigua**

If one wants to allocate an Airfield to the Aircraft, so just use this Syntax:

Plane1 assignToAirport 1

It's also possible now to allow the Aircraft to land on one of the other Airfields.

Plane1 landAt 1

Once all important aircraft have been given, that each Airport has a fixed index and all Aircrafts can receive other Airports to land, there's actually only one thing still missing, The respective Side.

The Side of each Airport can be given by using this Syntax:

0 setAirportSide WEST

WEST - EAST - GUER - CIV - LOGIC



5.83 - Decrease grass details

It's quite simple to decrease or even to remove the grass on ArMA's Islands. It's a very useful possibility to become a grass free landscape like Operation Flashpoint®. The results of decreasing or removing the grass are different. Either one wants to save the users performance in Multiplayer Games or while crating a camera sequence or whatever.

The syntax will be

setTerrainGrid 12.5

There're fixed values given. If one defines another, not really existing value, so the engine will select the next available value, automatically. The values are predefined on a quite unusual way. So smaller the value so higher the grass:

50	- No Grass
25	- Less Grass
12.5	- Middle Grass
6.25	- Default Grass
3.125	- High Grass details

5.84 - Place sloped Objects

By using following Syntax it's possible to place Objects on a sloped way on the map. That's a very unusual but sometimes necessary way to place units. This may be for special camera sequences for example where a Tank is lying on its side or even on its head. It works for all objects and vehicles of any kind, aircrafts, objects or whatever. All values between **-1** and **1** will work.

Following Syntax's have to be used:

Object setVectorUp [0,0,0]

Object setVectorDir [0,0,0]

It's also possible to combine both variants:

Object setVectorDirAndUp [[0,0,0],[0,0,0]]

The order of coordinates is not XYZ but YZY!

Object setVectorUp [x,z,y]



5.85 - Lock or unlock missions

Some original ArMA® Missions are using this system. That means, one can only play one of these missions once he has absolved his training for example or has successfully finished other missions.

That wouldn't make any sense for only one released mission. But if one has a combination of several Missions so it makes sense again. May be he wants to keep the order of playing the missions.

The needed keys need to be defined in the description.ext. The following rules are to be followed:

- keys[] = {"Key1", "Key2", "Key3"};** - The keys have to be defined right here
- keysLimit = 2;** - Minimum number of needed keys to unlock the mission again
- doneKeys[] = {"Key4"};** - Name of the key which has to be checked in the mission plan of the SP Mission

If the mission was finished successful so the following key needs to be entered:

activateKey "Key1"

One can find more Information in **Chapter 11.2 - The ArMA Cheats**.

5.86 - Empty searchlight with light

If the user is placing an AI used searchlight on the map, this lamp will be switched on automatically at night. But if the position behind the searchlight is empty so there's no way to activate the Light. If one wants to light up this Lamp even when there's no unit behind, a game logic is needed.

To do this so an empty named searchlight (may be **Light1**) has to be placed on the map before. The **Game Logic** has to be placed on the map as well. Now enter following Syntax into the init line of the **Game Logic**:

this movelnGunner Light1

So the Game logic has turned to the AI Position of the searchlight. But the disadvantage is the Game Logic will not move the searchlight. But by using the script example from **Chapter 6.14** it's possible to make it move again.

The same is working with vehicles. If one wants to place an empty vehicle on the map, with nevertheless switched on lights, so one can handle this with a **GameLogic** again!

Chapter 6

- Mission Specials -

After you read the first 5 chapters, hopefully quite attentively, you'll get some more information and specials for your mission. These can be realized with a little exertion quickly and easily. Your mission will be much more interesting and exciting by utilizing some of these specials. All features of this chapter have been built basically on scripts, but that will not make them less functional, they work as well as functions would.

Because all of the examples are extensive, and retyping them could create errors in the scripts, you can download the scripts with additional example missions from the forums at www.forum.german-gamers-club.de or www.mapfact.net, which I've uploaded there. .

6.1	The paratroopers	179
6.2	The GPS-System	180
6.3	The action menu entry	181
6.4	The backpack	181
6.5	Random positions	185
6.6	The mapclick	187
6.7	The artillery	189
6.8	Deleting killed units and vehicles	194
6.9	Suppressing gaming speed constantly	195
6.10	The bullet mode	196
6.11	Track down enemy units	197
6.12	The air strike	198
6.13	The air vehicle creator	201
6.14	The searchlight	203
6.15	The time counter	204
6.16	The house patrol script	205
6.17	The mine script	208
6.18	The vehicle transport script	209
6.19	The seagull script	213
6.20	The insect script	215
6.21	The saboteur	216
6.22	The spotter	217
6.23	Unit is capitulating itself	218
6.24	The teleport	221
6.25	The persecution script	222

6.1 - The paratroopers

Paratroopers are always a nice feature in missions, so I will explain one of several variants here.

The Helicopter

In this example we place a helicopter named **Heli1** on the map. If the user doesn't want the helicopter to take off right when the mission begins, just set the fuel status down to empty by using the command **this setFuel 0**. When the helicopter has to take off later in the mission, just set the fuel status back to **1** by using the command **this setFuel 1**. At the time we have a disadvantage here, because the helicopter crew will exit the chopper. We have to hope that one of the following patches will fix that problem.

The altitude

The altitude should be set up to **80** or **100** metres or your troopers will get hurt and probably die. To avoid this just use the following syntax in the respective waypoint of the chopper:

Heli1 flyInHeight 120

The landing zone

The landing zone should be selected far way from villages or forests to offer a good landing zone for the soldiers. If the soldiers land directly in forests or villages they could get hurt as well.

The group

To start a mission with the group already sitting in the helicopter, just enter following syntax in the init line of the group leader. The group leader was named, **Group1**.

{_x moveincargo Heli1} foreach units Group1

The script

The script can be freely named by the user, so it looks as follows.

```
_aunits = units Group1;  
_i = 0;  
_j = count _aunits;  
#Here  
(_aunits select _i) action ["EJECT", Heli1];  
unAssignVehicle (_aunits select _i);  
_i = _i + 1;  
~1  
?_j>_i : goto "Here"  
exit;
```

Now the user has the possibility to run the script with a waypoint, a trigger or even a script, by using the Syntax **this exec "Scripts\Heli.sqs"**. Another possibility without using your own scripts, is to use scripts which exist in the game already. To run this script just use the following syntax:

[GroupName,HeliName] exec "Para.sqs"

6.2 - The GPS-System

This system is quite useful if someone wants to allocate tactical signs to several units on the battlefield or display the location himself or of another unit.

To enable this, a script has to be used which needs to be defined in the **Init.sqs** or the **init line** of the player unit. That'll make the script run right when the mission begins. This script doesn't only place a marker on the position of the unit, it will check whether the respective unit is still alive or not. If the unit dies the marker gets deleted automatically.

Example 1:

```
"S1-Symbol" setMarkerText Name Soldier1;  
#START  
; Checking whether Sold 1 is still alive, if not script will go to the Label END  
? (!(alive Soldier1)) : goto "END";  
; Set Marker  
#MARKER  
"S1-Symbol" setMarkerPos getPos Soldier1;  
~1  
; Script jumps back to Label Start  
goto "START";  
#END  
deleteMarker "S1-Symbol";  
exit;
```

Example 2:

This one can be realized by using the **If-Then-Else-Syntax**. Actually the following syntax needs to be written in one line, but because this book is not wide enough it will be shown in several lines. The following script is not as long as the one above, so one can enter everything in one command line.

```
#Start  
If(alive Soldier1)Then{"S1-Symbol" setMarkerPos getpos Soldier1}  
Else{"S1-Symbol" setMarkerType "Empty";exit};  
goto "Start";
```

Explanation:

(If) **Soldier1** is still alive (Then) set **S1-Symbol** on **Soldier1** or (Else) delete **S1-Symbol** and exit script (Exit).

Note:

A further beloved GPS-variant, which will not be explained here, is to create the marker for the respective unit with a script and paste it onto that one. But he who will work carefully through the guide, will be able to create those kinds of scripts by himself.

6.3 - The action menu entry

The action menu is the one which is located in the right corner at the bottom of the screen. It's possible to add new entries or even delete them later if they are no longer needed. One can add another entry by using following syntax:

```
ID = Player addAction ["Entry", "Script.sqs", false, -1, false, false, "false"];
```

To delete an entry use this Syntax:

```
Player removeAction ID
```

If one wants to delete an entry, the respective name has to be used which has been defined with **ID**. If one wants to get an entry while a unit is sitting in a vehicle, the vehicle name has to be defined as well:

```
ID = Vehicle addAction ID = ["Entry", "Script.sqs", false, -1, false, false, "false"];
```

Trigger example:

A trigger needs be placed on the map first, then it has to be connected with the player character, so that only this unit can execute the trigger. The following settings are needed:

Activation: Repeatedly

Axis a/b: 5

On Activation: `ID = Player addAction ["Entry", "Script.sqs", false, -1, false, false, "false"];`

On Deactivation: `Player removeAction ID`

One can test this trigger now by running in and out of that area. The result should be that the entry will appear and disappear again when leaving the area.

The short Version of the action menu Syntax was earlier used in OFP only. The longer one will now be used in ArmA® although the smaller one still works well, but less efficient. So the Syntax is explained as follows. Note that nearly everything but priority and hot keys can be adjusted just by using true or false.

```
["Entry", "Script.sqs", Arguments, Priority, Display, Hiding, "Hotkeys"];
```

6.4 - The backpack

By using the action menu entries it's furthermore possible to simulate a backpack, trouser pockets or similar stuff. The following part will introduce the backpack feature which explains the possibilities by using those entries. The example is currently working fine for single player missions only.

The player character has to be placed on the map. The following syntax is needed in the initline:

```
RID = Player addAction ["Open Backpack", "backpack\backpack.sqs"]
```

Now one has to create a sub-folder called **Backpack** into the missions folder. Please make sure that all files in the missions folder, are written with small letters!

Now put all the scripts, which are needed for all the different actions, into the backpack folder. The following example displays four different scripts.

```
backpack.sqs  
save.sqs
```

```
firstaid.sqs  
close.sqs
```

Backpack.sqs

In the first step, the entry - **Open Backpack** - will be removed. The other entries will be added in the next step. The special thing here is that the first aid pack can only be used three more times. To make sure that the correct entry will appear every time a first aid kit was used, a variable will be set to **true** each time. The game remembers the last time the script was called. If the player has already used the first aid kit 3 times (see: ? **Bandage3:goto "close"**), the label **bandage3** is set set to **true**. That makes the script go on to the label **#Close** and ends the script by executing the script **close.sqs**.

```
; Entry will be removed
Player removeAction RID
playSound "OpenBackpack"

; Entries will be added
RIID = Player addAction ["- Save", "backpack\save.sqs"];

#Bandage
? verband3 : goto "Close";
? verband2 : goto "Bandage3";
? verband1 : goto "Bandage2";

#Bandage1
RIIID = Player addAction ["- First aid (3)", "backpack\firstaid.sqs"];
goto"Close";

#Bandage2
RIIID = Player addAction ["- First aid (2)", "backpack\firstaid.sqs"];
goto"Close";

#Bandage3
RIIID = Player addAction ["- First aid (1)", "backpack\firstaid.sqs"];

#Close
RIIID = Player addAction ["- Close Backpack", "backpack\close.sqs"];
exit;
```

A further exception for this script are the sounds which have been given for each respective action. See: **playSound "OpenBackpack"**. Those sounds are not quite important, but if one wants to use them anyway, they need to be defined in the **Description.ext**.

To make it more realistic, it's possible to add a second feature which would allocate a special animation to the player character. The next script, called **FirstAid.sqs** shall serve as example in this case. The character will kneel on the ground while healing himself.

Attention! Those four scripts are all encapsulated with each other! The old version of the Action menu was used here and can be exchanged by the new one which has been presented in **Chapter 6.3**.

Firstaid.sqs

One can see here that the script is checking if, and how often, the mission has been saved. If one would save the mission the first time, no variable was set to **true**. The script would go to the next label called **#Bandage1**, then it would set the respective variable **#Bandage1** on **true** and go to the next label called **#Heal** where all sub-entries would be deleted again. The player would receive the entry - **Open Backpack** - again in his action menu and is now able to heal someone else.

If the game would be saved for the second time, the script would go to the label **#Bandage2** because the label **#Bandage1** has already been set to **true**. If the game is saved for the third time, the script would jump from the second script line to the label called **#Bandage3**. Now **#Bandage3** has been set to **true** as well, and the script would go to exit and end the script if gets executed again (**?bandage3 : exit**).

```
? bandage3 : exit;
? bandage2 : goto "Bandage3";
? bandage1 : goto "Bandage2";

#Bandage1
bandage1=true;
goto "Heal";

#Bandage2
bandage2=true;
goto "Heal";

#Bandage3
bandage3=true;
goto "Heal";

#Heal
RID = Player addAction ["Open Backpack", "backpack\backpack.sqs"];
Player removeAction RIID;
Player removeAction RIIID;
Player removeAction RIIIID;
~0.2
Player playMove "AinvPknIMstpSlayWrflDnon_healed";
~1
Playsound "Sanipack";
~2
Player switchmove "AinvPknIMstpSlayWrflDnon_healed";
~5
Playsound "Pain";
~1
Player setDammage 0;
exit;
```

Save.sqs

The **Save.sqs** will save the current game. All entries will be deleted, that's because the script is running the **Close.sqs** from here. The current game status can be saved now.

```
exec " backpack\close.sqs";  
Savegame;  
exit;
```

Close.sqs

And the last file of course, which is needed if one is using the entry - **Close Backpack** -.

```
Playsound "Rucksackzu";  
Player RemoveAction RIID;  
Player RemoveAction RIIID;  
Player RemoveAction RIIIID;  
RID = Player addAction ["Open Backpack","backpack\rucksack.sqs"];  
exit;
```

And now a short description of the given entry-names and the backpack is ready to be filled up with its contents.

Name: RID

The name which is required to open the backpack.

```
RID = Player addAction ["Open Backpack", "backpack\backpack.sqs"]
```

Name: RIID

The name which is used for saving the game.

```
RIID = Player addAction ["- Save", "backpack\save.sqs"]
```

Name: RIIID

Was defined for all first aid entries, because only one is active.

```
RIIID = Player addAction ["- First aid (1)", "backpack\firstaid.sqs"]
```

```
RIIID = Player addAction ["- First aid (2)", "backpack\firstaid.sqs"]
```

```
RIIID = Player addAction ["- First aid (3)", "backpack\firstaid.sqs"]
```

Name: RIIIID

The name which has been defined for closing the backpack.

```
RIIIID = Player addAction ["- Close Backpack", "backpack\close.sqs"]
```

6.5 - Random positions

A mission which has almost the same storyline, might become boring quite soon and the player may put it away or delete it. But if someone has created a mission which is full of surprises, and enemy units are always attacking from different directions, there's much more tension in the mission and the chance to get played several times is much higher. It isn't a very fun way of playing Multiplayer missions if one will always have the information of where the enemy will come from and which location has to be destroyed. It also would be a better way of playing if the player character will get spawned at different places and the target locations will change as well each time playing the mission.

The editor offers the user the **radius of placement** for each single unit. That alone makes the mission more dynamic. But this option is actually meant for static objects only, which have been defined before. The following script will be defined either in the init line of a unit or in the init.sqs. It defines the starting positions when the mission begins.

Example: Dynamic start-points

The random command would get used here. The respective script can look like the example below:

```
_Start = random 3;
? _Start < 1 : goto "P1";
? _Start < 2 : goto "P2";
? _Start < 3 : goto "P3";

#P1
Player setPos [x,y,z];
exit;

#P2
Player setPos [x,y,z];
exit;

#P3
Player setPos getPos HP1;
exit;
```

A random value of 3 has been used here. When the script starts to run, a value will be created and will also be checked to see how much it is. Then the script is going to the next step.

The script would go to **#P1** if the value is smaller than **1**

The script would go to **#P2** if the value is smaller than **2**

The script would go to **#P3** if the value is smaller than **3**

One can define the positions behind the respective label now, for example **#P1**.

An XYZ-position has been defined for **#P1** and **#P2** while the player will be set onto an invisible **Heli-H** at **#P3** which is named **HP1**.

One wouldn't need to investigate the respective XYZ-Position for every single place, one only has to place several Heli-H onto the map and name them. At this point it's possible again to use the **Radius Of Placement** to enable a higher dynamic to the mission.

The user now has a dynamic spawn point and he doesn't know at what position he'll get spawned next time (**P1,P2,P3**). Because of the Heli-H radius definition, it's no longer possible to define the places where the targets will be spawned.

Example: Using Start points with coordinates

The following example explains the way how to define an XYZ-position, which also uses an additional variable radius of placement (**_radius = 500**).

The order of definitions has to be done as follows. The position **_pos** will be defined at first by using **[0,0,0]**, then the radius has to be defined next, the random value **3** will be defined for **_start**. The script is checking how much the value is and will jump to its respective position. The mark **_pos** will get one of the three XYZ-values allocated (**_pos=[X,Y,Z]**), which has to be chosen and defined before.

Now **_pos** has a fixed XYZ-position and will get an additional radius allocated around this position (**_radius=500**). Now this radius will be the area where the player character will be spawned each time when a mission begins.

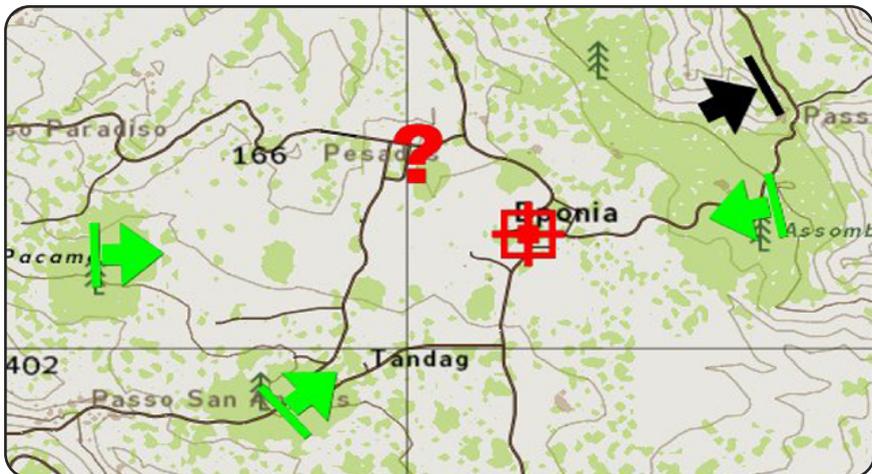
```
_pos = [0,0,0];
_radius = 500;
_start = random 3;

? _start < 1 : _pos = [X,Y,Z];
? _start < 2 : _pos = [X,Y,Z];
? _start < 3 : _pos = [X,Y,Z];

;This array actually has to be defined in one single line, but this is not possible here:
_pos = [(_pos select 0) + _radius/2 - random _radius, (_pos select 1) + _radius/2
- random _radius, _pos select 2];

Player setPos _pos;
exit;
```

One can imagine it on a map. The green markers are the possible starting positions for the unit or the player.



6.6 - The mapclick

The mapclick function offers a lot of new possibilities to the user, as one can see in the subsection "The Artillery" in this chapter. But the artillery script is only one of many possibilities using the mapclick. The following example is explaining the controlling of groups, calling an air-strike, controlling of supply movements and lots more things. This example is to be used for single player missions only.

The following example will explain how to control an AI-group named **Alpha1** on the map, by using the radio menu. The target position has to be defined by mapclick first, and a marker named **AMoveP** will appear right on this position. At the same time the leader of the unit will get the order to move to position **AMoveP**, and agrees with an added "**Roger**" sound which has to be defined in the Description.ext.

This group can now be marked and tracked on the map by using the GPS System, which is explained in **Chapter 6.2 - The GPS-System**. The tracked marker shall disappear again at position **AMoveP** and shall appear again only by using the next mapclick. To do this there will be defined a "waiting position" for the marker at position **[0,0]** which is defined in the end of the script.

Group Alpha 1:

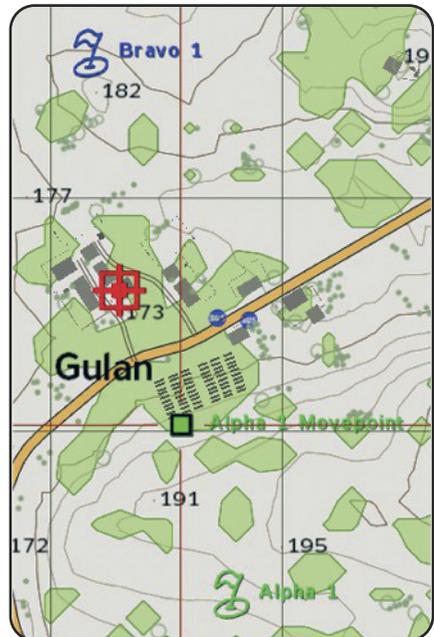
Name: Alpha1
Initline: Alpha1=group this
Size of the Group: Random

Radio trigger:

Activation: Radio Alpha
Repeatedly
Axis a/b: 0
Text: 0-0-1 Alpha 1
On Activation [] exec "scripts\alpha.sqs"

Marker:

Name: AmoveP
Text: Alpha 1 Movepoint
Symbol: Dot
Axis a/b: 1



Alpha.sqs

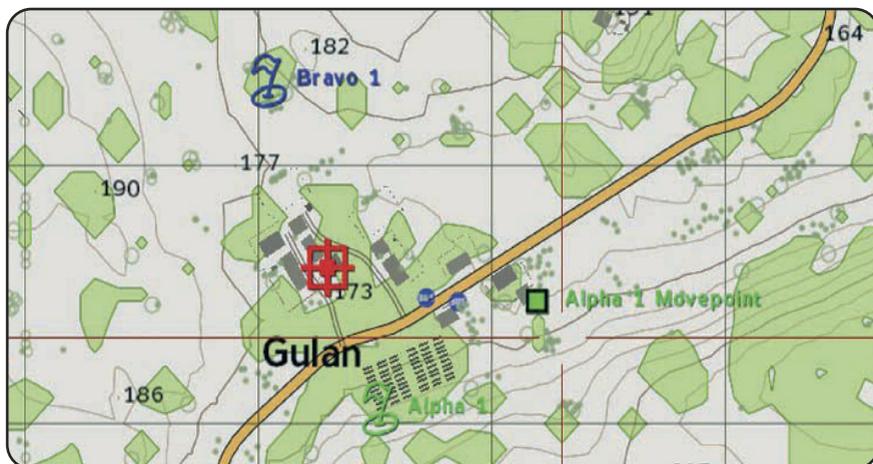
The following things will happen when this script gets executed: The variable **alphaclick** will be set on **true** and a text line will appear on the screen which asks the player to click on the map. Arma® notes the definition at **onMapSingleClick** and the script will break at **@!alphaclick** until the player has clicked on the map so that all commands after **OnMapSingleClick** which are defined between the **quotes** have been executed. In the next step, the variable **alphaclick** will get set back on **false** again.

The screen text will be deleted again and a sound file named "**Roger**" will be played. The marker **AMoveP** will beam to a non visible position after a break of **20** seconds until the next click will be done by the user.

```
alphaclick=true;
titleText ["Click on the map!";"plain down"];

;This array actually has to be defined in one single line, but this is not possible here:
onMapSingleClick "Leader Alpha1 move _pos;
                 alphaclick=false; ""AMoveP"" setMarkerPos _pos";

@!alphaclick;
onMapSingleClick "";
titleText [ "", "plain down"];
~1
playSound "Roger";
~20
"AMoveP" setMarkerPos [0,0];
exit;
```



6.7 - The artillery

The artillery is a very special feature in a mission. The player has the possibility to allocate a target to the artillery by clicking on the map or the AI can call the player for artillery support against a spotted enemy unit. The special thing about this artillery is that these guns can really exist on the map, but this feature is actually not important to the artillery script. If the guns are visible it's quite interesting to watch them lining up into the fire direction and firing.

This feature is unfortunately not possible without a little script work, which is luckily not as difficult to realize as it seems on the first view. First it's needed to adjust some things in the editor and create a subfolder in the missions folder which is called **Artillery** (make sure that it's written small).

Using with players:

The player should have the possibility to call the artillery by using the radio and allocate them a target by clicking on the map. While the firing process is running, a marker called **Firepoint** has to appear on the map and shall disappear right when the job is done.

Radio trigger:

Activation: Radio Alpha
Repeatedly

Axis a/b: 0

Text: 0-0-1 Artillery

On Activation: [] exec "Artillery\Setfire.sqs"



Marker:

Name: Firepoint

Symbol: Destroy

Axis a/b: 1



Invisible Heli H:

Name: ATarget

Position: Somewhere on the map



The guns:

This example has been defined with **6** guns with the gun type **M119** of the **BLUEFOR Side**. Those guns have to be set on the map and renamed to:

Names: W1, W2, W3, W4, W5, W6

Another type of gun wouldn't work with this example! Explanation will follow.

Setfire.sqs

The following script will be executed by using the radio trigger. The variable **setfire** will be set on **true** and a screen text (**titleText**) appears which asks the player to click on the map to define the target position. The definition of the mapclick will get started in the next line of the script and breaks again at the position **@!setfire**. The script is now waiting for the player to click on the map.

An invisible marker of the type Heli-H which is named **ATarget**, will be moved to the position (**_pos**) on the map. The variable **setfire** will be set back on **false** again which makes the condition **!setfire** (not setfire) complete. Now the script can go on.

The marker **Firepoint** will be moved onto the position of the **Heli-H (ATarget)** and the **onMapSingleClick** will be deactivated again. In the next step the script **ari.sqs** will be called. The screen text, which asks the player to click on the map, disappears again.

```
setfire=true;
titletext ["Click on the map to set your firedirection";plain down"];
onMapSingleClick "ATarget setPos _pos;setfire=false";

@!setfire;
"Firepoint" setMarkerPos getPos ATarget;
onMapSingleClick "";
[] exec "Artillerie\Ari.sqs";
titleText ["";plain down"];
~15
"Firepoint" setMarkerPos [0,0] ;
exit;
```

Ari.sqs

A radio sound will be played after this script is activated through the mapclick script. This sound has a length of 10 seconds and needs to be defined in the Description.ext first. The script fire.sqs will get executed after a delay of **10** seconds (**~10**) by the respective guns and the **ATarget**. This example shows only one round.

```
playsound "Firedirection";
~10
;FIRE
{[_x, ATarget] exec "Artillerie\Fire.sqs"} foreach [W1,W2,W3,W4,W5,W6]
exit;
```

If one wants to fire more than one round, the part fire including the respective delay needs to be copied and pasted between the last arty call and **exit**. The guns will fire again after a small reloading break.

Fire.sqs

The artillery guns are lining up and firing at the position which has been defined by clicking the map, after this script has been activated in the **Ari.sqs (Gun and ATarget [W1, ATarget])**.

The first object of the Array (**W1**) will be used with the local variable **_K** and the second one (**ATarget**) with the local variable **_Z**. The local variable **_X** receives the **X-position of ATarget (_Z)** and the local variable **_Y** is receiving the **Y_value from ATarget**.

By the order **_K doWatch [_X, _Y, 5000]** the script is telling to **W1 (_K)** that it has to watch to **ATarget** and in height of **5000** metres. After a delay of **5** seconds (**~5**) both options **W1** and **_K fire "M119"** are getting the order to fire. After a short while the grenades will impact in the predefined random area **_X+((random 80)-40)** and **_Y = _Y+((random 80)-40)**. The random area is variable of course.

With the order **_H say "Ari"**, a sound of an incoming shell will be played. This sound has to be defined in the Description.ext of course. But this sound will be audible only in a close area near to the impact point.

```
_K = _this select 0;
_Z = _this select 1;
_X = getPos _Z select 0;
_Y = getPos _Z select 1;
_K doWatch [_X, _Y, 5000];
_A = _K Ammo "M119";
~5
_K fire "M119";
@_A > _K Ammo "M119";
~2
_N = nearestObject [_K, "HeatM119"];
_X = _X + ((random 80) - 40);
_Y = _Y + ((random 80) - 40);
_H = "HeliHEmpty" createVehicle [_X, _Y];
~1
_H say "Ari";
~1
_N setPos [_X, _Y, 0];
"SH_125_HE" createVehicle [_X, _Y, 0];
deleteVehicle _H;
exit
```



This example works with the **M119** gun only, because it has been defined that way in the script. If one wants to use a different gun, the gun class (here: **M119**) and the respective ammunition (here: **HeatM119**) has to be defined in the script.

The different classes of the available guns are listed in **Chapter 3.2 – The weapon class names**.

Using with enemys:

The player or even friendly units can be attacked by enemy artillery fire in a predefined trigger area when they've been spotted by the enemy. The way to do this is similar to the one just explained. The only different is that the needed scripts are now to be saved in the subfolder called **enemy Artillery** and a **setfire.sqs** is not needed as well. A marker which defines the **fire zone** and a **Heli H** is not needed.

Variant 1:

One doesn't have to use eastern Artillery but needs to place a trigger on the map which contains the following settings:

Trigger:

Activation: WEST
Repeatedly
Detected by East

Axis a/b: 2000 (defines the area)

On Activation: [thisList] exec "EnemyArty/Ari.sqs"



The script called **ari.sqs**, which is located in the subfolder, needs some changes as explained on the next page. The **fire.sqs** remains set up on the **M119** gun!

All West units which are detected by east units will get attacked now by Artillery. The fact that the West guns are firing wouldn't get recognized by the player.

Variant 2:

If one wants to use special Eastern weapons, one has just to define it as shown below:

Trigger:

Activation: WEST
Repeatedly
Detected by East

Axis a/b: 2000 (defines the area!)

On Activation: [thisList] exec "EnemyArty\Ari.sqs"



The guns:

This examples has been defined with **4** guns with the eastern gun type **D30**. These need to be placed on the map and renamed as follows:

Names: E1, E2, E3, E4,E5,E6

Ari.sqs

The special feature in this script is the result of the fixed trigger syntax in the trigger which has defined as follows: **[thislist] exec " "**. That means that each West unit which has been spotted by East units will receive the local variable **_Target** in the script. The effect of this is that the target coordinates will be given automatically (West units). The respective gun and **_target** would execute the fire.sqs script which is located in the subfolder **enemy artillery**.

```
_Target = _this select 0;
;FIRE
{[_x,_Target] exec "EnemyArty\Fire.sqs"} foreach [E1,E2,E3,E4,E5,E6]
exit;
```

Fire.sqs

That script only needs to be set up on the gun type **D30**. Therefore the class names of the gun and the respective ammunition (**D30** and **HeatD30**) needs to be allocated as well. One also could use a tank instead of an artillery gun. **Chapter 3.16** explains the way how to get the weapon and ammunition types called.

```
_K = _this select 0;
_Z = _this select 1;
_X = getPos _Z select 0;
_Y = getPos _Z select 1;
_K doWatch [_X,_Y,5000];
_A = _K Ammo "D30";
~5
_K fire "D30";
@_A > _K Ammo "D30";
~3
_N = nearestObject [_K,"HeatD30"];
_X = _X+((random 80)-40);
_Y = _Y+((random 80)-40);
_H = "HeliHEmpty" createVehicle [_X,_Y];
_H say "Ari";
~1
_N setPos [_X,_Y,0];
"SH_125_HE" createVehicle [_X,_Y,0];
deleteVehicle _H;
exit;
```

6.8 - Deleting killed units and destroyed vehicles

It's quite important to save performance while playing multiplayer games, but that is also necessary for single player missions. The more units that are moving on the map or have to be displayed by the engine, the slower and more unreliable the mission becomes. This script deletes killed units in a predefined area and in a predefined time from the map. This executing syntax **[2] exec "bodydelete.sqs"** enables one to define the number of bodies which will not be deleted from the map. One only has to enter a number which is higher than **2**. If the script is executed now, only two bodies will keep lying on the ground.

This script is not referring to any side. Its more up to the way how it was adjusted. The following example explains a setting for East:

Trigger:

Activation: EAST
Once
Present



Axis a/b: 2000 (Define the area!)
On Activation: [2] exec "scripts\bodydelete.sqs"

If one wants all units, indifferent of which forces they belong to, to be deleted, one simply has to select **everyone** out of the **Activation** menu. If only West units have to be deleted, select **West** and so on. One can create a new subfolder in the missions folder called **scripts**. This script is located in the subfolder **scripts**.

The special thing with this script is that the killed units will sink in the ground before they will disappear completely. That function will work by using a special definition called:

(Gravedigger) action ["hidebody",_P]

To define it, one needs a unit which has to be named **Gravedigger**. It doesn't matter what side this unit belongs to. The **Gravedigger** makes it possible that the killed units will sink down into the ground before they get deleted. That unit needs to be placed far away from the battlefield. The best choice would be another island if available, to avoid getting killed by enemy forces. It's quite important that the name is exactly as used in the script, otherwise it won't work. Vehicles will not sink into the ground, they will get deleted directly from the map

Using with Soldiers:

This script has been set up on the type class **men** (**_T="men"**). It'll delete only vehicles which are defined as the respective type class. If one would define ground so all type classes which belong to this **ground** will be deleted. All vehicles which belong to the type class **ground** will be deleted from the map when they've been killed. Because aircraft's and ships are the only types of vehicles which are not moving on the **ground**, all destroyed or killed vehicles/units which belongs to this type will be deleted from the map.

The needed script called "bodydelete.sqs" looks like:

```
?!(local server):exit;
_W=_this select 0;
_L=[]+thislist;
_A=[];
_G=[];
_T="Man";

{ if (_T counttype [_x] == 1) then { _G=_G+[_x]} } foreach _L;

#Again
{ if (not alive _x) then { _A=_A+[_x]} } forEach _G;
_G=_G-_A;
? count _A > _W : _P=_A select 0; _A=_A-[_P];
(Gravedigger) action ["Hidebody",_P];
~10
deleteVehicle _P;
? count _A == _W and count _G == 0 :exit;
goto "Again"
```

6.9 - Suppressing gaming speed constantly

Sometimes it might be necessary to suppress the gaming speed. That's useful if one doesn't want the player to be able to speed up the game. The reason is that some missions have a special story line which needs to be seen and understood completely. Another reason is that some missions contain huge numbers of script's and speeding up the game could cause errors in the scripts.

One single command isn't enough to fix that problem in the editor, so a small script would be the best choice and even the easiest way:

```
; Suppressing gaming speed
#Check
? not alive Player : exit;
setAccTime 1.0;
~0.1
goto "Check"
```

This script should run directly at the start of the mission from the init.sqs or the player init line. A further possibility is to use the init line of an object. The script will always reset the time back to 1. Even if the player clicks the increase game speed button, it won't have any effect on the game speed.

6.10 - The bullet mode

This section will present a small but fine feature which isn't actually very realistic but shall demonstrate the possibilities of Arma®. The Bullet Mode enables the player to switch to slow motion while the game is running. That enables one to get some great screenshots. In this example the Bullet Mode has been added to the Action menu and was realized by using two scripts. That feature is usable in the single player mission only, and quite unnecessary in multiplayer missions.

The player needs an Action menu entry first:

```
ID=Player addAction ["Bullet Mode ON", "Bulleton.sqs"];
```

This entry is needed to run the bullet Mode . The respective script looks like this:

Bulleton.sqs

```
;Entry will be removed  
Player removeAction ID;  
  
;Entry will be added  
IID = Player addAction ["Bullet-Mode OFF ", "bulletoff.sqs"];  
  
;Slow motion will be set  
setAccTime 0.0900;  
exit;
```

The entry **Bullet-Mode ON** will be removed out of the Action menu and the new entry **Bullet-Mode OFF** will be added. The game speed will be displayed in slow-motion now until the player deactivates the Bullet Mode by clicking **Bullet-Mode OFF** again. The following script works as the one above only reversed.

Bulletoff.sqs

```
;Entry will be removed  
Player removeAction IID;  
  
;Entry will be added  
ID = Player addAction ["Bullet-Mode ON ", "bulleton.sqs"];  
  
;Slow motion will be revoked  
setAccTime 1.0;  
exit;
```

It would be a nice feature if one would add a music track which would get played if the mode has been activated and stopped again and if the mode gets deactivated again.

6.11 - Track down enemy units

If one wants to add a special feature to get enemy units spotted by friendly AI units to make them displayed with a blinking marker on the map (maybe to allocate Artillery fire or send other units to this position), that can be realized as shown in the following example. This script is executable in MP missions (of the game server only). To do this, the script needs the additional line **?(!(local server)):exit**.

The user has to place a marker and a trigger on the map. The trigger will define the respective area. For that function, adjust the options as follows:

Trigger:

Activation: EAST
Detected by WEST
Repetedly
Axis a/b: 5000
On Activation: thisList exec "scripts\signal.sqs"



Marker:

Name: Target1
Color: Red
Axis a/b: 1
Symbol: Destroy



The needed script looks like the following:

```
_Target = _this select 0;
signalcounter = 0;
"Target1" setMarkerPos getPos _Target;
"Target1" setMarkerType "Destroy";
#Start
? (signalcounter>=10) OR not alive _Target : goto "Ende";
signalcounter = signalcounter+1;
~0.8
"Target1" setMarkerColor "ColorRed";
~0.8
"Target1" setMarkerColor "ColorBlack";
goto "Start"
#Ende
~1
signalcounter = 0;
"Target1" setMarkerType "Empty";
"Target1" setMarkerColor "ColorBlack";
exit;
```

6.12 - The air strike

Airstrikes are always a tactical advantage. This is why an example will be shown here of how to create an Airstrike. But note that this version of the script needs to get adjusted to your own mission. That means that one needs to test the Airstrike in the respective mission area and adjusts the altitude of the aircraft or the time until the bomb will be dropped. The hits will become more and more accurate.

The Airstrike isn't usable in Multiplayer games unfortunately, because the bomb used by the Harrier would make the game crash. To add the Airstrike into the mission, one needs to place following objects on the map

Invisible Heli-H:

Empty/Objets: H (invisible)
Name: ASTarget



Radio trigger:

Activation: Radio Alpha
Text: 0-0-1 AIRSTRIKE
Axis a/b: 0
On Activation: [] exec "Airstrike.sqs"



Marker:

Name: Firedirection
Farbe: rot
Symbol: Destroy
Axis a/b: 1



The player has to click on the map when radio Alpha has been activated. ASTarget and the Marker Fire direction will be set right on the position where the player clicked on the map. The game will generate an Aircraft including Pilot, which is approaching the target. The pilot will get the bomb drop command when he reaches a predefined distance to the target and would drop the bomb. The aircraft is flying away out of the players view and would get deleted only a few seconds later.



Airstrike.sqs

The Logic and the marker are have to be placed somewhere on the border of the map to make them invisible to the player. Once the job is done the marker and the game logic will be moved back onto this position. You only need to add this script below and can get started.

```
setfire=true;
titleText ["Click on the map to set your firedirection";plain down"];
onMapSingleClick "ASTarget setPos _pos; setfire=false";

@!setfire;
"Firedirection" setmarkerpos getPos ASTarget;
playSound "Firedirection";
onMapSingleClick "";
titleText ["", "plain down"];

;=====DEFINE=====
_dropPosition = getpos ASTarget;
~0.5
_dropPosX = _dropPosition select 0;
_dropPosY = _dropPosition select 1;
_dropPosZ = _dropPosition select 2;
~0.1
_planespawnpos = [_dropPosX + 3000, _dropPosY, _dropPosZ + 1000];
_pilotspawnpos = [_dropPosX + 3000, _dropPosY, _dropPosZ + 1000];

;=====CREATE=====
_PlaneG = creatigroup WEST;
_plane = createVehicle ["AV8B",_planespawnpos,[], 0, "FLY"];
_plane setPos [(getPos _plane select 0),(getPos _plane select 1),900];
_pilot = "SoldierWPilot" createUnit [getMarkerPos "Firedirection", _PlaneG, "P1=this"];

_Plane setVelocity [100,0,0];
~0.4
P1 moveinDriver _plane;
P1 setDamage 0;
P1 action ["gear_up", vehicle P1];
_plane flyinHeight 100;
_plane setSpeedMode "full";

#CHECK
P1 doMove getPos ASTarget;
P1 doTarget ASTarget;
P1 doWatch ASTarget;
? (_plane distance ASTarget) < 1500 : goto "DROP"
goto "CHECK"
```

Continued on next page.

```

;=====FIRE=====
#DROP
_i = 0
_plane flyInHeight 100;
_plane setPos [(getPos _plane select 0),(getPos _plane select 1),100];
~13
#FIRE
_i=_i+1
_plane fire "BombLauncher";
~0.2
? _i <= 6 : goto "FIRE"

;=====FLY AWAY=====
ASTarget setPos [0,0,0];
"Firedirection" setMarkerPos [0,0];
_plane setSpeedMode "Full"
~4
_plane flyInHeight 300;
P1 doMove getPos ASTarget;

#Check2
_plane setDamage 0;
P1 setDamage 0;
? (_plane distance Player) > 2500 : goto "ENDE";
goto "Check2"

;=====DELETE=====
#ENDE;
deleteVehicle _plane;
deleteGroup _PlaneG
deleteVehicle P1;
exit

```

The accuracy will be better or even worse if the green marked labels **#Drop** and **#Check** will be adjusted, but this is also up to the landscape as in reality.

The pilot will get the order to fly away when he has dropped his bomb. This is needed to delete the aircraft out of the players view. The sound would stop immediately if the aircraft was deleted right when the bomb has been dropped, and that would take away any realism in the mission. A sound file called **playSound "Firedirection"** has been started at the beginning of the script. That sound file needs to be defined in the Description.ext.

6.13 - The air vehicle creator

This script enables one to generate aircraft in any kind and number. To do this one has to define a start point somewhere on the map. This start point can be an invisible Heli-H. Then one only needs to allocate a target to the newly generated unit, i.e. another unit or a further Heli-H.

This array enables the user to define a unit as Leader of a group or just place a unit named dummy somewhere on the map, ideally far away from the battlefield. This option is very much needed, because the script is not working without a Team leader. In the case of a Multiplayer mission don't name the leader as player. Make sure that the **Game Logic** named **Server** is present!

To run the script and the Array, use following Syntax:

```
[StartPos, "ParachuteC", "SoldierWB", TargetPos, 100, 0, 0.6,  
"Combat", Dummy, 10] exec "aircreate.sqs"
```

Actually this Syntax needs to be defined in one line, but that is not possible here.

- The start position:** That is an object on the map named Startpos. The best object would be an invisible Heli-H which should be placed far away enough from the Action Point. For fighter, it's recommended to place the object off-shore.
- The vehicle class:** The class name of the vehicle which has to be generated will be defined here. In the example above, it was already defined with "ParachuteC".
- The unit class:** The kind of unit which has to fly the aircraft will be defined here. In the normal way it'll be a pilot. But because paragliders are possible as well, other kinds of units can also be used.
- The leader:** Every unit which will be generated needs a leader. So it's possible to define any unit on the map. In the current case a Dummy has been used.
- The altitude:** The altitude has to be defined here. In our example the Jet will fly in a height of 100 meters.
- The gunner:** Some aircraft are not supported with a gunner position, so it's possible to disable this one by using the value **0** for "no gunner" and **1** for "gunner".
- The skill:** The skill of a unit will be defined here. What has been defined with 0.6, can also be defined randomly. To make it work just use a random value i.e. **random 1**.

The behavior: The behavior of a unit will be defined here. Combat has been given in the example above.

The target: In this part of the script the target will be defined. In this example the target, which is located somewhere on the map, will be named as **TargetPos**.

The number: Last but not least the number of ground vehicles which have to be generated is defined here.

Aircreate.sqs

```
?!(local server):exit;
_StartPos = _this select 0;
_AirTyp = _this select 1;
_PilotTyp = _this select 2;
_Target = _this select 3;
_Height = _this select 4;
_Gunner = _this select 5;
_skill = _this select 6;
_behaviour = _this select 7;
_Leader = _this select 8;
_count = _this select 9;
_counter = 0;

#Start
_counter = _counter + 1;
_Typ = createVehicle [_AirTyp,[(getPos _StartPos select 0)+ random 200,(getPos
_StartPos select 1)+ random 200,_Height + random 150], [], 0, "FLY"];
_Typ FlyInHeight _Height;
_Typ SetSpeedMode "full";
_Typ setDir getDir _StartPos;
_pilot = _PilotTyp createUnit [[[getPos _StartPos select 0),(getPos _StartPos select 1),
2000], _Leader,"Pilot1=this"];

Pilot1 moveInDriver _Typ;
Pilot1 setSkill _Skill;
Pilot1 doMove getPos _Target;
Pilot1 setBehaviour "_behaviour";

? _gunner == 0 : goto "Next"
_gunner = _PilotTyp createUnit [[[getPos _StartPos select 0),(getPos _StartPos select 1),
2000], _Leader,"Gunner1=this"];
Gunner1 moveInGunner _Typ;
Gunner1 setSkill _Skill;
Gunner1 setBehaviour "_behaviour";

#Next
? _counter >= _count : exit;
~0.5
goto "Start"
```

6.14 - The searchlight

The searchlight is a static object which is lighting when it becomes dark, but it's not moving around like a real search light. So I created this script. One has the possibility to define the very left and the very right border in an Array. That means that the unit which is standing at the search light will move the light until its left defined point in the array and then to its right defined point in the array. The searchlight is always moving from the first direction to the other one. Now it's up to the mission creator how far the light may go.

First one has to create the searchlight on the map and give it a name (i.e. **Guard1**). Then the array for the syntax needs to be defined.

To run the script the following syntax is needed:

```
[Name, left value, right value] exec "light.sqs"  
[Guard1,100,180] exec "light.sqs"
```

Those values will be given to the script **light.sqs** when it has been executed.

Light.sqs

```
_unit = _this select 0;  
_left = _this select 1;  
_right = _this select 2;  
_dir = (getDir _Unit);  
  
#Start  
? !(alive _unit) : exit;  
~0.5  
_dir = _dir+1;  
_unit setFormDir _dir;  
?(_dir > _right) : goto "Next"  
goto "Start"  
  
#Next  
? !(alive _unit) : exit;  
~0.5  
_dir = _dir -1;  
_unit setFormDir _dir;  
?(_dir < _left) : goto "Start";  
goto "Next"
```

Known Bug

It's possible that the searchlight won't start moving unless a unit is crossing its location, that's due to the system is moving down a lot in ArmA® and Objects which are far away from the player's position might not be visible. To avoid that the searchlight is not moving just let a patrolling unit crossing its location. Another possibility is to create and delete again a unit right in front of the searchlight as explained in **Chapter 5.45**.

6.15 - The time counter

If one wants to use a time counter, there are several ways possible to realize this. Two examples will be explained here. First the title text, and second the hint.

The trigger syntax will be defined as follows:

```
[60] exec "time.sqs"
```

The **60** means the time which is left until the counter has reached the **0** and will end the game.

Time.sqs

This script is for the title text. This countdown will appear as text at the bottom of the screen.

```
_time = _this select 0;

#Start
~1
_time = _time -1;
Titletext [format["Noch %1 Sekunden", _time], "plain down"];
if (_time >= 1) then {goto "Start"} else {};

Titletext [""; "plain down"];
exit;
```

And the hint variant: The numbers will be visible at the top left screen border.

```
_time = _this select 0;

#Start
~1
_time = _time -1;
Hint format["Noch %1 Sekunden", _time];
if (_time >= 1) then {goto "Start"} else {};

Titletext [""; "plain down"];
exit;
```

If one wants a value to count up, there's a small change needed in the script above. In the line **_time = _time -1** the **-** right before **1** has to be changed to **+** and the parameter **(_time >= 1)** has to be changed to **(_time <= _time)**. The syntax itself will get started by using this command:

```
[0] exec "time.sqs"
```

6.16 - The house patrol script

This script enables one to become units patrolling a building. But I have to say that the Hotel is the one where it currently makes the most sense to realize. The user has the possibility to adjust the script as he wants to do cause of the very huge Array. The script is also made to run perfectly within a Multiplayer game. The script will be executed by the now following, quite confusing Syntax. A explanation about how to adjust the Array will follow immediately but at first the executing Syntax with the Array.

[Name, "SAFE", Hotel, 161, 1, 160, 2, 10, 100, 200, 256, 0, 1] exec "housepos.sqs";

Description	Script	Explanation
Name	_man	Name of the unit
Behavior	_behaviour	Behaviour of the unit
Building	_house	Name of the building (see Chapter 5.61). Place a Logic right on the building and define Hotel=nearestBuilding this within the init line
Start position	_startpos	Startposition of the unit within the building
Accident	_accident	Random positions (1=on / 0=off) The random position for additional positions in the building will be activated here. If the value is 0 so the random position number and the numers of random positions will be ignored.
Accident position number	_accidentnr	Random position number, The respective value which should be used to generate the random position has to be defined right here. In example the value 160 , so a random position between 0 and 160 will be used now and assigned as waypoint.
Accident position count	_accidentcount	Number of random positions how much should such o software getting generated. 2 runs are defined above.
Pause	_rest	Break on position (Exampe above: 10) How long is the unit intended to wait, until she'll move to the next one.
Waypoint 1	_wp1	First Waypoint (example above: 100)
Waypoint 2	_wp2	Second waypoint (example above: 200)
End waypoint	_ende	Endwaypoint (example above: 250)
Static/Dynamic	_form	Start (Static=0/Dynamic=1) If the value is 0 (as shown above) so the unit will be placed right on the defined position of the building.
Repeat	_loop	Repeat script or end patrol.

It's possible to define values like break or Start-, waypoint 1, waypoint 2 or end waypoint with additional random commands to provide much more dynamic to the mission. To do this just write **random** right in front of the value within the Array.

This Array needs some concentration while adjusting because there are a bunch a numbers which are following one by one and one could become confused pretty fast. That's why one has to work correct.

HousePos.sqs

```
?!(Local Server): exit

_man = _this select 0;
_behaviour = _this select 1;
_house = _this select 2;
_startpos = _this select 3;
_accident = _this select 4;
_accidentnr = _this select 5;
_accidentcount = _this select 6;
_rest = _this select 7;
_wp1 = _this select 8;
_wp2 = _this select 9;
_ende = _this select 10;
_form = _this select 11;
_loop = _this select 12;

#LOOP
_counter = 0;
_man setbehaviour _behaviour;

;//Form of employment (Static=0/Dynamic=1)
? _form == 0 : { _x setpos (_house buildingPos _startpos)} foreach units _man;

;//First Waypoint (Enable=Buildingposition;Disable=0)
#WP1
? _wp1 == 0 : goto "WP2"
Leader _man move (_house buildingPos _wp1);

#WP1Dest
~1
? Leader _man distance (_house buildingPos _wp1) > 2 : goto "WP1Dest";
~_rest

;//Second Waypoint (Enable=Buildingposition;Disable=0)
#WP2
? _wp2 == 0 : goto "AccidentPos";
Leader _man move (_house buildingPos _wp2);
```

The script will go on the next page.

```

#WP2Dest
~1
? Leader _man distance (_house buildingPos _wp2) > 2 : goto "WP2Dest";
~_rest

;//Accidenpositions (Enable=1;Disable=0)
#AccidentPos
? _accident == 0 : goto "EndP";

#Accident;
_counter = _counter +1;
_apos = random _accidentnr;
~1
Leader _man move (_house buildingPos _apos);

#APosCheck
~1
? Leader _man distance (_house buildingPos _apos) > 2 : goto "APosCheck"
? _counter >= _accidentcount : goto "EndP"
~_rest
goto "Accident"

;//End Waypoint (Enable=Buildingposition;Disable=0)
#EndP
? _ende == 0 : goto "Ende"
~_rest
Leader _man move (_house buildingPos _ende);

#EndPDest
~1
? Leader _man distance (_house buildingPos _ende) > 2 : goto "EndPDest"

;// LOOP (Enable=1;Disable=0)
#Ende
? _loop == 1 : goto "Loop"
exit;

```

This script can be used now very flexibly. An example:

A user wants some units to patrol some waypoints around a building. Then the script will be activated on a special waypoint and the units that are patrolling the desired positions will go back to the building again once all predefined positions were touched. The parameter dynamic/static needs to be set on **1** (dynamic). The unit would get the order to move to waypoint **1**. Loop needs to get deactivated by setting the value on **0**!

It's basically recommended that a group shouldn't have more than 3 units while patrolling in a building. But 1-2 units would be much more realistic. It's possible to make more than one group patrol the Hotel. The player has to do a lot to escape unharmed when there're non patrolling units located there as well.

6.17 – The mine script

Because the mines in Arma® are not quite realistic, they need a little more power to do their job. The solution is a script again. To do this, two different scripts are needed. One for anti-human mines and one for anti-tank mines. The difference between both scripts are only the shell classes which are needed for the detonation. At least anti-tank mines are much more powerful then anti-human mines.

To do this, a mine needs to be set on the map at first. It also needs an additional trigger which is executing the script and causing the detonation. This one has to be set directly onto the mine so that a unit who's entering the trigger area will execute the script and cause the detonation.

Anti person mine

Trigger:

Activation: SIDE
Once

Axis: 1/1

on Activation: thislist exec "apmine.sqs"



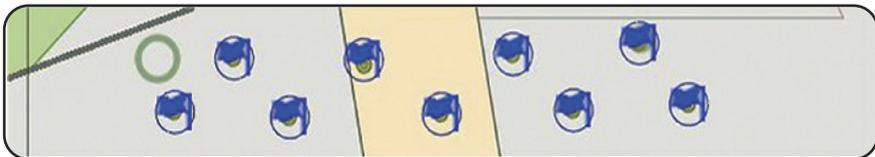
APmine.sqs

```
?(!(Local Server)): exit;  
  
_Soldier = _this select 0;  
_Bomb="SH_125_HE" createVehicle position _Soldier  
exit;
```

The user has a big advantage now. Once one mine has finally been finished, he can just copy the mine and past it in an endless number on the map. Each mine will work as expected.

Anti tank mine:

The same can be used for the anti-tank mine, but only the shell class needs to be changed. One can get the different shell classes from **Chapter 3.10**. The script itself has to be named as **TankMine.sqs** (or any other name). It's further recommended to size up the radius of the trigger.



6.18 – The vehicle transport script

This script enables the game to add an object or a vehicle right under an aircraft to take it to a predefined position. There're two ways possible to do this. The first, that 2 objects (helicopter(**Heli**) and vehicle(**Cargo**)) are to be predefined on the map to execute the VehTrsp script with their names. That makes the Cargo getting transported and debarked again at the predefined position, by the helicopter. This possibility is unfortunately not working in multiplayer missions at the time and can be realized in singleplayer missions only. To execute the script, use following syntax:

[Heli,Cargo, 70, Player, EndPos] exec "vehTrsp.sqs"

The array can be explained as follows:

[Transporter, Cargo, Altitude, DropPosition, EndPosition]

VehTrsp.sqs

```
?(Local Server): exit
_TranspVeh = _this select 0
_Cargo = _this select 1
_Height = _this select 2
_Place = _this select 3
_EndPos = _this select 4
_TranspVeh domove getpos _Place;
_TranspVeh flyinheight _Height;
;////////////////////////////////////TRANSPORT////////////////////////////////////
#Loop
_xPos = (getPos _TranspVeh) select 0
_yPos = (getPos _TranspVeh) select 1
_zPos = (getPos _TranspVeh) select 2
_Cargo setPos [_xPos, _yPos, _zPos -10];
_Cargo setDir (getDir _TranspVeh);
? _TranspVeh distance Player < 100 : goto "Next"
~.01
goto "Loop"
;////////////////////////////////////LANDING////////////////////////////////////
#Next
_TranspVeh flyinheight 13;
_TranspVeh setspeedMode "Limited";
_TranspVeh action ["AUTOHOVER", _TranspVeh];
#Loop2
_xPos = (getPos _TranspVeh) select 0
_yPos = (getPos _TranspVeh) select 1
_zPos = (getPos _TranspVeh) select 2
_Cargo setPos [_xPos, _yPos, _zPos -10];
_Cargo setDir (getDir _TranspVeh);
? (getPos _Cargo select 2) < 1.5 : goto "Loop3"
~.01
goto "Loop2"
```

```

##Loop3
_xPos = (getPos _TranspVeh) select 0
_yPos = (getPos _TranspVeh) select 1
_zPos = (getPos _TranspVeh) select 2
_Cargo setPos [_xPos, _yPos, _zPos -10];
_Cargo setDir (getDir _TranspVeh);
? speed _TranspVeh < 1 : goto "Ende";
~.01
goto "Loop3"
;////////////////////////////////////UNLOADING////////////////////////////////////
#Ende
_Cargo setpos [(getpos _Cargo select 0),(getpos _Cargo select 1),0]
_TranspVeh action ["CANCELACTION", _TranspVeh]
_TranspVeh flyinheight _Height
_TranspVeh domove getpos _EndPos
_TranspVeh setSpeedMode "Normal"
exit

```

Spawn-Variant

The second way is that both vehicles are getting created on the map right onto a start point. The helicopter is getting the cargo to its predefined position, dropping the cargo and flying away to an endpoint where it gets deleted. To create that, all viewable within the editor, all of it has been realized with 3 invisible Heli-H markers, called:

StartPos	Target	EndPos
----------	--------	--------

These can be moved on the map now. The special thing is that the helicopter will be aligned right in to the direction of view of the **StartPos Object**. So it's possible to turn the flight direction of the helicopter by turning the **StartPos object**.

It's also possible to define the player or any other similar thing as debark position. To do this just define the parameter **Player** or **Name** of the unit instead of **Target** where the cargo has to be dropped to.

In this case a **Dummy** called Leader has been created on the map as leader of the group. That's why the helicopter may fly and land better then if no leader would be used.

The following Syntax is executing the script:

```

["UH60MG", "HMMWVMK", "SoldierWPilot", 70,
StartPos, Target, EndPos, Dummy] exec "vehtrspcreate.sqs"

```

The Array can be explained as follows:

```

["HeliTyp", "CargoTyp", "Pilottyp", Altitude,
StartPosition, DropPosition, EndPosition, Leader]

```

VehTrspCreate.sqs

```
?(Local Server): exit
_AirVeh = _this select 0
_Vehicle = _this select 1
_Pilottyp = _this select 2
_Height = _this select 3
_StartPos = _this select 4
_Place = _this select 5
_EndPos = _this select 6
_Leader = _this select 7

;//////////////////////////////////////CREATE//////////////////////////////////////
_TranspVeh = createVehicle [_AirVeh,[(getpos _StartPos select 0),(getpos _StartPos
select 1),_Height], [], 0, "FLY"];
_TranspVeh setdir getdir _StartPos
_Cargo = _Vehicle createVehicle getpos _TranspVeh
_pilot = _Pilottyp createUnit [[(getpos _StartPos select 0),(getpos _StartPos select
1),2000],
_Leader,"Pilot1=this"]

Pilot1 moveinDriver _TranspVeh
_TranspVeh flyinheight _Height

;//////////////////////////////////////TRANSPORT//////////////////////////////////////
#Loop
Pilot1 domove getpos _Place
_xPos = (getPos _TranspVeh) select 0
_yPos = (getPos _TranspVeh) select 1
_zPos = (getPos _TranspVeh) select 2

_Cargo setPos [_xPos, _yPos, _zPos -10];
_Cargo setDir (getDir _TranspVeh);
? _TranspVeh distance _Place < 90 + random 50 : goto "Next"
~.01
goto "Loop"

;//////////////////////////////////////LANDING//////////////////////////////////////
#Next
_TranspVeh flyinheight 13;
_TranspVeh setSpeedMode "Limited";
_TranspVeh action ["AUTOHOVER", _TranspVeh];

#Loop2
_xPos = (getPos _TranspVeh) select 0
_yPos = (getPos _TranspVeh) select 1
_zPos = (getPos _TranspVeh) select 2

_Cargo setPos [_xPos, _yPos, _zPos -10];
_Cargo setDir (getDir _TranspVeh);
```

```

? (getPos _Cargo select 2) < 1.5 : goto "Loop3"
~.01
goto "Loop2"

#Loop3
_xPos = (getPos _TranspVeh) select 0
_yPos = (getPos _TranspVeh) select 1
_zPos = (getPos _TranspVeh) select 2

_Cargo setPos [_xPos, _yPos, _zPos -10];
_Cargo setDir (getDir _TranspVeh);
? speed _TranspVeh < 0.7 : goto "Ende";
~.01
goto "Loop3"

;/////////////////////////////////UNLOADING/////////////////////////////////
#Ende
_Cargo setpos [(getpos _Cargo select 0),(getpos _Cargo select 1),0]
_TranspVeh action ["CANCELACTION", _TranspVeh]
_TranspVeh flyinheight _Height
_TranspVeh domove getpos _EndPos
_TranspVeh setSpeedMode "Full"

;/////////////////////////////////DELETE/////////////////////////////////
@ _TranspVeh distance _EndPos < 100
deleteVehicle _TranspVeh
deleteVehicle Pilot1
exit

```



6.19 – The seagull script

A swarm of Seagulls adds a lot of atmosphere to a mission. There's actually already a lot of stuff in the air, but when it's possible to create a swarm flying around a predefined route, it's much more impressive. All of that has to be realized again by using a script. For that example 2 objects are needed, ideally invisible Heli H's, which have to be placed on the map. They also need a name and the target and start parameters needs to be defined also. Then they can be defined within the execute array of the Script.

[Start, Destination, NumberOfBirds] exec "scripts\bird.sqs"

Bird.sqs

```
_spos = _this select 0
_zpos = _this select 1
_count = _this select 2
_i = 0

#Check
_x = (random 30) + 10
_y = (random 30) + 10
_z = (random 30) + 10

? _i >= _count : exit
_i = _i + 1

_bird="seagull" camcreate [(getpos _spos select 0) - (sin getdir _spos * _x),
                          (getpos _spos select 1) - (cos getdir _spos * _y), _z]

[_bird, _spos, _zpos] exec "birdpos.sqs"
goto "check"
```

Once the script has been executed, the birds will be created within an random area around the start position. (defined by: x,y;z). This area can be freely defined at any time by the user again. Each bird is executing the second script and gets a random destination allocated where it flies to.



BirdPos.sqs

```
_bird = _this select 0
_spos = _this select 1
_zpos = _this select 2
_x = (random 10) + 10
_y = (random 10) + 10
_z = (random 10) + 10

#Loop
_bird camsetpos [(getpos _zpos select 0) - (sin getdir _zpos * _x),
                 (getpos _zpos select 1) - (cos getdir _zpos * _y), _z]
? _bird distance _zpos <= 30 : goto "Loop2"
~1
goto "Loop"

#Loop2
_bird camsetpos [(getpos _spos select 0) - (sin getdir _spos * _x),
                 (getpos _spos select 1) - (cos getdir _spos * _y), _z]
? _bird distance _spos <= 30 : goto "Loop"
~1
goto "Loop2"
```

This example can be expanded with further features and positions of course. It's also possible to use bees or similar objects instead of seagulls. To get the respective class names, please go to **Chapter 3.9 - The units classes**.

Now a few syntax examples will follow:

- | | |
|--|------------------------------------|
| deletevehicle _bird | - The birds will be deleted |
| _bird camSetPos position Player | - The bird flies to the player |
| _bird camsetpos [x,y,z] | - The bird flies to X,Y,Z Position |
| _bird camCommand "Landed" | - The bird is landing |



6.20 - The insect script

The insect script is similar to the seagull script. But this one has been created a little different. In this case flies shall fly right over a dead body. To realize this just paste the following syntax into the init line of the respective unit.

[this,60] exec "scripts\insect.sqs"

It's also possible to use the name of the unit instead of **this**. The number **60** represents the number of Flies which are used to orbit the body It's further possible to use special sounds for the flies, which will be heard close around the body.

Insect.sqs

```
?!(local server) : exit
_deadman = _this select 0
_count = _this select 1
_i = 0
@not alive _deadman
#Check
_x = random 1
_y = random 1
_z = random 3
? _i >= _count : exit
_i = _i + 1
_insect="HOUSEFLY" camcreate [(getpos _deadman select 0) + _x,
                             (getpos _deadman select 1) + _y,_z]
[_deadman,_insect] exec "insectpos.sqs"
goto "check"
```

Insectpos.sqs

```
_deadman = _this select 0
_insect = _this select 1
#Loop
_insect camsetpos position _deadman
? _insect distance _deadman <= 0.1 : goto "Loop2"
~2
goto "Loop"
#Loop2
_x = (random 3) + 1
_y = (random 3) + 1
_z = (random 3) + 2
_insect camsetpos [(getpos _deadman select 0) + (sin getdir _deadman * _x),
                  (getpos _deadman select 1) + (cos getdir _deadman * _y), _z]
? _insect distance _deadman >= _x : goto "Loop"
~2
goto "Loop2"
```

6.21 – The saboteur

This section will show how to use a default AI unit as saboteur. The unit will move to the destination point, dropping the bombs and ignite them from a safe position, resp. reached its next waypoint.

To explain how it works, a quite simple example. At first the unit needs to own the resp. bombs, so one can allocate them to the unit by using the syntax

this addMagazine "Pipebomb";

which has to be entered into the init line of the unit. In this example we're using an East-Saboteur, which is already equipped with respective Ammunition.

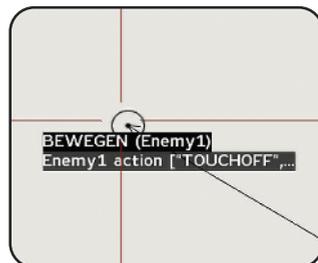
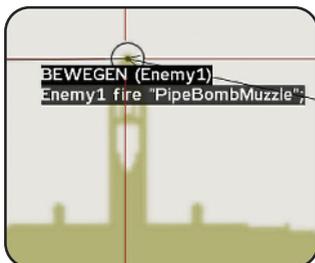
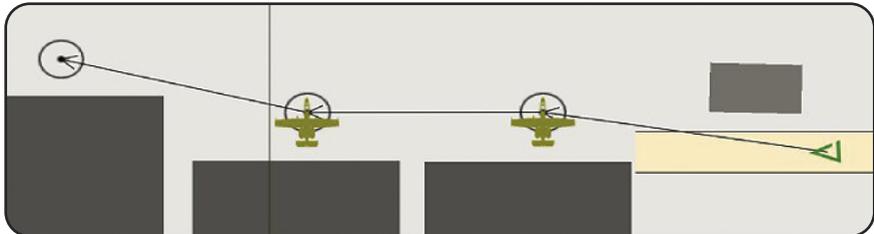
Now we place the unit called **Enemy1** onto the map and allocate it the needed waypoints. Now it's important to know that each waypoint needs to get the following syntax into its OnActivation line:

Enemy1 fire "pipeBombMuzzle"

The needed syntax to execute the detonation has to be entered into the last waypoint. When the unit arrives at that waypoint, the pipebombs shall get ignited.

Enemy1 action ["TOUCHOFF", Enemy1];

Example pictures:



It's possible now of course to pimp that up, maybe with further waypoints, actions and so on. The way how the waypoints are to be adjusted is always up to the user himself. And don't forget the behavior settings, because these are not quite unneeded. Be sure to adjust the time settings as well (Min, Mid, Max) to add a nice dynamic to the mission.

6.22 - The spotter

This example explains how to realize an intelligence unit. One has the possibility to switch between the units by using the radio menu, to get a small overlook over the current position. It's unimportant whether the unit is a soldier, a tank or an aircraft.

The basic commands enable one to switch to the unit by using different views. The following views are available:

"INTERNAL"	- Regular 1st Person view
"EXTERNAL"	- 3rd Person view (onto the units back)
"GUNNER"	- Optical view (visor)
"GROUP"	- View to the Group (leader only)

To switch to the units with the desired view, use the syntax below:

Player switchCamera "INTERNAL"

The following syntax example explains the situation where the player has the possibility to switch into the perspective of the reconnaissance unit within a predefined time and a predefined view. Once the time has ran out, the player will switch back into its own perspective again. The start syntax is to be freely defined again.

Every kind of unit can serve as reconnaissance unit. The execution syntax is defined as follows, where the **player** itself, **the spotter**, **the time** and the **perspective** need to be defined.

```
[Player,Spotter,10,"EXTERNAL"] exec "spotterSwitch.sqs"
```

SpotterSwitch.sqs

```
_leader = _this select 0
_spotter = _this select 1
_timeo = _this select 2
_mode = _this select 3
_i = 0
disableUserInput true
_spotter switchCamera _mode
#Loop
_i = _i + 1
? _i >= _timeo : goto "Ende"
~1
goto "Loop"
#Ende
titleCut [" ", "Black Out"]; titleFadeOut 1
~0.5
_leader switchCamera "INTERNAL"
disableUserInput false
titlecut [" ", "Black In"]; titleFadeOut 1
exit
```

6.23 – Unit is capitulating itself

The next example will explain how one can get units to capitulate themselves so long as a special condition has already been executed. A condition could be that the unit would have less ammunition than what is defined, or the unit is injured more than what was predefined or that less allied units are still existing in the desired area.

It's possible now to concentrate the conditions to avoid each unit capitulating although there are still enough units left in the respective area. There needs to be more than one condition executed to make the units capitulate. For example:

```
?!(canStand_enemy) AND (_ammo < _AmmoStat) : goto "PutDown"
```

The array above defines the condition, that the unit "is not able to keep standing" and additionally that the "status of the remaining ammunition is smaller than what was predefined". If both conditions become true, the script will jump to the label **PutDown** and the unit will capitulate himself. There were several mixed condition lines added to the script within the label **Check**. If one of those become true, the unit will capitulate as well.

Now a special feature. In real life, it happens sometimes that units are capitulating but nevertheless grabbing the weapon in an unexpected moment, or they have an additional weapon hidden somewhere in their equipment. Other units may capitulate totally without any unexpected issues. This script enables one to define these situations randomly. One can define whether the unit will really capitulate or try to flee someone. One can further define whether either the weapons are all deleted or one weapon is still existing and can be used by the respective unit.

If a unit is capitulating himself, he will drop all his primary and secondary weapons and cross his hands behind the neck. If the user has defined that the unit should get fixed right on their position, nothing more will happen. But if it has been defined that the unit shall try to flee, it might be that he'll fire back if one weapon hasn't been deleted. It might also be that this unit is throwing a hand grenade. And it can further be that the unit is rearming by a dead unit. Because of SetCaptive, the AI unit will not fire on capitulated units, but they would if the unit is trying to flee or trying to grab a weapon.

If fixed values were used in the script, one can be sure that the units will always capitulate, if desired conditions became true. But if there are random values used within executing the array, the player will always remain excited about what will happen next.

The Trigger Array

The trigger array will be explained as follows:

```
[Name,EscapeValue,Ammunition,Injured>DeleteWeapons,Fix,Friends,Area]
```

Here an example of fixed values:

```
[this,0.5,1,0.5,0,0,3,150]
```

Here an example with random values:

```
[this,random 0.7,2,random 1,random 1.5,random 2,6+(random3),200]
```

These variable, regardless of whether fixed or variable, will be given to the script. But this still has to be defined right behind the array. To execute the script use the following syntax:

```
[this,0.5,1,0.5,0,0,3,200] exec "capitulation.sqs";
```

Array definitions

- Name** - Name of the unit. **This** parameter can be used within the init line also.
- Escape Value**- Escape value of the unit which represents a condition.
- Ammo** - Minimum value of ammunition which is needed to make the units capitulate.
- Injured** - Minimum grade of injury which is needed to make the units capitulate.
- Weapons deleted** - Delete all weapons or not. Value 1 will delete all weapons while the values **0.1** till **0.9** will keep the weapons. Unknown what happens by using random **1.5**. Within ≥ 1 all weapons are removed so the random values shouldn't be larger than **1.5**!
- Fix** - The unit will definitely keep standing once she has been capitulated. Values ≤ 1 are fixing the unit, Values ≤ 2 enables the unit to escape if possible. The example shows random 2 = unknown.
- Friends** - How many allies are still in the respective area? The condition is caused if the value has fallen below the predefined number. The values used in the example are **6+(random3)**, so **6** plus **random value**. There were East units defined in the script, if one wants to get other units, these parts (unit classes) need to be edited by the user.
- Area** - Area which shall be checked. If there are no units left in the defined area, the trigger will execute.

Single Unit

If the user only wants one unit eventually to become capitulated, the following syntax needs to be pasted into the init line of the unit. Now the desired values only need to be defined.

```
[this,0.5,1,0.5,0,0,3,200] exec "capitulation.sqs";
```

Units within an area

All units will execute the script by using the following executing syntax. If the random values were used, each units will behave different to the others.

Trigger:

- Activation:** EAST
Once
- Axis a/b:** optional
- on Activation:** {[[_x,random 0.7,2,random 1,random 1.5,random 2,6,200]
exec "capitulation.sqs"} forEach thislist

This example can be expanded indefinitely and even the conditions can be freely defined.
Very detailed... but all must have a limit..

```
?(!(local server)):exit
_Enemy = _this select 0
_Fleeing = _this select 1
_AmmoStat = _this select 2
_Injured = _this select 3
_Removeall = _this select 4
_Standfix = _this select 5
_friendly = _this select 6
_Area = _this select 7
_Enemy allowFleeing _Fleeing

#Check
? not alive _enemy : exit
_ammo = count magazines _enemy
_dammage = getdammage _enemy
;The following 2 lines are representing one single line! Please add/remove class names!
_friends = count nearestObjects [_enemy,
["SoldierEAA","SoldierEAT","SoldierECrew","SoldierEMiner","SoldierEG","SoldierEMG",
"SoldierEMedic","SoldierESniper","SquadLeaderE","T72","BMP2","UAZMG","ZSU"], _area]
;The following 2 lines are representing one single line!
? (_friends < _friendly) AND (_ammo < _AmmoStat) AND (_dammage > _Injured) :
goto "PutDown"
;The following 2 lines are representing one single line!
? (_ammo < _AmmoStat) AND (_dammage > _Injured) AND (_Fleeing > 0.6) :
goto "PutDown"

~5
goto "Check"

#PutDown
_enemy action ["DROPWEAPON", _enemy, primaryWeapon _enemy]
~1

playSound "Dont-shoot"
_enemy action ["DROPWEAPON", _enemy, secondaryWeapon _enemy]
~2
? _Removeall >= 1 : removeAllWeapons _enemy
_enemy playMove "AmovPercMstpSsurWnonDnon"
~1
? _standfix <= 1 : goto "Standfix"
? _standfix <= 2 : goto "Fleeing"
exit

#Standfix
_enemy disableAI "ANIM"
_enemy setCaptive true

#Fleeing
_enemy disableAI "ANIM"
_enemy setcaptive true
~30 + (random 60)
_enemy enableAI "ANIM"
_enemy allowFleeing _Fleeing
_enemy setCaptive false
exit
```

6.24 - Teleport

This example is actually not meant for regular use as a mission feature but as quite good help while editing a mission. It enables the user to beam to any desired position on the map by using map-click only to check whether all created things are running correctly. So the user doesn't need to keep restarting the mission or running over the map. That saves a lot of time and nerves.

This example is combined with a radio trigger. The following syntax is needed to be written within the init line of a unit or an object. But one can also create an **Init.sqs** within the missions folder to make the script run once the mission has been started.

[] exec "teleport.sqs"

The variable **Teleport** will be set on **true** later by using the radio trigger and the script starts to work. If the player clicks now on the map, he'll beam to the position where he clicked on the map and the script will end. The marker **PosM** is beaming now to the position where the user has clicked, so the position will be visible on the map where the user's character is currently located.

Radio Trigger:

Activation: Radio Juliet
Repeatdetly
Text: 0-0-0 Juliet
onActivation: teleport=true



Marker:

Name: PosM
Color: rot
Symbol: Destroy
Axis a/b: 1



Teleport.sqs

```
#start
@teleport
titleText ["Click on the map to teleport yourself","plain down"]
onMapSingleClick "player setpos _pos; teleport=false; ""PosM"" setMarkerPos _pos"

@!teleport
titleText [" ", "plain down"]
~10
"PosM" setMarkerPos [0,0]
goto "start"
```

To get the same result without a script just paste the syntax below right into the init line of the player character, but this function will permanently be active. The advantage is that this version is pretty easy to realize, but the disadvantage is that the player will always move over the map so far scripts are also used which are including the map-click.

onMapSingleClick "Player setPos _pos";

6.25 - The persecution script

The following example will explain a possible variant of the persecution script in much more detail. So the persecutor which was named **Hunter** in this example will get the correct behaviour assigned before he starts to get to the **player** character. The script will be active as long as the player or the persecuting unit or group is no longer alive. To run the script use following syntax:

```
[Hunter, Player] exec "scripts\hunter.sqs";
```

Hunter.sqs

```
_hunter = _this select 0
_target = _this select 1

_hunter setBehaviour "AWARE"
_hunter setCombatMode "RED"
_hunter setSpeedMode "Full"
_hunter doTarget _target

#Loop
?_target distance _hunter >= 10 : {_x doMove getPos _target} foreach units _hunter
~4
? (count units _hunter) <= 1 : exit
?!(alive _target) : exit
goto "Loop"
```

There're possibilities of course to realize this by using waypoints, but this is only an example of one way to do it. The collaboration between the different commands should be shown here like the implementation of your own ideas and combining them with each other as well.



Chapter 7

- Multiplayer -

This chapter explains the basics of multiplayer missions. After working with this chapter, you will be able to create and edit your own multiplayer missions.

7.1	The multiplayer mission	224
7.2	The respawn positions	224
7.3	Flexible respawn pos	225
7.4	The MP-Description.ext	226
7.5	The different ways to respawn	227
7.6	The deathmatch	227
7.7	Defining the multiplayer area	228
7.8	Time and rating	229
7.9	Assigning and displaying Scores	231
7.10	Time display	232
7.11	The class header	233
7.12	The respawn dialog	233
7.13	Stringtable MP basic values	234
7.14	The vehicle respawn	235
7.15	Mr-Murray's vehicle respawn	236
7.16	Flag basic informations	238
7.17	Capture the flag	240
7.18	The public variable	246
7.19	Preface information for MP missions	247
7.20	The controlling commands	249
7.21	The armament within multiplayer	250
7.22	Text messages for a specific player	251
7.23	Join in progress (JIP)	252

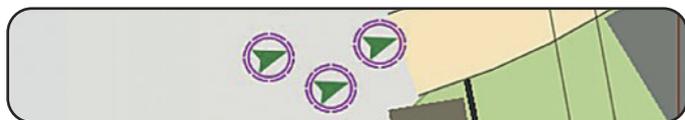


7.1 - The multiplayer mission

The information about creating multiplayer missions could fill a whole separate book. This chapter will explain the most important parts of multiplayer mission creation and will allow you to create your own simple multiplayer mission. The information given here can be used in more complex missions later on. It's basically recommended to place a game logic on each Multiplayer map which is called **Server!**

Units

To make sure that units which are placed on the map are playable later in the mission, one has to select the option "playable" in the respective drop down menu of the unit menu in the editor. If one wants specific units to only exist if they are controlled by a player, the following command has to be added to the Description.ext: `disableAI=1`. Playable units which are not in use will be deleted and they will be not replaced by AI units and are not visible.



Number of human players?

One can get this information as a value which can be further used as a condition!

hint format ["West: %1\nEast: %2", playersNumber WEST, playersNumber EAST];

or now the condition:

? playersNumber > 4 : hint format ["WEST: %1", playersNumber WEST]

7.2 - The respawn positions

Markers are the best way to define respawn points. Respawn markers have to be renamed to match the side using the respawn point:

West:	Respawn_west	Resistance:	Respawn_guerrilla
East:	Respawn_east	Civilian:	Respawn_civilian

If one wants to add more respawn points, a respective number has to be added behind the name. I.e.:

Respawn_west_1, Respawn_west_2,...



One also has the possibility to use other objects instead of markers. It's possible to use Objects and Game Logics. The disadvantage of doing this is that the unit will be respawned exactly in the place of the object or game logic, while the radius of the marker is adjustable which means that the units can be respawned at any point within the marker radius.

7.3- Flexible respawn points

If one wants to create a mission which contains flexible respawn points, the user has several possibilities. To explain the "how to", an execution of a mission target shall serve as example.

Every mission begins the same way. The units are placed at some point on the map. Until the first target has been destroyed, the respawn point will not move. If Target1 has been destroyed, the respawn point moves to the position of Target1 and enables the player to be respawned at the new position. If the mission contains several mission targets, the respawn point will jump from target to target after the respective objective has been destroyed or has been executed.

The player has several respawn possibilities. An example:

Respawn Marker

Name: Respawn_West
Axis a/b: 50/50



Trigger

Type: Once
Name: AreaOne
Axis a/b: 50
Activation: OPFOR
Not present



OnActivation: "Respawn_West" setMarkerPos getPos AreaOne
hint "Congratulations - Target one accomplished!"

In that example, a new trigger called "**AreaOne**" will be placed, which has to check whether the zone is free of enemy units. If it's true, the marker called **Respawn_West** will be moved right onto the position of the trigger called "**AreaOne**" and a screen text appears which says: "**Congratulations - Target one accomplished!**" That shall serve as a small example only. It's possible to have the marker moved to another position within the target zone.



7.4 - The MP-Description.ext

One can define the basic settings in the Description.ext. For example, the type and countdown to the respawn. The following components are needed:

respawn=3;	- The kind of respawn
respawnDelay=6;	- The countdown until the unit will respawn
respawnVehicle=3;	- The kind of the vehicle respawn
respawnVehicleDelay=10;	- The countdown until the vehicle will respawn
disabledAI=1;	- Units which have been defined as playable will not be present as AI units in the game
Alkills=1;	- The score of the AI units will be counted as well.

The following example shows the most important parts which need to be defined in the Description.ext.

Description.ext

```
respawn=3;
respawnDelay=6;
respawnVehicle=3;
respawnVehicleDelay=10;

disabledAI=0;
Alkills=1;
respawnDialog = false;

class Header
{
    gameType = CTF;
    minPlayers = 2;
    maxPlayers = 10;
};

titleParam1 = "Time limit:";
valuesParam1[] = {10000, 300, 600, 900, 1200, 1500, 1800, 2100, 3600, 7200};
defValueParam1 = 1800;
textsParam1[] = {"Unlimited", "5 min", "10 min", "15 min", "20 min", "25 min",
                "30 min", "35 min", "60 min", "120 min", };
titleParam2 = "Score to win:";
valuesParam2[] = {10000, 5, 7, 10, 15, 20, 25, 30};
defValueParam2 = 5;
textsParam2[] = {"Unlimited", 5, 7, 10, 15, 20, 25, 30};
```

7.5 - The different ways to respawn

There're several possibilities for respawning once the player has been killed. Those possibilities will be defined in the Description.ext when the mission is created. That is meant for vehicles and for units as well. But it doesn't make much sense to respawn a destroyed vehicle back into the game as seagull. The vehicle respawn is more accurately explained in **Chapter 7.10 - The Vehicle Respawn**.

The ways to respawn:

- | | |
|--------------------------------|--|
| 0 or " None " | - No Respawn |
| 1 or " Bird " | - Respawn as Seagull |
| 2 or " Instant " | - Respawn right on the position where one has been killed |
| 3 or " Base " | - Marker respawn (Respawn_west,...) |
| 4 or " Group " | - Group based respawn (If no more friendly AI units are left, then respawn as seagull) |
| 5 or " Side " | - Side based respawn (If no more friendly AI units are left, then respawn as seagull) |

Vehicles can only spawn again with the values **0**, **2** and **3**.

7.6 - The deathmatch

If one wants to create a deathmatch mission which shall be playable by only one or more players or even against the AI, it's necessary to turn units on the same side against each other.

Variant 1

The "Setfriend-order" could be a possibility:

East setFriend [East,0.1]

If there are several sides, all of them need to become enemies of each other:

Variant 2

This syntax makes units enemies of each other, indifferent of which side they belong to.

this addRating ((- rating this) - 100000)

Units which haven't got this entry will not be shot by their own side.

Note!

While creating deathmatch mission with AI units, it's necessary to allocate two waypoints as a minimum, which have to cover a wide enough range to make sure that those units will move throughout the playing field.

7.7 - Defining the multiplayer area

Because the island is so large (it's 400 square kilometers), it's necessary to enclose the battlefield. This area can be visible on the map and in the landscape. The game already has an integrated function available to make this possible. The function enables the creator of the deathmatch mission to define his playing area. An object needs to be placed on the map which should ideally be an invisible heli-pad. This object defines the center-point. The syntax only needs to be entered in the init line of this object. The invisible heli-pad can be found in units **(F1) Empty/Objects**.

This object needs to be placed in the middle of the gaming field. Then enter following syntax into the init line:

```
Area1 = [this,400,400,100,10] execVM "area.sqf"
```

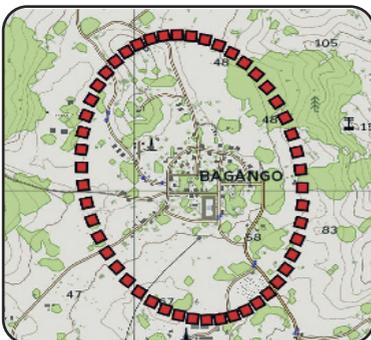
There is no need to script your own **area.sqf**. It's already implemented in the engine!

The object, in this case the heli-pad, is renamed to **"Area1"** automatically when the mission begins. Some warning signs, which define the outer border of the battlefield, will be generated automatically around the heli-pad. It's up to the definitions in the syntax how large or small this size will be in the mission.

The syntax explains itself as follows:

```
Name = [Center, X-Value, Y-Value, Number of Objects, Angle]
```

All Values are variable and freely definable. You can see an example about how it might look in the images below:



One can save a lot of work by using this option. Otherwise all those objects need to be placed on the map individually. It's now also possible to place dead-zones around the battlefield to avoid players leaving the gaming area.

7.8 - Time and rating

The limitation of the time and its rating are multiplayer settings which can be adjusted, but one needn't do this. To do this two parameters within the description.ext need to be defined which are looking like as follows. They are also displayed later in the multiplayer lobby.

Time limit

```
titleParam1      = "Time limit:";
valuesParam1[]  = {10000, 300, 600, 900, 1200, 1500, 1800, 2100, 3600, 7200};
defValueParam1  = 1800;
textsParam1[]   = {"Unlimited", "5 min", "10 min", "15 min", "20 min", "25 min", "30 min", "35 min", "60 min", "120 min"};
```

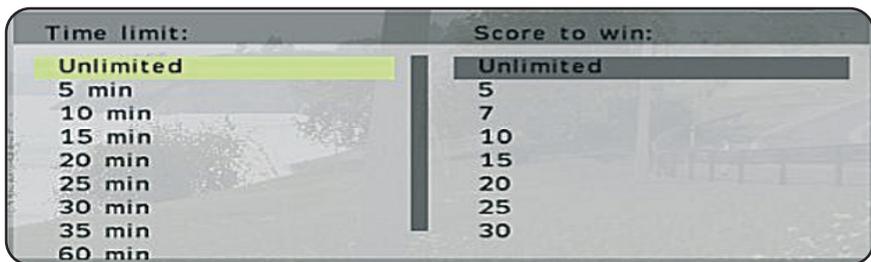
All values in here are variable. The time values are selectable right now within the menu. The appropriate text of time is defined in the last line and can also be freely defined

Score limit

```
titleParam2      = "Score to win:";
valuesParam2[]  = {10000, 5, 7, 10, 15, 20, 25, 30};
defValueParam2  = 5;
textsParam2[]   = {"Unlimited", 5, 7, 10, 15, 20, 25, 30};
```

The values are freely adjustable here as well and are representing the scores which are needed to finish the mission successfully.

As one can see on the image below, the defined parameters above are used as well as they have been defined. Now they are selectable within the multiplayer lobby.



By doing this the rough configuration has been finished. The user now has the possibility to adjust the score and time settings as he likes to do. The only missing thing right now is the configuration on the map which tells the game, that the needed score has been reached or the respective time ran out. So now we come to the checking triggers.

Checking Trigger

Additionally to all the above, we now need three triggers which are checking the advance. Those triggers have to be defined as follows:

Trigger 1 (Time guard):

Name: TimeEnd
Condition: (Param1<10000) and (time>=param1)
on Activation: EndOfMission=true
Axis a/b: 0/0



Trigger 2 (Score guard):

Name: ScoreEnd
Condition : (Param2 < 10000 and (({score_x} >= Param2}) count [S1, S2, S3, S4, S5, S6] > 0))
on Activation: EndOfMission=true
Axis a/b: 0/0



Trigger 3 (End trigger):

Typ: Ende 1
Condition : EndOfMission
Axis a/b: 0/0



There're units defined within the condition line which are named **S1, S2, S3, S4, S5, S6**. These can be renamed as one wants, but always make sure that the new names are adapted within the condition line of trigger 2.

To make sure that the points of an AI unit will be counted, it has to be predefined in the Description.ext with **AIkills=1**.



7.9 - Assigning and displaying scores

There're several ways existing to get the reached score to be displayed. That may happen if someone has conquered a enemy flag, executed a special mission target or just got points another way. But that depends on the missions storyline.

The Description.ext needs to be defined first as explained in **Chapter 7.8**.

Score limit

```
titleParam2           = "Score to win:";
valuesParam2[]        = {10000,5,7,10,15,20,25,30};
defValueParam2        = 5;
textsParam2[]         = {"Unlimited",5,7,10,15,20,25,30};
```

The score variable needs to be defined within either the Init line of the unit or in the init.sqs which are defined with

```
WestScore = 0
EastScore = 0
```

here to assign the value **0** right at the beginning of the mission. Each side shall receive points for executing a mission target, killing an enemy or whatever. to enable the mission to allocate score, the following syntax is needed **WestScore = Westscore +1**. Each time a predefined condition has been executed the syntax above will add the value **1** (1 point) to the respective side's score count. To remove points again the same syntax is needed with the only difference **Westscore = Westscore -1**.

If one wants to get the score displayed, there're several possibilities. Maybe by using a radio trigger which can be used when it's needed, or that the points will be displayed on the screen for a few seconds when the opposing side has got one or some points.

To do this, the following syntax will be used regularly:

```
TitleText [format [localize "STR_MP_STATUS", WestScore, EastScore], "Plain down"]
```

A radio trigger could look like this:

Getting score displayed (radio trigger)

Activation	Radio Alpha Repeatedly
Axis a/b	0
Text	@STR_MP_SHOWSCORE
on Activation	TitleText [format [localize "STR_MP_STATUS", WestScore, EastScore], "Plain down"]



7.10 - Time display

In some missions it might be useful to get the time displayed to inform the player that the mission will end soon and to heat them up to making some points before it's over. An approved method is to use several checking triggers within the mission which are getting executed when the predefined time has run out. These triggers will be defined as follows so far we've defined some parameters to the description.ext.

Time limit

```
titleParam1 = "Time limit:";
valuesParam1[] = {10000, 300, 600, 900, 1200, 1500, 1800, 2100, 3600, 7200};
defValueParam1 = 1800;
textsParam1[] = {"Unlimited", "5 min", "10 min", "15 min", "20 min", "25 min",
                 "30 min", "35 min", "60 min", "120 min"};
```

The last two lines are actually to be defined within one single line which is not possible right here.

The following example will show how the code for a checking trigger looks, which will display that there's only **one hour** left to play

Time Display (checking trigger)

```
Condition (Param1<10000) and (Param1>=3600) and ((Param1-time)
<=3600)
Axis a/b 0
on Activation hint localize "STR_MP_07"
```

Now the respective value actually needs to be defined for each further trigger. In this example the string table basic values were used as explained in **Chapter 7.13**.

Here is a closer explanation of the condition:

(Param1<10000) and (Param1>=Value) and ((Param1-time)<=Value)

Value stands for the value which was defined at valuesParam1 in Parameter1 (i.e: 600)

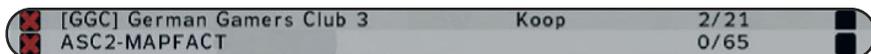
(Param1<10000) and (Param1>=600) and ((Param1-time)<=600)

One would now define the correct string table value in **on Activation** right now which would be displayed then. In this case it would be:

Hint localize "STR_MP_04" - 10 Minutes left

7.11 - The class header

The class header is just a definition which needs to be defined in the description.ext. Its quite necessary to define the class header because all the needed information will be displayed here. It contains the minimum and maximum numbers of players and the mission type which will be displayed to the player. It's also needed because it lets the player know the required information which he or she may need to decide which server to join. The image



below

shows two servers. The 1st one has a class header definition in its description.ext, while the one below hasn't got one, and the result can be seen in the mission-type.

GameType: The mission type will be defined here:

- SC** - Sector Control
- DM** - Deathmatch
- CTF** - Capture The Flag
- COOP** - Cooperation
- TEAM** - Team

MinPlayers: The minimum numbers of players

MaxPlayers: The maximum numbers of players

The example below shows the final edited class header in the Description.ext

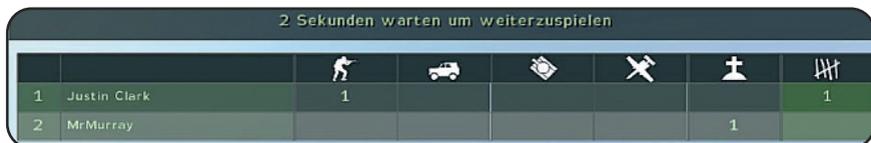
```
class Header
{
    gameType = CTF;
    minPlayers = 2;
    maxPlayers = 8;
};
```

7.12 - The respawn dialog

The Respawn dialog is the dialog which will be displayed when the player is killed. It displays the time which is remaining until the player can be respawned into the mission. One can activate or deactivate this option by using following entry which has to be defined in the Description.ext:

respawnDialog = false;

If the dialog needs to be visible again, just change false to **true**.



7.13 - Stringtable MP basic values

There're a lot of stringtable values already available by default in the game, which can be used without a Stringtable.csv. The following list contains a little overview of the most important stringtable values which belong to the Multiplayer Missions:

STR_MP_POINT_W	- US scored a point
STR_MP_POINT_E	- SLA scored a point
STR_MP_FLAG_TAKEN_W	- Name has the US flag
STR_MP_FLAG_TAKEN_E	- Name has the SLA flag
STR_MP_FLAG_BACK_W	- The US flag has returned
STR_MP_FLAG_BACK_E	- The SLA flag has returned
STR_MP_SECTOR_ATTACK_W	- US is attacking sector XY
STR_MP_SECTOR_ATTACK_E	- SLA is attacking sector XY
STR_MP_SECTOR_W	- Sector XY is now under US control
STR_MP_SECTOR_E	- Sector XY is now under SLA control
STR_MP_02	- 2 minutes remaining
STR_MP_03	- 5 minutes remaining
STR_MP_04	- 10 minutes remaining
STR_MP_05	- 20 minutes remaining
STR_MP_06	- 30 minutes remaining
STR_MP_07	- 1 hours remaining
STR_MP_NOLIMIT	- Unlimited
STR_MP_SCORE	- Scores
STR_MP_SHOWSCORE	- Show Scores
STR_MP_TIME	- Time
STR_MP_STATUS	- Status – USA: SLA:
STR_MP_GAMEOVER_W	- The US won!
STR_MP_GAMEOVER_E	- The SLA won!
STR_MP_GAME_DESC_CTF	- Two opposing teams, two flags. Capture the enemy flag.
STR_MP_GAME_DESC_DM	- Every man for himself. No-one is your ally.
STR_MP_GAME_DESC_SCONTROL	- Take control of the designated sectors to gain points.
STR_MP_GAME_DESC_PILOTDOWN	- Find and rescue the pilots.
STR_MP_GAME_DESC_HOLDCASTLE	- US: conquer the castle SLA: defend the castle

To get them displayed, just use the already known ways to display text on the screen. See **Chapter 7.17** to get more information.

7.14 - The vehicle respawn

The vehicle respawn can be used in two ways. The default way and the self-made way, which I'd like to present in the next point. But first the default variant:

Every vehicle which shall respawn needs to get a special entry entered into its init-line. This entry defines the individual configuration. The Syntax is as follows:

Vehicle1 respawnVehicle [Time,Number]

Vehicle1 respawnVehicle [Time]

If one defines an individual respawn time-value to this vehicle, the game will ignore the time that is defined in the Description.ext, and will use the **time** which was defined for the vehicle directly. If the **number** for respawns in the init line was defined as **0**, the vehicle would respawn eternally.

The Description.ext needs some standardized lines which need to be defined to make it work:

- respawnVehicle=3;** - The type of respawn
- respawnVehicleDelay=10;** - The time which is left until the vehicle can spawn again

The different kinds of respawn:

Vehicles only have 2 ways to respawn. These are to be respawned at "**The place of death(2)**" or to be respawned at a "**Predefined location (3)**". To define the type of respawn, the following syntax needs to be written in the Description.ext:

- respawnVehicle=3;** - The kind of respawn
- 0** or "**None**" - No Respawn
- 2** or "**Instant**" - Respawn at the place of death
- 3** or "**Base**" - Marker respawn (respawn_west, ...)

The Respawn Points

The respawn points used for vehicles are called as follows:

- West: **Respawn_Vehicle_West** Resistance: **Respawn_Vehicle_Guerrilla**
- East: **Respawn_Vehicle_East** Civilian: **Respawn_Vehicle_Civilian**



7.15 - Mr-Murrays vehicle respawn

Beause the default variant doesn't offer many features, so I worked around a little bit with that subject and recreated the following solution. The special thing is that this one doesn't need an entry in the Description.ext.

At first one needs place a vehicle on the map which is to be used for the vehicle respawn. This vehicle has to be named **Veh1** so it further needs a fixed respawn point which will be represented by a game logic, named **Veh1Pos**. Now we have a vehicle with defined spawn point. What we further need now is a checking trigger which is checking whether the vehicle is still alive or not or has the ability to drive (depends on level of damage).

Each vehicle which shall be resapwnable receives it's own **positions point** (Logic) and it's own **checking trigger** with the respective name of the vehicle.

Checking trigger (Vehicle guard)

Name: Veh1Guard
Condition: !Canmove Veh1
on Activation: [Veh1,"M1Abrams",Veh1Pos,360,2,10,1,0,0]
exec "vehicle-respawn.sqs"
Axis a/b: 0/0



Logic (Positions point)

Name: Veh1Pos



The **Array** which is present in the Activation line of the checking trigger (above) contains all settings which are important/needed. So one doesn't have to switch back into the script but has also the possibility to adjust all the things only by configuring the array. This one can be explained as follows:

[VehicleName, VehicleClass, RespawnPosition, Azimut, RespawnTime, RespawnNumber, Delete, Static, Effect] exec "vehicle-respawn.sqs"

VehicleName	- Name of the vehicle
VehicleClass	- Class of the vehicle (see Chapter 3.7)
RespawnPosition	- Point where object will spawn again
Azimut	- Direction of view (Value of direction of view)
RespawnTime	- Time left to the next respawn
RespawnNumber	- Number of respawns (0=No Respawn)
Delete	- Delete vehicle (0=no; 1=yes)
Static	- Respawn static or flexible (0=Flexible; 1=Static)
Effect	- Explosions effect while deleting (0=No Effect; 1=Effect)

This Array is calling the following script: **vehicle-respawn.sqs**.

```
?!(Local Server): exit

_vehicle = _this select 0
_vehicleClass = _this select 1
_respawnArea = _this select 2
_azimutCode = _this select 3
_respawnDelay = _this select 4
_respawnRate = _this select 5
_deleteVehicle = _this select 6
_staticRespawn = _this select 7
_deleteEffect = _this select 8

_counter = 0

#Start
~1
?(Canmove _Vehicle) : goto "Start"

;// Respawnrate (Number of Respawns)
? (_counter >= _respawnRate) : exit
_counter = _counter + 1

;// Staticposition (0=Flexible/1=Static)
? (_staticRespawn == 0) : _respawnArea setPos getPos _vehicle
? (_staticRespawn == 0) : _VehAzimut = getDir _vehicle

~_respawnDelay

;// deleteVehicle (0=No Delete/1=Delete)
? (_deleteVehicle == 0) : goto "Respawn"
deleteVehicle _vehicle

;// Delete effect
? (_deleteEffect == 0) : goto "Respawn"
_bomb="M_Javelin_AT" createVehicle [0,0,1000]
_bomb setPos getPos _vehicle

;// Respawn
#Respawn
~2
_vehicle = _vehicleClass createVehicle getPos _respawnArea
_vehicle setDir _azimutCode
? (_staticRespawn == 0) : _vehicle setDir _VehAzimut
goto "Start"
```

7.16 - Flag basic informations

To create missions like Capture the Flag (CTF) or Sector Control (SC) or similar, a basic knowledge about using flags is needed which will now be handled in this subitem. At first it's important to explain that all flags can be allocated to a special side. If a flag has already been allocated to a side (maybe West), this side does not have the possibility to capture the flag, while all the other sides will have.

The needed syntax is called:

this setFlagSide SIDE

This Syntax has to be defined into the init line of the respective flag. So each flag will be configured now individually by receiving an own entry into it's init line. The following is an example about how to define a flag:

this setFlagSide WEST; this setFlagTexture "Flag.jpg"

Once both of these entries have been made, the flag is finally configured. The flag has got a side and a flag texture allocated. See **Chapter 5.35** for flag textures or the next page. Additionally to this, each flag has to be named as a variable. I.e. **FlagWest** for the West flag and **FlagEast** for the East flag.

So if an East soldier is getting close to the West flag, he'll have the possibility to capture the flag, which will be placed now onto his shoulder, so that he has to carry it around. If he dies, the flag will keep lying next to his body. If another East soldier gets to the dead body, he has the possibility to grab the flag. But if a West soldier gets close, then the flag will automatically get beamed back to its Flagstaff.

There are still some commands to explain:

ObjNull

This value stands for zero, which is representing the fact that the flag is still on its flagstaff. If a player is grabbing the flag, the value is no longer ObjNull. It's possible to request or build that status.

Requesting of a status

```
flagOwner FlagWest == objNull  
flagOwner FlagWest == EAST
```

Building of a status:

```
FlagWest setFlagOwner objNull  
FlagWest setFlagOwner Name
```

FlagOwner

The player which currently owns the flag is also called the flagOwner. So this command is telling the engine which person is currently the respective carrier. If no one is currently using the flag, the command **objNull** will get returned. The needed syntax is called:

```
flagOwner FlagWEST  
flagOwner FlagWest == Name1
```

SetFlagOwner

By using this command a special unit will be defined as Flagowner. If the **objNull** is used as name, the flag will be beamed back to its staff. To do this just use following Syntax:

Returning to staff:

```
FlagWest setFlagOwner objNull
```

Giving to player:

```
FlagWest setFlagOwner Name1
```

Flag

One can get the information of which unit is currently carrying the **flag**. So it is a check of a flag owner. If no one is carrying the flag currently, the value **objNull** will get returned.

```
Flag Name1
```

Example: Checking for flag owner:

```
Flag Name1 == FlagWest
```

SetFlagTexture

One can define the texture of the flag just by using the syntax setFlagtexture. But this is also explained in **Chapter 5.35**. The syntax for custom flag (which has to be saved within the missions folder), is called:

```
this setFlagTexture "Flag.jpg"
```

A source path for default flags always have to be declared (see **Chapter 5.35**):

```
this setFlagTexture "\\ca\misc\data\usa_vlajka.paa"
```

SetFlagSide

By using this command, one can adjust the side of the flag. Units which belong to the defined sides are not able to pick up the flag. The syntax is called:

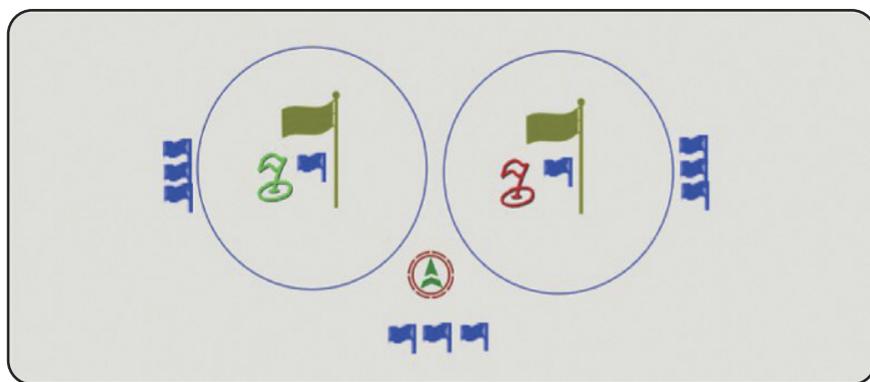
```
this setFlagSide SIDE
```

7.17 - Capture the flag

Capture the Flag is a special way of multiplayer gaming where one side has to capture the flag of the opposing side to get it back into their own area and receive points. The winner will be the side which is reaching the predefined number of points first, or receiving the most points within a predefined time window.

What sounds pretty easy for playing is not as easy to be realized within the Editor. Because there're thousands of possibilities to create such a mission, I'll introduce a special variant where the user has the possibility to define the needed CTF-Bases himself. That means, if one is placing all the following contents on the map, the functionality will already be proofed.

Once all the components have been placed on the map as explained on the following pages, the map should look like as follows:



The **West Flag (FlagWest)** is located on the left side with proper trigger area (**ZoneWest**), a green marker (has only a marking meaning) and three further checking triggers, located on the left of the checking trigger area.

East has almost the same components but with different values. The East Flag is named **FlagEast** and the trigger area, **ZoneEast**. Additionally to this, there is also the red marker which is representing the position of the flag on the map. Also on the right are the three further checking triggers.

The further three triggers right below the player, containing an **initiating trigger** a time and a score **guard trigger** and finally the **end trigger** which is exiting the mission once the predefined conditions have been executed.

This example doesn't contain any respawn or similar stuff, only the pure CTF content is to be seen here which is securing the CTF-functionality.

CTF-Components

All the needed components including their configuration will be shown here individually. For the parameters of Activation, Condition or init line some contents are written in several lines, so it represents one single line only. But this is not possible due to the size of the book.

W E S T S I D E

Flag West:

Name: FlagWest
Init: this setFlagSide WEST;
this setFlagTexture "\\ca\misc\data\usa_vlajka.paa"



Area West (Trigger area):

Name: ZoneWest
Axis a/b: 10/10



If the conqueror of the East flag reaches this trigger area, West side will score which will become visible by a display on the screen

Conqueror East Flag (Checking trigger I):

Axis a/b: 0/0
Condition: not isNull flagowner FlagEast and
flagOwner FlagEast != OwnerEast
on Activation: OwnerEast = flagOwner FlagEast;
titletext[format [localize "STR_MP_FLAG_TAKEN_E",
name OwnerEast], "Plain down"]; FlagManW=true



This trigger is checking who is currently the owner of the flag. As long as the flag is on the flagstaff, nothing will happen, but if a West unit gets the flag, a text message appears which displays the name of the player who got the flag.

Conqueror of the East Flag is getting score (Checking trigger II):

Axis a/b: 0/0
Condition: (OwnerEast in list ZoneWest) and
not FlagManE and not Wert 1
on Activation: Kdo=OwnerEast; OwnerEast=objNull;
FlagEast setFlagOwner objNull;
WestScore=WestScore+1;
titleText[format [localize "STR_MP_POINT_W",
WestScore, EastScore], "Plain down"];
FlagManW=false; Kdo addScore 5;
{_x addScore 5} forEach units Group Kdo



This trigger is now checking whether the conqueror of the East flag, so the **FlagOwner FlagEast**, has entered the trigger area **ZoneWest**. The West side will get a score value allocated now that will be displayed by a **screen message**.

Conqueror of East Flag is losing flag (Checking trigger III):

Axis a/b: 0/0

Condition: (isNull flagOwner FlagEast) and FlagManW

on Activation: FlagManW=false;

titleText[localize "STR_MP_FLAG_BACK_E", "Plain down"]



This third trigger is checking now whether the flag is still owned by the conqueror. If he dies and one of the East units is getting the flag again from the body, so the flag will be beamed back to its flag staff. A new text message appears on the screen that East got his flag back.



E A S T S I D E

Flag East:

Name: FlagEast

Init: this setFlagSide EAST;

this setFlagTexture "\\ca\misc\data\rus_vlajka_co.paa"



Area East (trigger area):

Name: ZoneEast

Axis a/b: 10/10



If the conqueror of the West flag is reaching this trigger area, so East will get a score value allocated, which will appear again by a Text message.

Conqueror West Flag (checking trigger I):

Axis a/b: 0/0

Condition: not isnull flagOwner FlagWest and
flagOwner FlagWest != OwnerWest

on Activation: OwnerWest = flagowner FlagWest;
titleText[format [localize "STR_MP_FLAG_TAKEN_W",
name OwnerWest], "Plain down"]; FlagManE=true



This trigger is checking for the owner of the West flag. So long as this one is still on its staff, nothing will happen. But so far a East unit is getting the flag, so a text message appears again which says which player has just got the flag.

Conqueror of West Flag is receiving score (checking trigger II):

Axis a/b: 0/0

Condition: (OwnerWest in list ZoneEast) and
not FlagManW and not Wert1

on Activation: Kdo=OwnerWest; OwnerWest=objNull;
FlagWest setFlagOwner objNull;
EastScore=EastScore+1;
titletext[format [localize "STR_MP_POINT_E",
WestScore, EastScore], "Plain down"];
FlagManE=false;
{_x addScore 5} forEach units group Kdo



This trigger is checking now whether the conqueror of the West flag, so the **FlagOwner FlagWest**, has entered the trigger area **ZoneEast**. If this condition is executed East side will get a score value allocated now which will again be displayed by a screen message.

Conqueror WestFlag is losing flag (checking trigger III):

Axis a/b: 0/0

Condition: (isnull flagOwner FlagWest) and FlagManE

on Activation: FlagManE=false;
titleText[localize "STR_MP_FLAG_BACK_W","Plain down"]



This third trigger is checking now whether the flag is still owned by the conqueror. If he dies and one of the West units gets the flag from the body, the flag will get beamed back to its flagstaff. A new text message appears on the screen that West got his flag back.

Additions:

Once all the functionalities have been defined for each side, only the **initializations trigger**, the **time-** and **score guard trigger** and the **end trigger** are missing, to become the mission ended.

CTF-Init (Initialization trigger):

Name: CTF-Init
Axis a/b: 0/0
Condition: true
on Activation: Wert1=false; WestScore=0; EastScore=0;
OwnerWest= objNull; OwnerEast=objNull;
FlagManE=false; FlagManW=false; Zeit=0;



This trigger could alternatively get lost, but the commands from the **onActivation** line would need to be defined within the **Init.sqs**. So I decided to use this variant for functionally and safer reasons. Different variables will be defined here which are cooperating with the triggers of each side.

Evaluation (Checking trigger):

Name: ScoreTimeGuard
Condition: (param1<10000 and ((time >= param1) or (Zeit >= param1))) or (param2<10000 and ((WestScore>=param2) or (EastScore>=param2)))
on Activation: Zeit=Time; publicVariable "Zeit"; Wert1=true; titletext[format [localize "STR_MP_GAMEOVER_FINAL", WestScore, EastScore], "Plain"]; EndOfGame=true



This trigger is checking the Time and score settings which have been predefined by the Admin right before the mission begins. The mission will be finished so far the respective conditions have been executed. In this example the mission will be closed immediately, because **EndOfGame=true** has been defined in the onActivations line. But this could get lost and an alternative script like an Outrosript (i.e.: [] exec "outro.sqs") can be implanted to set the variable **EndOfGame** on **true** right at the end of the script. So this trigger is checking whether the respective variable value (**WestScore**, **EastScore** for the score or **Time** for time) has reached or exceeded the predefined parameter value (**Param1**(score), **Param2**(time)) . If it does, so the condition has been caused and the trigger will get executed.

Take care that the time and score parameters have been defined as explained in **Chapter 7.8**. Please only edit the Description.ext entry, nothing else because the trigger has already been configured.

End Trigger (Checking trigger):

Name: EndGuard
Condition: EndOfGame
Typ Ende 1



The end trigger will get caused so far the variable **EndOfGame** has been set on **true** and is finishing the mission. It is advantageous to add a little time delay by adjusting the Min-Mid-Max settings to avoid the trigger getting executed immediately but rather a few seconds later.

Displaying the score (radio trigger):

Activation: Radio Alpha
Repeatdety
Axis a/b: 0
Text: @STR_MP_SHOWSCORE
on Activation: titleText[format [localize "STR_MP_STATUS",
WestScore, EastScore], "Plain down"]



This trigger can get inserted as well although is not visible on the main graphic. This enables all players to request the current score status just by using the radio chat Alpha.

Score and time definition within the Description.ext:

To complete the score and time control, they have to be defined as explained in **Chapter 7.8**. But here are the needed lines also which are used within the Description.ext.

Time limit:

```
titleParam1 = "Time limit:";
valuesParam1[] = {10000, 300, 600, 900, 1200, 1500, 1800, 2100, 3600, 7200};
defValueParam1 = 1800;
textsParam1[] = {"Unlimited", "5 min", "10 min", "15 min", "20 min", "25 min",
"30 min", "35 min", "60 min", "120 min"};
```

The last lines are actually representing one single line, but this is not possible here.

Score limit:

```
titleParam2 = "Score to win:";
valuesParam2[] = {10000, 5, 7, 10, 15, 20, 25, 30};
defValueParam2 = 5;
textsParam2[] = {"Unlimited", 5, 7, 10, 15, 20, 25, 30};
```

7.18 - The public variable

The public variable has to be used in Multiplayer games only, that's why its explained in this chapter and not in the Scripting chapter. One can basically compare it with the global variable, although it's nevertheless a little bit different. By using the public variable one makes a global variable become public. That means that the information will be sent from a client to all of the other ones and even to the Server or the Host also to execute the respective Action.

Lets differentiate again:

Local Variable Valid in local areas only. As an example in a script(SQS) or in a function(SQF). The variable is recognizable because of the underscore right before the variable. Example: **`_Variable`**.

Global Variable When it comes to the global variable then it means that a unit or an object, which was named, effects the whole mission. The best example is the case if a second unit/object should receive exactly the same name. An error message will appear that this variable name is already in use.

If one is playing a Multiplayer Mission, so the Client will be recognized as a local variable, what means that the global variable, which is meant for the whole Mission onto the whole Map, turns to a local one. This is why that all happens for all clients (player) individually. That all is not a Problem so long the respective units (Clients) has received fixed names. Because every Client, who is joining the Mission is receiving his values. But if the creator of a Mission wants to allocate a value to a variable, i.e. **true** or **false**, this will only take effect for the respective player who is executing such a trigger, waypoint etc. So this information now needs to be sent to all the other clients become to make the result of the caused action visible for every client.

To make a local variable public, just use following syntaxes:

```
Target1=true; publicVariable "Target1"
```

So one turns **Target1** on **true** and makes this Value public by using the additional command **publicVariable "Target1"**. This information will now be sent to all of the other clients who are currently in the server or the host.

If one wants to use 2 or more variables which should be set on **true** or **false**, so they need to be defined within the **Init.sqs** before. Then they can be defined as public variables and the respective value can be set, as shown in the example below:

```
publicVariable"Variable1"  
publicVariable "Variable2"  
Variable1=FalseVariable2=False
```

7.19 - Preface information for MP missions

This subchapter will explain some basic things, which are also important while creating Multiplayer Missions. Because there is a big difference between editing Single player- or even Multiplayer Missions, so it's recommended to know where to take care while Multiplayer editing.

Dedicated Server

The dedicated Server is a fixed assigned Server, where all Clients and the Administrator need to access from outside. So it is only a server and nothing else, no Player is playing on this machine. This server can also be located all over the world and it's also possible to rent such a server at your own Internet provider. This server will then be controlled ingame by the Key commands which are explained here in **Chapter 7.20**.

Host

A host is a Computer where a Mission will be served and a player is playing at the same time. The host needs to open (host) a mission, where all the other players can get access on. The host is playing right with the same Computer, so he is server and Player.

Client

A Player who is joining a Mission is called a Client. It doesn't make a difference whether the Mission will be offered by a server or a host. If all Players are using a dedicated server so all (incl. the admin) joining Players are Clients. But if a Player is offering a Mission by hosting so he'll be the Host while all the other ones are the Clients



Difference between Singleplayer and Multiplayer

Every Mission creator who has already created Multiplayer Missions will know about the issue that all the nice features which were hard to create or edit will run clearly within the Editor, but not later in the Multiplayer game. So it's important to recognize the single subchapters in this chapter to define the important contents in your own MP Mission.

Logic Server resp. Logic AI

It's always recommended to place a Logic on the map which is called Server. An alternate variant would be a Logic called AI, as called by BIS. That truly makes no difference whether a server or AI will be used. This Logic will be created by the server only and will not appear for the Clients. That becomes known that this System is the server. So one can define the actions which shall run on the Server by using the following commands.

? !(local player) : exit

This Syntax right at the beginning of a script makes sure that this script is running for the Clients only. So the System will get checked whether it's a server or a Client. Because every Player is getting the fixed variable Player allocated automatically by the System, it knows that it may run the script. The information will be delivered to all Clients.

If the Player is hosting this Mission he'll nevertheless be able to execute the script, because he is the Client and Server in one (person) System.

? !(local server) : exit

If this Syntax is defined right at the beginning of a script, this script will be executed on the Server only. The Server can't tell the difference between a local or a dedicated host. This script will get executed normally for both versions, as long the system is hosting or serving.

This Syntax should basically get used. It will work for the server resp. as a global machine, which makes regulary more sense. Because most of the information needs to be given to the Players as well. So we're always working global! An alternate option would be **isServer**. Examples:

? isServer : hint "Server" or ?! isServer : hint "Not Server"

Script for Players

If one wants to get a script executed which is determined for a special Player only so that fact needs to be defined right at the beginning of the script. At first the unit should be named within the Editor. When that happens, then the Syntax needs to be defined.

An example:

player != Soldier1 : exit

If the player is not named Soldier1 within the editor the script will exit. That could happen if a Sound, or a text message, hint or what ever shall appear for one special Player only.

Two further Syntax examples. Is Player1 == Soldier1 then do this or the second Syntax, if the players name will be == "Mr-Murray", then do this (here: exit)

(player == Soldier1) : exit

(name vehicle player == "Mr-Murray") : exit

7.20 - The controlling commands

This Section has actually nothing to do with Editing, but it's nevertheless worth an explanation because nearly every user will join MP Missions sooner or later, or wants his Missions to be tested.

The Multiplayer section offers certain orders, which enable the Client among others, to vote the Admin, to vote for a cheaters kick and lots more. But the Server admin has a bunch more possibilities. These commands are nearly the same which were used in the predecessor Operation Flashpoint®. Just three names were added. The following is a list of the commands and their explanations.

Administrator commands

#login password	Login as Administrator
#logout	Logout as Admin
#vote admin (name or ID)	Vote the Administrator (Even for clients)
#mission filename	Select a mission with known Name
#missions	Is calling the Mission selection menu
#restart	Restart the Mission
#reassign	Restart and reselect of a rolle
#kick (name or ID)	Is kicking Player by Name or ID
#shutdown	Restart the server
#init	Is reloading the Server config file
#monitor (Interval in Sek.)	Displays Performance Informationen of the Server (Time in seconds. 0 stops the Monitoring)
#debug	Calls certain information (checkfile, console, totalsent, usersent, userinfo, userqueue)
#exec users	Displays a list of all connected Players
#exec kick ID	Allows to kick a Player from the Server
#exec ban ID	Allows to ban a Player from the Server

Client commands

#vote missions	Vote for Mission selection
#vote mission (name)	If a special mission is preferred (vote)
#vote kick (name/ID)	Vote for Client kick
#vote restart	Vote for restart the Mission
#vote reassign	Restart and reselect a role
#userlist	Displays a list of all connected Players

To call the different commands just use the chat function ingame. To do this just hit the – key, the same one which ebnables the underline _ (European Keyboard). After the game is expecting an input, type your preferred command and press enter.

7.21 - The armament within multiplayer

At first, there's no difference between the armament in Multiplayer or Singleplayer missions. The same method is used in Singleplayer as it is here as well. To remove or add a weapon the known commands like **removeWeapons** this and this **addWeapon** are just the same. But the really difficult thing happens when the client was killed and respawned on the map. Then the default armament of the respective class is used again. The client has to grasp his favourite weapons again.

The problem can be solved just by using a small script. Just place a trigger on the map, which is checking the status of the respective player, in this case **Soldier1**.

Checking trigger

Type: Repeatedly
Condition: ! alive Soldier1
on Activation [Soldier1] exec "weapon.sqs"
Axis a/b: 0/0



If **Soldier1** got killed now, this trigger will execute and run the script **weapon.sqs**. The script will make a break at **@alive_Unit** and will also wait until **Soldier1** respawns again. The delay depends on the adjusted respawn settings within the `description.ext`. But so far **Soldier1** will get spawned again, so the script will be activated and removes all weapons at first. Then the defined new armament will be added to the Character. Don't forget to define the magazines first so the weapons are already loaded with ammunition (as shown in the scriptfile below).

```
_Unit = _this select 0
@alive_Unit
removeallWeapons _Unit
_Unit addweapon "Binocular";
_Unit addweapon "NVGoggles";
_Unit addmagazine "8Rnd_9x18_MakarovSD";
_Unit addweapon "MakarovSD";
_Unit addmagazine "30Rnd_545x39_AKSD";
_Unit addweapon "AKS74UN";
exit;
```

7.22 -Text messages for a specific player

Text messages are actually quite easy to understand. But while in Multiplayer Missions, those messages will be displayed global, that means that up to the mission each player will receive or see that message. That takes us to the question how can a side related message be seen or if only one player has to receive the message, how do we go about doing this?

The solution is a small script again, where actually the first script line is the most important one. Because this line will define the player or the side which can see the message. Following Syntaxes shall help fixing that problem.

Used for players: **? (player == Name1) : exit**
 ? (player != Name1) : exit
 ? (name vehicle player != "Mr-Murray") : exit

Used for Sides: **? side Player == EAST : exit**
 ? side Player != EAST : exit

So it's possible to nearly create every syntax by yourself. That's only up to the used Parameters. The Syntaxes shown above are examples only.

Samples for usage

In the following example, the client who plays the unit named S1 will receive the message. For example, the leader.

```
? (player != S1) : exit  
hint format ["%1, you have a new task!", name S1];  
exit
```

So there're several ways to define the conditions. For example, if the player has a special name then he does or does not receive the message.

```
? (name vehicle S1 != "Mr-Murray") : exit  
hint format ["Hello %1, you are play unit XY.", name S1];  
exit
```

The ways of receiving messages appeared on the screen is really up to the user and what you personally prefer. So you can use hints as well as you can use screen messages or resource appearances.

```
titleText [format["%1, you have a new task", name S1],"Plain Down"]
```

7.23 - Join in progress (JIP)

Join in Progress is an important point which should always be looked out for and surly be considered for creating Multiplayer Missions. Players will always connect or disconnect while a mission is running. Especially if this mission is not password protected and runs as a public server. That makes the real problem appear. If a Client is joining while a mission is running, his machine needs to get the same status existing for the other clients. Especially the mission targets and variables need to be synchronized with each other.

If **Missiontarget1** is already done and also marked and the Variable **Start** is already set on **true** (Start=true), then these values need to be aligned on the just joined client.

Normally the server should do that automatically, so things like Mission targets or the status of variables would be compared with all clients. Unfortunately ArmA® doesn't provide this option in the current status of the game, so one has to help a little to make the mission playable on a dedicated server.

To do this the server regularly needs to call the public variables. So he would make the current variables public and all Clients would always get supported with the newest ones. That works also for just connected Players as well.

At first the Init.sqs, where all variables will generally receive one coherent value to become all players is up date.

Init.sqs

```
Var1=FALSE
Var2=FALSE
Var3=FALSE

? (! ( Local Server ) ) : goto "Skip"

PublicVariable "Var3";
PublicVariable "Var2";
PublicVariable "Var3";

#Skip
onPlayerConnected "Server exec ""scripts\update.sqs"" ";
```

The **onPlayerConnect** Line is a very interesting one. If a new client is connecting, the sever has to run the script update.sqs. This script should be stored in the user created folder scripts. The server Would call all public variables and would also synchronize the status of variables for all systems.

Update.sqs

The update.sqs is a pretty easy and small script. Only the public variables are defined in there which will become public by the server once it was created.

```
? (! ( Local Server )) : exit  
  
publicVariable "Var1";  
publicVariable "Var2";  
publicVariable "Var3";  
  
exit;
```

Mission targets

You can nearly do the same for the Mission targets. They should get synchronized if a new client joins in progress to receive the current status of the targets. You will realize it again by using a script or a function.

Ideally you will always let it run when a mission target was done or a new Player connects. The way to do this is almost the same as used for the public variables. Only the init.sqs will receive an additional line.

```
onPlayerConnected "[ ] exec ""scripts\TargetUpdate.sqs"" ";
```

TargetUpdate.sqs

```
? (! ( Local Server )) : goto "Skip"  
  
publicVariable "Var1";  
publicVariable "Var2";  
  
#Skip  
~4  
? Var1 : "1" objStatus "DONE"  
? Var2 : "2" objStatus "DONE"  
?! (alive Name1) : "3" objStatus "DONE"  
  
exit;
```

Variables were used as a condition for achieving an objective. An example is if a group has reached a special waypoint that was defined as a mission target. The variable **Var1** will get set on **true** and if the script gets called now and recognizes that **Var1** became true, so **Missiontarget1** will get set on the status Done.

Missiontarget3 is a further example which is done so Name1 is not active anymore. It is recommended to always call that small script when a mission target is done or a new player has connected.

Chapter 8

- Cam Scripting -

This chapter contains some hard Stuff. Camera scripting is not as easy as it seems to be, but the result can be compared with a Hollywood movie. Special features like intro's, outro's and some sequences while running the mission are quite necessary tools to better understand the characters and/or the storyline. Explaining everything about cam scripting would fill a whole book, so I will only explain the most important things here so that you will be able to create your own scenes.

8.1	Controlling the camera	255
8.2	The camera coordinates	256
8.3	Creating a camera	257
8.4	The first scene	258
8.5	Patching the camera on a vehicle/unit	260
8.6	Text and blending Effects	261
8.7	Camera effects	262
8.8	Preload objects and positions	262
8.9	Executing map animation	263



8.1 - Controlling the camera

It's actually quite simple to control the camera because only a few keys on the keyboard are needed. The following explanation about the different function keys is assuming the keys are still set at their defaults. Before the camera will get started, we should run it up first. So place a unit or an object on the map and enter following syntax into the init line.

Name1 exec "camera.sqs" or **this** exec "camera.sqs"

The following list shows the most important keys to control the camera, locating objects and defining positions. The option menu, especially the key configuration, offers many more possibilities, but we are not interested in those right now. The controls which are listed below will be enough to realize some good sequences.

Left Mouse button	- Save current coordinates
Move mouse forward and backwards	- Camera will do the same
Arrow key up	- Move camera forward
Arrow key down	- Move camera backwards
Arrow key left	- Move camera left
Arrow key right	- Move camera right
Numpad 4	- Rotate left
Numpad 6	- Rotate right
Numpad 8	- Rotate up
Numpad 2	- Rotate down
Numpad +	- Zoom in
Numpad -	- Zoom out
Picture up	- Lift camera up
Picture down	- Lower camera
L	- Crosshair on/off
V	- Camera off
Left Shift key	- Camera speed
Ctrl	- Select object (as shown below)



8.2 - The camera coordinates

When the user finally finds the position he wants to use in the movie, he only needs to press the left mouse button to save the current position (coordinates) in the RAM. This feature has changed a lot since Arma® has been released. While the user only needed to press the ctrl button in Operation Flashpoint® to save each position in an automatically created clipport.txt file, in Arma® the user has to go back to desktop by pressing Alt + Tab to save each camera position in his camera script.

This script is just an editor text file which can be renamed as the user wants. Only the intro and outro sequences need to be named a specific way (**Intro.sqs**, **Outro.sqs**). The saved camera positions need to be saved in this file by press the right mouse button and selecting paste out of the appearing context menu or just use the key combination **Ctrl+V**. The result looks like the example below:

```
;=== 0:06:18
_camera camPrepareTarget [101880.56,-28486.36,1887.85]
_camera camPreparePos [9626.16,10062.31,2.00]
_camera camPrepareFOV 0.691
_camera camCommitPrepared 0
@camCommitted _camera
```

The image above shows a newly created text block which has been saved in the RAM. This script contains following details.

;=== 0:06:18

The time of day. Not really important, can be deleted or renamed

_camera camPrepareTarget [101880.56,-28486.36,1887.85]

Direction of view of the camera can also be defined as an object name as well.

_camera camPreparePos [9626.16,10062.31,2.00]

Camera position (X,Y,Z)

_camera camPrepareFOV 0.700

Camera zoom. So smaller the value so higher the zoom factor.

_camera camCommitPrepared 0

This value defines the time which the camera needs from one position to the next one, defined in seconds. The position will change immediately if this value is still defined with 0.

@camCommitted _camera

The script will make a break here and wait until the camera has reached its next position.

8.3 - Creating a camera

When all the camera positions have been saved and finally edited, then the script needs a definition to create the camera and run the script. To do this just use the following command:

```
_camera = "camera" camCreate [9626.16,10062.31,2.00]
```

The values, which are defined in `_camera camPrepareTarget`, can be used as **X,Y,Z**. There's also the possibility to use **[0,0,0]** if the camera shall get further-placed immediately.

```
_camera camPrepareTarget [101880.56,-28486.36,1887.85]
```

```
_camera camPreparePos [9626.16,10062.31,2.00]
```

```
_camera camPrepareFOV 0.691
```

```
_camera camCommitPrepared 0
```

```
@camCommitted _camera
```

The camera position and the command to run up the script have already been defined at this point. Only the camera effects have yet to be defined.

```
_camera cameraEffect ["internal", "back"]
```

The following order makes the cinema border disappear, so that the movie will be displayed in a full-screen format:

```
showcinemaborder false
```

When the work is done then we have the very first part of our newly created camera script.

```
_camera = "camera" camCreate [9626.16,10062.31,6.00]  
_camera camPrepareTarget [101880.56,-28486.36,1887.85]  
_camera camPrepareFOV 0.700  
_camera camCommitPrepared 0  
@camCommitted _camera  
_camera cameraEffect ["internal","back"]  
showcinemaborder false
```

So far we have the first part of our script, but it only would give us a picture which shows a part in the landscape, and this is not quite spectacular. To get a movie with moving camera drives and scenes, there are some more things needed.



8.4 - The first scene

We are using the same current camera position as created in **Chapter 8.3**, but we don't use the angle of view of the camera. In this example we're about to use the name of a unit called **Aircraft1** in **_camera camPrepareTarget [101880.56,-28486.36,1887.85]** instead of the **[XYZ]** coordinates, so just delete them and enter **Aircraft1**. Furthermore the zoom needs to be adjusted, so we set it up to **600** to get quite close to our Harrier.

Now we only need another block of coordinates to tell the camera to move to this new position. After this position has been defined it will be used in the script and finally adjusted.

Intro.sqs

```
;Intro sequence
titleCut [" ", "BLACK IN"]; titleFadeOut 4
playMusic "Track1"

;Aircraft Position 1
_camera = "camera" camCreate [9626.16,10062.31,6.00]
_camera camPrepareTarget Aircraft1
_camera camPrepareFOV 0.600
_camera camCommitPrepared 0
@camCommitted _camera
_camera cameraEffect ["internal","back"]

showcinemaborder false

;Aircraft Position 2
_camera camPrepareTarget Aircraft1
_camera camPreparePos [9657.99,10121.22,1.04]
_camera camPrepareFOV 0.500
_camera camCommitPrepared 30
@camCommitted _camera
```

As one can see, two new lines have been added here:

```
titleCut [" " "BLACK IN"]; titleFadeOut 4
```

Fading from black into the sequence with a time of 4 seconds.

```
playMusic "Track1"
```

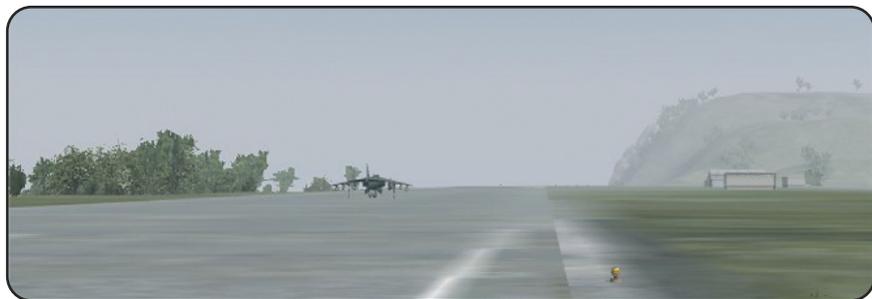
This syntax will run a music track to give some more atmosphere to the scene.

The camera is fixed now on **Aircraft1** and needs **30** seconds to reach the next position. Aircraft one is rolling and the camera starts to move to its next position. The camera is zooming in, but the jet is much faster and will disappear in the sky.

Aircraft Position 1:



Aircraft Position 2:



It's possible to add much more effects to this scene, but the basics should be explained well enough. The script only needs a command to be ended, so the lines below need to be added to the bottom of the script:

```
6 Fademusic 0
titleCut ["" , "BLACK OUT"]; titleFadeOut 4
~6
player cameraEffect ["terminate""back"]
camDestroy _camera
~1
playMusic " "
0 fadeMusic 1
exit
```

The scene is fading out slowly and the music is fading down as well. The camera will be deleted after 6 seconds and the player will get the control back and can start to play. It's quite important to make sure that the music track that is faded down will get faded up again. If you forget to do this and want to use the music again later in the mission, it won't be audible.

8.5 - Patching the camera on an vehicle/unit

One also has the possibility to hang the camera on a vehicle, so that the vehicle is pursuing the object. Its possible to add the camera at any position of the vehicle, but the vehicle needs to be generated first. In this example we are using a car which has been named "**Car**".

```
;Kamera erzeugen
_camera = "camera" camCreate [0,0,0]
_camera camSetTarget Auto
_camera camSetPos [0,0,0]
_camera camSetFOV 0.700
_camera camCommit 0
@camCommitted _camera
_camera cameraEffect ["internal";"back"]

; Position of camera in/at/about the vehicle
_car = Auto

;Position of camera (front/back/inside)
_dx = -6

;Position of camera (left/right/inside)
_dy = 0

;Highness of Camera (below/above/inside)
_dz = 2

#LOOP
;The following two blocks actually have to be defined in one single line, but
;this is not possible here:
_camera camSetTarget [(10 * sin (getdir _car))+ (getpos _ car select 0), 10*cos
(getdir _car)+(getpos _ car select 1), (getpos _ car select 2)]
_camera camSetPos [(getpos _ car select 0) + _dx * sin (getdir _car) - _dy * cos
(getdir _car), (getpos _ car select 1) + _dx * cos (getdir _car) + _dy * sin
(getdir _car), (getpos _ car select 2)+ _dz]
_camera camSetFOV 0.900
_camera camCommit 0
@camCommitted _camera

;We set a condition to end the script. In this case: If our car, gets closer than
;50 metres to the unit (P1), so the scene will be ended.

?P1 distance Auto < 50 : goto "Ende"
goto "LOOP"

#Ende
P1 cameraEffect ["terminate";"back"]
camDestroy _camera
exit
```

8.6 - Text- and blending effects

One has the possibility to define several kinds of screen text. To do this, just use following syntaxes:

```
titleCut ["Hallo", "Black Out"]; titleFadeOut 6  
titleText ["Test", "White In"]; titleFadeOut 6  
cutText ["Test", "Black In"]; titleFadeOut 6
```

The list below explains the possibilities individually:

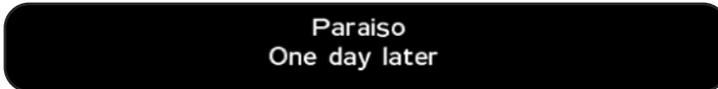
- Plain** - Text appears in the middle of the screen
- Plain Down** - Text appears at the bottom of the screen
- Black** - Blending out from the screen image to black
- Black Faded** - Blending out from the screen image to black as well
- Black In** - Blending back from black to the screen image
- Black Out** - Blending screen image into black
- White In** - Blending from white to screen image
- White Out** - Blending screen image into white

Linebreak

To insert a line break into the text just use the code `\n` at the part of the text which has to be broken. If one is using this code twice (`\n\n`), one will get an empty line and so on.

```
titleText ["Paraiso\nOne day later...", "Black In"]; titleFadeOut 6
```

The result can be seen in the image below:



Composetext

There's a further possibility besides default hint which is called composetext. This one can be freely defined when it comes to color and the font size. The Syntax below needs to be written in one single line!

```
hint composeText [parsetext format  
["<t size='1.2' align='center' color='#ff0000'>Hallo %1</t>", name player]];
```



Similar this Syntax merged with Text from a Stringtable.csv:

```
hint composeText [localize "STR_RA_M01V03",parsetext format  
["<t size='1.2' align='center' color='#ff0000'>%1</t>", name player]];
```

8.7 - Camera effects

This chapter will explain how to use camera effects in Sequences. Night vision goggle effect and special Sound effects are possible also.

Disable cinema border

The following command is deactivating the cinema border so that a full screen view is provided.

showCinemaBorder false

Night vision camera

To use the camera in Night vision mode following command is needed:

camUseNVG true

A more bright or darker ingame Graphic

The following command becomes the ingame graphic more bright or even darker

setAperture 1 - bright
setAperture 200 - dark

Let the camera say something's

The camera is to be handled similar to an Object, so the needed command is he formally known Say-command

_camera say "Sound1"

Disable environment sound

To disable the sound of birds or other sounds next to the music within the intro or other sequences just use the following command.

enableEnvironment false - Disables environment sound
3 fadeRadio 0 - Time fadeRadio radio volume
4 fadeSound 0 - Time fadeSound sound volume
2 fadeMusic 0 - Time fadeMusic music volume

8.8 - Preload objects and positions

The outermost well designed environment and the permanent loading and erasing of textures and Objects can cause incorrect displaying and jerking of the environment when the cameras position chances too fast. The System and the Engine are not able to follow that fast so those issues can sometimes happen.

To avoid these problems the Preload commands are provided. These commands are loading Objects, Positions, sounds or whatever into the RAM. Up to the command the camera will move to its next defined position so far this area or the used Objects has already been preloaded. Now a few Pre-load Syntax examples.

preloadObject - is preloading an Object (see the example)
preloadCamera - is preloading a Camera position (s. example)

- preloadMusic Track1** - is preloading a music track
- preloadSound Sound1** - is preloading a special sound
- preloadTitleRsc ["BIS", "Plain"]** - is preloading a resource
- preloadTitleObj ["BisLogo", "Plain"]** - is preloading a resource pic. an Object

The following two examples are describing the usage of **preloadObject** and **preloadCamera**. All files will get preloaded within the first example. The script will go on so far the preloading process has finished. The value **_preload** will be deleted automatically..

```
_preload = [] spawn {waitUntil {preloadCamera position Name}}
@scriptDone _preload
terminate _preload
```

The same will happen at **preloadObject**. The object Name will be load with a distance of 5.

```
_preload = [] spawn {waitUntil {5 preloadObject Name}}
```

8.9 - Executing map animation

It's quite interesting to get map animation executed within the briefing. It's also possible for the camera to move to differnt marker positions on the map. It's further possible to delete or edit markers and much more.

The following image will explain how to create such an Animation script. It's recommended to have the briefing and the radio disappear. It's further recommended to define 2 blocks for each marker as shown in the example. Block1 = zoom in, Block 2= zoom out. When these Blocks have been finished the script will go to the next Position.

```
forceMap true ;// Is forcing the map on the screen
showPad false ;// Is hiding the briefing
disableUserInput true ;// Is disabeling the user input

;// Zoom in marker Pos1 within 3 seconds to value 0.1
mapAnimAdd [3, 0.1, markerPos "Pos1"] ;// Is adding an animation
mapAnimCommit ;// Is executing an animation
@mapAnimDone ;// Is waiting until the animation has finished
~2

;// Zoom in marker Pos1 within 1 second to value 1
mapAnimAdd [1, 1, markerPos "Pos1"]
mapAnimCommit
@mapAnimDone
~1

;// Finishing mapanimation and reset original conditions
forceMap false ;// Closing the map
mapAnimClear ;// Is deleting values of MapAnimAdd
showPad true ;// Is reforcing the map on the screen again
disableUserInput false ;// Enables userInput again
exit;
```

Chapter 9

- Scripting -

This chapter will explain some of the scripting operations in more detail. This chapter will allow you to better understand more of the scripts that are shown in this guide and even allow you to be able to define some of your own small or large scripts.

9.1	The Variable	265
9.2	Logical Values	266
9.3	Logical Operators	267
9.4	The While-Do-Loop	268
9.5	The Counter	268
9.6	If-Then-Else	268
9.7	The Delay	269
9.8	Random	269
9.9	WaitUntil	269
9.10	The Brackets	270
9.11	The Semicolon	271
9.12	The Array	271
9.13	Basic knowledge about Functions	274



9.1 - The variable

A variable value is a changeable value. It can be a word or even a number depending on its intended use. There are two possible variables, the global and the local variable. While a global variable will work everywhere, a local one will work for a special thing only. Here is an example with variables:

Heli1 flyInHeight 120

If a user is placing a unit on the map, the units needs to be named in a special way. **MyHeli** for example. It's not possible to name a unit with a number (**1234**) only, but it's possible to merge a name with a number. **Heli1** for example.

Local Variable

A local variable can be recognized by its underline right in front of the variable. If the user defines a script, this variable will work in this script only in a localized section. So its possible to use this variable for several sections of the script without giving out more variables (for example, using the same variable for one or several units).

In the current example, we are using a script which was defined in a way so that 3 units called **Name1**, **Name2** and **Name3** have to execute an animation. But it might be that there are some more units on the map which shall be called on by that script, so its necessary to use a local variable. One has to define only one script and needs to use a local variable for a huge number of units.

The names which have to be executed by the script need to be defined as an **Array**:

[Name1,Name2,Name3] exec "script.sqs"

If the script gets started, so every single soldier of the **3** units will get allocated the local variable **_man**. Every single unit will get asked individually. The script would look like:

```
;Animation script

;Unit is getting local value allocated
_man = _this select 0

;Unit is executing animation
_man playMove "Animation";

;Script will exit
exit
```

The unit with, the global name **Name1** i.e., got the local variable **_man** allocated and will execute the given command:

Global Variables

Next to the local variables exists global variables. While a local variable will only work in a special predefined section, a global variable will work for a much wider section, as the name already implies. An example: If the user is renaming a unit so the name is a global variable, the name can be used only one time. If one wants to rename another unit with the same name, an error message will appear. The unit which has been named can be now be called by scripts, triggers, and waypoints - it's global.

Fixed variables

Some values are already used by the game, these are:

Player	- The Player
This	- Unit or object
Time	- Time of day in the game
_time	- Local time
_x	- An element out of an Array
_this	- Local unit
Pi	- 3,14...

Conditions of variables

It's possible to allocate conditions or values to a variable. Its also possible to set them on **true** or allocate them a text string.

Name1= true	- Variable is receiving the value TRUE
Name1= 44	- Variable is receiving a value
Name1= "MyText"	- Variable is receiving a text string
Name1= [Value1,Value2]	- Variable is receiving a array value

Saving variables

It's also possible to save variables at any time to call the again later

saveVar "Variablename"

9.2 - Logical values

A logical value is a special condition of a value. One can compare it with an **on/off switch**. If a variable has been set to **true**, a predefined action will get started. If the same action gets set back to **false**, this action will be ended. It's also possible to define **true** as **1** and **false** as **0**.

true	- Will be executed when a condition has been executed
false	- Will be executed when a condition has not been executed

9.3 - Logical operators

The list below explains some of the generally known and important operators.

AND	- A logical AND to combine two or more operations
OR	- Logical OR enables a controlled selection of two or more variables
NOT	- A logical NOT enables a controlled use of two or more variables
!	- Can be used as NOT as well
?	- IF
:	- DANN
If	- IF
Then	- THEN
Else	- ELSE
Exit	- Stops the execution of a script
Do	- Do (see While Do)
#	- Headline (Label) Note: Never set a semicolon behind a Label!
Goto	- Goto
>	- Bigger than
<	- Smaller than
<=	- Smaller or equal
>=	- Bigger or equal
==	- Equal
~	- Delay in seconds (~3)
;	- Will be ignored by the script
@	- Pauses the script and waits until the condition which follows next, has been executed
ForEach	- For each unit. Example {_x reveal Player} foreach List Area1
ThisList	- For each unit (Side) in a trigger area
Count	- Gives the number of existing elements of an array back top the script
Random	- Defines a random value
Case	- Case (ex.: case 1 : exit (translated: Is circumstance equal with value1 then exit)
Ceil	- Rounding value up. (Ex: ceil 5.25 would be 6 / ceil -5.25 would be 5)
Floor	- Rounding value down. (Ex: round 5.25 would be 5 / round -5.55 would be 6)
Round	- Rounding value up/down. (Ex: round 5.25 would be 5 / round 5.55 would be 6)

9.4 - The While-Do-Loop

This loop is going on so long as **a** is bigger than **b**. **A** will always receive the value **1** until it's bigger than **b**, so that the loop can stop. The maximum value for Armed Assault® is currently at around **100.000**.

```
While {a<b} do {a=a+1}
```

Translated: Add the value **1**, so long **a** is smaller than **b**.

9.5 - The counter

If one wants to use a counter in a script, the element value needs to be defined as **0** when the script or the mission begins. The value **0** will get allocated to the variable **counter**. Then the actual counter starts and always adds the value **1** to the local variable **_counter** at any cycle. But this will run only so long as the variable **_counter** is **>=** (bigger equal) **10**. Then script will end its sequence..

```
_counter = 0;  
#Start  
? (_counter>=10) : exit  
_counter = _counter+1;  
goto "Start";
```

9.6 - If-Then-Else

This syntax means exactly what it is called: **If-Then-Else**. An example: **IF** condition has been executed, **THEN** do **this**, or (**ELSE**) do **this**. Here is an example:

```
IF (a>b) THEN {c=1} ELSE {c=2}
```

An example: A marker has to be "patched" onto a unit so long this unit is still alive. If the unit will be killed, the script will exit.

```
#Start  
~0.5  
If(alive Soldier1)Then{"S1-Symbol" setMarkerPos getPos Soldier1}  
Else{"S1-Symbol" setMarkerType "Empty";exit};  
~0.5  
goto "Start";
```

If Soldier1 is still alive **Then** place **S1-Symbol** onto **Soldier1**, Or (**Else**) delete **S1-Symbol** and leave the script (**Exit**)

9.7 - The delay

The delay (written as “~” without quotes) will only be used in **SQS-Scripts**, while **Sleep** will be used in **functions** only. These orders will be defined as seconds. The script is counting down the given value and breaks the execution until the value 0 has been reached. When the count down has been finished the script will go on.

- ~300** - Script makes a break of 300 Seconds
- ~random 300** - Generates a random value and pauses the script break
- sleep 300** - Is a function. “sleeps” 300 seconds

9.8 - Random

The order random enables one to generate a random value. It's possible to define a random value with a variable. That could be such a script:

```
_start = random 4  
? _start < 1 : goto "Start1";  
? _start < 2 : goto "Start2";  
? _start < 3 : goto "Start3";  
? _start < 4 : goto "End";
```

A value, which has a maximum number of 4, has been generated randomly here and the script is checking the respective value. The script is jumping now to the respective Label which was defined with Start or End as shown above

That operator can be used another way as well, for example to place a unit at a random position inside a building, or to define a value to a delay. The needed syntax could look like this:

```
Name1 setPos (nearestBuilding this buildingPos random 10)
```

or the Delay:

```
~random Value
```

9.9 - Waituntil

The command waitUntil can be used if a special function, condition, or action gets paused. It's actually like the @ function. The function is waiting until this condition has been executed.

```
_Value = 0;  
waitUntil { _Value = _Value +1; _Value >= 100};
```

9.10 - The brackets

Certain kinds of brackets are to be used while scripting which have all their own properties. The following variants are different to each other:

[]	- Array (closer explanation in Chapter 9.12)
{ }	- Code (curly bracket)
()	- Mathematical Operators
" "	- Textstrings

Arma® gets the information from the brackets which it needs to combine all the code together.

{ } – Code

If some features are to be told to the engine in a special way like a code string, they need to be defined in a special kind of bracket, the curly bracket. A good example for such code is:

```
{_x moveInCargo Heli1} forEach Units GAlpha
```

All members of this group (**GAlpha**) should already start inside the chopper (**Heli1**) when the mission begins.

() - Mathematical Operators

Mathematical operators will be processed by using normal brackets (). That is exactly the same as the basic mathematic calculating rules related to brackets and the basic arithmetics called PEMDAS (US), BEDMAS (Can) or even BODMAS (UK, Austr. aso).

$(a+b)*c$

Example:

In the following example we want a unit to move to a special position inside of a building. An example about what we are telling the engine: **Unit** move to **House** plus **position value**. To make sure that the engine knows that the position belongs to the house, so we're using the brackets as shown below.

```
_Man move (_House buildingPos 120)
```

If there wasn't the brackets around Arma® wouldn't understand: **Unit** go to **House** AND then to a further position called **120**. So the engine doesn't know that the position belongs to the house and an error message would appear.

"Textstrings"

One can define text strings by using quotes (" "). It's alternatively possible to use ' This can be found regular on the @ key. Quotes can also be interlaced. Here is an example of the use of a map click.

```
onMapSingleClick "Leader Alpha1 move _pos; Player say ""Hello"""
```

9.11 - The semicolon

The semicolon means **separation** or even **break** in the script language. So it's possible to divide 2 or more command lines from each other (as you always do in the Init Line of a unit). It's further used to tell the engine that the current line has come to an end and the engine has to go on with the next line. So generally each script- or even function line has to be ended with a semicolon. It's further needed to know that signs which are located behind a semicolon will not be recognized by the engine. That enables the scripeter to define descriptions right behind a complicated script or function. An example:

```
;Animation Script
;Unit gets local value allocated
_man = _this select 0
playSound "Move";

;Unit is executing a animation
_man playmove "Animation";

;Leaving script
exit
```

One has the further possibility to divide commands from each other used in functions. One can see an example syntax, which contains two semicolons which are dividing the included code from each other.

Example = {private["_a","_b"];_d=[getPos player,_this select 0];_b = 100};

Attention! If one is using a delay like ~10 or the label #, so no semicolon has to be used right there, otherwise the script wont work correctly anymore!

9.12 - The array

One has the possibility to hand over values to a script or functions by using an Array. That provides the advantage that the same Array can be used very flexible for several units or similar stuff, while only using the local variable. So a function only has to be defined one time but can be used much more if needed. It's not fixed up to a special project or a special unit, but for a lot, one can see an Array as a list of several values which are collected within brackets to prepare them for a further usage.

Array

An Array consists out of squared brackets []. So all what will be listed within these brackets can get handed over to the script. Here's an example:

```
[Value1, Value2, Value3, Value4] exec "script.sqs"
```

The defined values within the Array will now be processed within the script as follows:

```
_Value1 = _this select 0  
_Value2 = _this select 1  
_Value3 = _this select 2  
_Value4 = _this select 3
```

Because we want to handle with local values within our script, a local variable, based in the Array, needs to be allocated to the respective value in the script. I think that this will become clearer now in the next example:

An object shall get created by using a script. The object we want to get created in this case is an empty **BMP2**, which has to be spawned on an invisible Heli H named **Point1**. This BMP should be aligned into direction **100** and also shall have a predefined fuel value of **0.5**. All the other things which actually have to be defined as well within this script are not to be focused here right now. So at first the Syntax with it's defined Array:

```
["BMP2", Point1, 100, 0.5] exec "script.sqs"
```

```
_Object = _this select 0  
_StartPos = _this select 1  
_Azimut = _this select 2  
_Fuel = _this select 3  
  
_Vehicle = "_Object" createVehicle position _StartPos;  
_Vehicle setDir _Azimut;  
_Vehicle setFuel _Fuel;  
  
exit;
```

Now in this script the values are getting a local variable allocated out of this Array. Those ones have been turned over as follows:

"BMP2"	receives the local variable value	_Object
Point1	receives the local variable	_StartPos
100	receives the local variable	_Azimut
0.5	receives the local variable	_Fuel

You can see very clearly in the bottom of the script that these local variables are now used in the script. The advantage is now that this script has been written one time but can be used as often as needed with different parameters. The following Syntax can also be used. The only difference is the **Random** command which defines a random value between **0** and **1**. That now becomes an Object, a helicopter in this case, created close to the position of the player.

```
["AH1W", Player, 100, random 1] exec "script.sqs"
```

Getting random values out of an Array

One has the possibility to get a random value automatically selected out of an Array by using the following command. This value could be used for somethings special later in the mission.

```
_array = [47,73,78,85,101,103];  
_Pos = _array select (random (count _array)-0.5);
```

The following example will explain it more: The unit **Soldier1** has to be moved onto a random position within the hotel by using the setUnitPos-value out of the call Array. But only the positions located in the first floor and predefined in the script have to be considered.

[Soldier1, "Middle", Hotel] exec "script.sqs"

```
_Unit = _this select 0  
_UnitPos = _this select 1  
_House = _this select 2  
  
_array = [47,73,78,85,101,103];  
_BPos = _array select (random (count _array)-0.5);  
_Unit setPos (_House buildingPos _BPos);  
_Unit setUnitPos _UnitPos;  
exit;
```

The system is selecting a value out of the script array and moves the unit to the desired position in the hotel. Please check **Chapter 5.61** to rename the building, because we're working here with an Object on the map



9.13 - Basic knowledge about functions

Functions have already been used in OFP resistance as well. At first they were recognizable as their different file format, which is actually not called **SQS** but **SQF**. So it actually hasn't changed a lot with Armed Assault®. The classic function with the file format **SQF** still exists with the file format called **SQS** used in OFP.

As explained in **Chapter 2.7**, so the function with the file format **SQF** is basically an enhancement of the faulty script Syntax called **SQS**. But it's needed to tell that it isn't replacing the scripts at all. The **SQS** Syntax still exists where the **SQF** Syntax is founded on.

BIS has adjusted the possibilities of functions and scripts for Armed Assault®. So a lot of new script-commands and amendments now exist for the Control Structures. So known **SQF**-Syntax Variants from OFP have clearly been advanced. And this is where it starts to get confusing.

Some of the new commands expect a special kind of syntax as some of the old ones do too. If you are using the Exec command you must be using the OFP script syntax. Labels and **Goto** loops can further be used and the script can be exited every time just by using the command **exit**.

With new **SQF** scripts each statement has to be finished with a semicolon **;** and while the usage of code no more in quotes **" "** but curly brackets **{ }**. The quotes are being used now by the strings.

OFP Example:

```
"_x addWeapon ""Binocular"" " foreach units group player
```

ArmA Example:

```
{_x addWeapon "Binocular"} foreach units group player;
```

Functions don't contain no delays no labels and no Goto loops, but **while-do** and **if-then-else** constructions. ArmA® also brought some new commands like **Switch**, **Do** or **Case**. Those command Constructions are also called **Control Structures**. So at this point I refer to the official Wiki of BIS.

http://community.bistudio.com/wiki/Control_Structures

A function has the job to effect a solution quick and efficient, so it consist regulary out of a short control sequence, while a script can be much more longer depending of its usage. Functions are often called out of scripts to allocate a value to a variable, evaluate values respective files or just calculate somethings. An example would be:

- recalculating the distance between 2 persons,
- pick out all Group leader out of a mass of units,
- find out the most known enemy unit,
- allocate special weapons to special units,
- search special elements of an Array or change it randomly

It happens quite often that a function is resending a value back to the script. Explained more accurate, many functions has especially been written, to give a value (the also called Return) back to the script. That doesn't work with a script. Functions are against scripts nearly always usable independently of Add-ons or Missions. Once they have been feed with information so they will deliver their results immediatly while many scripts were made for a special reason only.

Functions are offering the advantage that they can be preloaded while the mission is loading. They will be saved within the Engine so that they can get called so far they are needed.

They are much moiré faster available than a script and will be handled by the engine with a much higher priority. Ideally functions are to be load from out an init.sqs or Init.sqf.

If one wants to preload the functions while the mission is loading so they need to be named with a variable name at first. That enables one to call these functions again later.

Then the System knows:

NameOfVariable = Result of the preloaded SQF

The Syntax to load the functions:

SearchLight = compile preprocessFile "Searchlight.sqf";

SearchLight is the Variable which was used while the function was called (as explained above) so **SearchLight** stands now for the solution respective the whole content of the functions...or better...

SearchLight is now the function!

It's important to make sure that the variable names (i.e.: SearchLight) are not similar with command names which are already used in Arma®. In example for a position-function:

```
Position = compile preprocessFile "Position.sqf";
```

That would affect an error message right now, because the word Position is an Arma® command name. So it's really quite important to select the names carefully!

If one wants to recall his functions later in the mission, so the **Call-Command** is always to be used.

```
_variable = [Data] call SearchLight
```

Because this function has been preloaded resp. precompiled so they can be used immediately if its needed. The advantage against the script is that functions which have been precompiled mustn't get interpreted line by line.

But its also possible to get some duties of functions done which are actually defined as a script. That's why it's usual as well to use the known **While-Do** or **If-Then-Else** aspects. But mostly it's clearer to hand over these duties to a function to keep them out. In this case functions have to be called by the command **Call**. That enables one to go on with the results of the functions right in the next line of the script.

As I already explained so the SQF-Syntax for Arma® has been expanded a lot and was advanced with many more possibilities. It's possible now to use it for scripts which will receive the file format SQF of course. Within the scripts, as usual in Arma®, code blocks are still defined within curly brackets {} and statements will still be finished with a semicolon ;. It's further possible now to define time delay within SQF scripts, but in this case not by using the already known tilde key ~ but the command **Sleep**. Scripts which are used by the SQF file format have to be called by using the **Spawn** or even **execVM** commands.

If a script is intend to be used several times, so it's recommended to preload this script as a function by loading it into the RAM to become it executed much faster.

Preload scripts:

```
ExampleScript = compile preprocessfile "ExampleScript.sqf";
```

Call scripts:

```
[Data] spawn ExampleScript;
```

The Command **execVM** can be used for scripts which will be called only sometimes. That's why the script will not get preloaded.

```
[Data] execVM "ExampleScript.sqf";
```

If a script has to get called right out of the Editor, so its unfortunately needed to change the script call a little bit. It looks similar to a function call although the script will not give any values back.

Call a script out of the Editor (precompiled):

```
Variable = [Data] spawn ExampleScript;
```

or (not precompiled):

```
Variable = [Data] execVM "ExampleScript.sqf";
```

The word **Variable** is just a place keeper which is currently avoiding error messages. One can see the **Variable** as a **dummy** and also name it as well. But one can also just cut it by using **d**.

```
d = [Data] execVM "ExampleScript.sqf";
```

Scripts which are defined within the SQF file format are just ending so far the interpreter has coming to its end. The exit command is not only unnecessary but also improper, cause this is just a command to be used in SQS file formats scripts only. But should the case become true that its needed to cut off an SQF file format script so following construct is to use:

```
if (ConditionOfAbort) exitWith {};
```

But it's still possible to become code executed which is defined within the curly brackets right behind the **exitWith** Command. Example:

```
if (ConditionOfAbort) exitWith {Player sideChat "Finish!"};
```

Chapter 10

- Dialogs und Resources -

This chapter will introduce you to the most important parts of dialogs. Because my knowledge is limited on dialogs, I will explain to you only the most necessary things to offer you the ability to implement an image, a text, or even a small video as a highlight to your mission. All contents are expandable of course. So the following 11 sub-chapters will provide you a little basic knowledge to pimp up your mission.

To work with dialogs, basic knowledge dedicated to the Description.ext and how it's handled is needed. So you need to have patience and be smooth while working with the Description.ext. To forget a semicolon or a bracket always creates a horrible effect even for professional users, so CTD's (Crash to desktops) might happen often if the user doesn't take enough care about it. Don't give up if it doesn't work within the first few tries.

10.1	What actually is a dialog?	279
10.2	Base definitions (constants)	280
10.3	Basic classes and subclasses	283
10.4	The font styles	286
10.5	Fading in a graphic	287
10.6	Fading in a text	288
10.7	Fading scope views	289
10.8	An own tactical map	291
10.9	Defining a button	292
10.10	Defining a Frame	294
10.11	The video sequence	297



10.1 - What actually is a dialog

Dialogs are images, text or buttons which can be freely defined by the user for a mission or even for mission features as well. It enables the user to get graphics, text or additional scope views displayed on the screen while the mission or a sequence is running. It'll become much more interesting when it comes time to create buttons or drop down menus which shall also cause some things to happen.

The Editor is a good example to show the possibilities of dialogs, because nearly the whole Editor is consisted of dialogs. The Windows surface is a good example as well. The basics are the same as used in Arma®.

The image below is showing a drop down menu which is used within the Editor to select and insert groups on the map

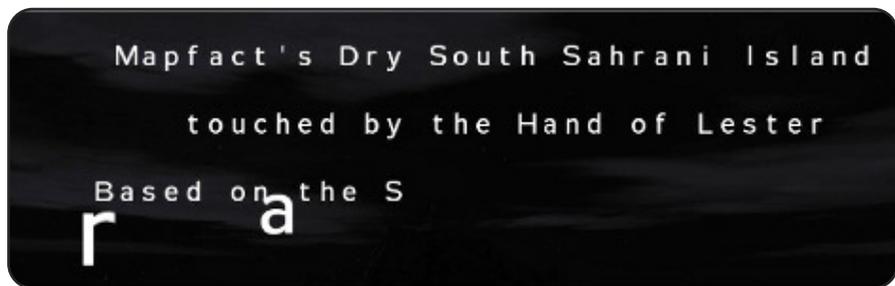


The possibilities have been expanded since Operation Flashpoint®. So it's possible now to create and define dialogs that are much more appealing. For example, the use of color, shadows or even flying letters as used in the Armed Assault® Credits.

Dialogs are a very useful and very nice tool to give some more features to a mission, but they're also hard to create, so this chapter will introduce you a little to the dialogs and by watching the video sequence you will also get a small overview about what is possible and what is not.

It's very important to work disciplined and accurately while using dialogs. These are defined within the Description.ext and only a small missing thing may cause a crash to desktop (i.e.: caused by missing or misplacing a semicolon or a bracket).

The font has been defined to be flying on the image below.



10.2 - Basis definitions (constants)

It's not possible to avoid creating some Basic Definitions. Here are some which are already used and hidden deep inside the Arma® engine. But some times these can be redefined for your own requirements to be further saved under a new name.

In some cases it's not needed to transfer all definitions into the Description.ext and as long the user is satisfied with the basic definitions, nothing further is needed. See fading in graphics or fonts, you'll note that no definitions were made there. But to be safer than safe, it's possible to transfer all definitions into the Description.ext to avoid a CTD (Crash to desktop) or something like this.

But what are Defines actually? Well, defines are just a kind of place keeper. There're values already defined by the game engine which are covering every Define individually. As long as these defines are used for their respective reason, it's not necessary to define your own. But if one wants to define his own font style, it's really possible. An example: The values true and false can also be defined as **1** for **true** and **0** for **false**.

The following definition should be existing in the Arma® engine:

```
#define false 0
#define true 1
```

These definitions are also to be seen in the unpacked Mission.pbo in Armory.intro and then within rscCommon.hpp. A variety of things can be done with just these two defines.

Define actually means to define something. So one is defining something as he wants to do. So that could be something like this...

Booleans	- Basic value like true and false
Sounds	- Sounds for Buttons etc.
Elements	- Frames, Surfaces, Buttons, List boxes aso.
Fonts	- The font styles
Colors	- The colors

...or even much more. One has many possibilities for his own definitions, which should be limited sometimes of course.

It's easier to understand when it says to assign a word for a value, because it's easier to keep a word then a sign in mind. An example: A definition of a users font color. Colors like black and white are not as bad to keep in mind as colored values. But if one wants to get a special blue or a special red color, it's easier to create your own definition.

As one can see the values of black and white are pretty simple:

```
#define Color_White {1, 1, 1, 1}
#define Color_Black {0, 0, 0, 1}
```

But it is different to another color, i.e. blue:

```
#define My_Blue {0, 0, 0.7, 1}
```

Excursion

Colors will be mixed out of values. All values between **0** and **1** are to be used only. The last value represents the transparency

```
{red, green, blue, Transparent}
```

End of Excursion!

Define-Name

As one can see in the Array (define) of the color blue above, a custom name has been given as a define name (**My_Blue**). So it's possible to use custom names. But it's generally recommended to keep the default names but to create whole new (your own) ones later. That might help avoid getting in trouble while using these definitions later in the mission.

The following is a very small list of some default defines including their names and values:

```
#define CT_STATIC          0
#define CT_BUTTON         1
#define CT_EDIT           2
#define CT_SLIDER         3
#define ST_TITLE_BAR      32
#define ST_PICTURE       48
```

It doesn't make much sense to use these names now. So one should select his own names to avoid having overlaps.

```
#define Murray_Title      ST_TITLE_BAR + ST_CENTER
#define Murray_Font       "Zeppelin32"
#define Murray_Sound      {"\ca\ui\data\sound\mouse2", 0.09, 1}
```

A custom name has been allocated in the first example. This one got the values **ST_Title_Bar** and **ST_Center** allocated which is also possible.

So I will stop here right now to explain the defines! One knows now that it's possible to use custom names and values and one also knows how to create custom colors. An example of some default defines can be seen on the next page which can be found again in many official missions.

Default Defines

The following is a selection of default defines out of Armed Assault®. It's recommended to extract the Description.ext from some a official ArmA-Mission to get the defines right from there. If one has already got template, it can always be used again.

```
// Control Types
#define CT_STATIC 0
#define CT_BUTTON 1
#define CT_EDIT 2
#define CT_SLIDER 3
#define CT_COMBO 4
#define CT_LISTBOX 5
#define CT_TOOLBOX 6
#define CT_CHECKBOXES 7
#define CT_PROGRESS 8
#define CT_HTML 9
#define CT_STATIC_SKEW 10
#define CT_ACTIVETEXT 11
#define CT_TREE 12
#define CT_STRUCTURED_TEXT 13
#define CT_CONTEXT_MENU 14
#define CT_CONTROLS_GROUP 15
#define CT_XKEYDESC 40
#define CT_XBUTTON 41
#define CT_XLISTBOX 42
#define CT_XSLIDER 43
#define CT_XCOMBO 44
#define CT_ANIMATED_TEXTURE 45
#define CT_OBJECT 80
#define CT_OBJECT_ZOOM 81
#define CT_OBJECT_CONTAINER 82
#define CT_OBJECT_CONT_ANIM 83
#define CT_LINEBREAK 98
#define CT_USER 99
#define CT_MAP 100
#define CT_MAP_MAIN 101
```

```
// Static Styles
#define ST_POS 0x0F
#define ST_HPOS 0x03
#define ST_VPOS 0x0C
#define ST_LEFT 0x00
#define ST_RIGHT 0x01
#define ST_CENTER 0x02
#define ST_DOWN 0x04
#define ST_UP 0x08
#define ST_VCENTER 0x0c
#define ST_TYPE 0xF0

#define ST_SINGLE 0
#define ST_MULTI 16
#define ST_TITLE_BAR 32
#define ST_PICTURE 48
#define ST_FRAME 64
#define ST_BACKGROUND 80
#define ST_GROUP_BOX 96
#define ST_GROUP_BOX2 112
#define ST_HUD_BACKGROUND 128
#define ST_TILE_PICTURE 144
#define ST_WITH_RECT 160
#define ST_LINE 176

#define FontM "Zeppelin32"

//My Defines
#define Murray_Title 32 + 64
#define Murray_Font "TahomaB"
```

Both tables above are actually representing one single table. But this is unfortunately not possible to handle right here, so I separated them into two images.

At first there are the **Control Types** and then the **Static Types** listed. But the order is not very important. As one can see there is a basic kind of orderliness. A special font style has been defined at the end by using **FontM**, so self created defines can simply be added to this one, so that makes it easier to see where the default defines are ending and the self created defines are beginning

One needs basic classes and subclasses for opening new dialogs or to use resources. The basic things are defined within the basic class while the specific things are defined within the subclasses. The Sound classes shall serve here as a parallel example which has to be defined within the Description.ext as well.

```
class CfgSounds
{
    sounds[] = {
        Sound1
    };

    class Sound1
    {
        name = "Sound1";
        sound[] = {"sounds\sound1.ogg", db+0, 1.0};
        titles[] = {};
    };
};
```

The basic classes and subclasses are present here as well. The basic class **CfgSounds** is defining that it's about a sound, while the subclass (**Sound1**) contains the respective definitions for **Sound1**.

And this is exactly the same for the classes which are needed for dialogs and resources. There are many more possibilities for dialogs and resources than there are for sounds.

Basic Class

Now the following is a basic class which contains basic things like types, styles, backgrounds, font colors, font styles and font sizes. This basic class RscText can now be used as a kind of base for all following subclasses.

```
class RscText
{
    type = CT_STATIC;
    idc = -1;
    style = ST_CENTER;
    colorBackground[] = {0, 0, 0, 0}; // Background Color
    colorText[] = {0, 0, 0, 0}; // Text Color
    font = "TahomaB"; // Typeface
    sizeEx = 0.080; // Font Size
};
```

Once the base was founded, all additional subclasses can be adjusted individually. It's possible now to define a larger and different colored font style for **Font1** then used for **Font2**, or define a different position on the screen.

Subclass

The subclass itself is separating now again. One can say that she has a further subclass inside. To avoid confusion we call it a component. The following example will show a structure.

```
// Main Class
class RscText
{

};

// Subclass
class RscTitles
{
    [List of Ressources]

    class MyRessource
    {
        [List of Components]

        class MyComponents
        {

        };
    };
};
```

One has to take care about the order of curly brackets, which are providing a better overview. So one can always see what belongs to what. The following example, also used in **Chapter 10.6**, explains how to define a script resource. One can see here the defined basic class `RscText` and the lower located subclass **Font1**.

Description.ext

```
// Basis Class
class RscText
{
    type = CT_STATIC;
    idc = -1;
    style = ST_CENTER;
    h = 0.05;
    colorBackground[] = {0, 0, 0, 1}; // Background Color
    colorText[] = {0, 0, 0, 1}; // Text Color
    font = "TahomaB"; // Typeface
    sizeEx = 0.040; // Font Size
};

// Next Side
```

```
// Subclass
class RscTitles{Titles[]={"Font1"};    // List of Ressources: "Font1, Font2,.. ."
    class Font1
    {
        Idd= -1;
        MovingEnable=False;
        Duration=10;                                // Fade Duration
        FadeIn=1;                                    // Fade Time
        Name=" Schrift1";                            // Name
        Controls[]={"FontButton1"};              // List

        class FontButton1 : RscText
        {
            ColorText[]={1,1,1,1};                 // Font Color
            Font="Bitstream";                       // Typeface
            Text="OPERATION SNAKEBITE";             // Text
            x = 0.30;                               // X-Axis
            y = 0.50;                               // Y-Axis
            w = 0.50;                               // Window Width
            h = 0.05;                               // Window Height

        };
    };
};
```

As one can see in the basic class, **RscText {0,0,0,1}** (black) has been used as font color. But as shown on the image below, the font color is white. That's why the color white **{1,1,1,1}** has been defined in the subclass **Fontbutton1** for **Font1**. But there are many more things defined in the subclass as one can see.

A very important line is:

```
class FontButton1 : RscText
```

One can see now that the definition of **RscText** will be allocated to the class **Scriptbutton1**. So everything which was predefined in the basic class **RscText** will be used here and could actually get lost within the subclass without s problem.

The font color and the font style were redefined for **Fontbutton1** in the example above. So that's why the lines were shown there.

So this image will now be called as follows:



OPERATION SNAKEBITE - CASTLEFIGHT

```
TitleRsc ["Font1", "PLAIN"];
```

10.4 – The font styles

Some font styles are already predefined in Armed Assault®, which can be selected by the user depending on his or her plans. One has the possibility to define one of these font styles in the Defines or in the respective class resp. subclass

The list below will show a little selection of the existing fontstyles:

Arial Bold
Bitstream
TahomaB
Zeppelin33
Zeppelin33Italic
Zeppelin32

Here are two examples:

TahomaB

OPERATION SNAKEBITE - CASTLEFIGHT

Zeppelin33Italic

OPERATION SNAKEBITE - CASTLEFIGHT

The font style **Bitstream** with a size of **0.04** will be allocated to the resource RscText in the following Description.ext example.

```
class RscText
{
    idc = -1;
    type = CT_STATIC;
    style = ST_LEFT;
    colorText[] = {1, 1, 1, 1};
    colorBackground[] = {0, 0, 0, 0};
    font = Bitstream;
    sizeEx = 0.04;
};
```

10.5 – Fading in a graphic

To get a simple graphic faded in, a definition of Defines within the Description.ext is not quite needed and can get lost. The only thing which has to be defined in the Description.ext is a class that can be used by the engine.

A good example is a graphic which will get faded in right at the beginning of a mission or something similar. One only needs to define a simple class for the image within the Description.ext. The example below is showing all what is needed to make it work.

The call will be done by:

```
titleRsc ["Bild1", "PLAIN"];
```

Description.ext

```
class RscPicture
{
    idc = -1;
    type = CT_STATIC;
    style = ST_PICTURE;
    colorBackground[] = {0, 0, 0, 0};
    colorText[] = {1, 1, 1, 1};
    font = Zeppelin32;
    sizeEx = 0;
};

class RscTitles
{
    titles[] = {Picture1};

    class Picture1
    {
        idd = -1;
        movingEnable = true;
        duration = 10; // Fading Duration
        fadein = 2; // Fade Time
        name = "Picture1"; // Name in Editor
        controls[] = {Picture};
        class Picture : RscPicture
        {
            x = 0.30; // X-Axis
            y = 0.50; // Y-Axis
            w = 0.40; // Window Width
            h = 0.05; // Window Height
            text = "pics\picture1.paa"; // Graphic Direction
            sizeEx = 0.04;
            style = 48;
        };
    };
};
```

10.6 – Fading in a text

The definition of custom text can be much more complicated than getting images faded in. Because additional things like font styles, font size, font color, and the position need to be defined as well, to list the most important ones.

The basic class has to be defined first, where all the basics will get configured. A custom subclass is needed each time new text appears on the screen and one can also adjust each subclass individually. For the first subclass (Script1) a headline with larger and more colored letters could get defined and for the second subclass (script2) a smaller and white colored font style could get defined. The needed syntax is:

titleRsc ["Font1", "Plain"]

Description.ext

```
class RscText
{
    type = 0;
    idc = -1;
    style = 0;
    h = 0.05;
    colorBackground[] = {0, 0, 0, 0};           // Background Color
    colorText[] = {0, 0, 0, 0};                // Text Color
    font =TahomaB;                             // Typeface
    sizeEx = 0.080;                            // Font Size
};

class RscTitles{Titles[]={ "Font1" };

    class Font1
    {
        Idd=-1;
        MovingEnable=0;
        Duration=10;                            // Fade Duration
        FadeIn=1;                               // Fade Time
        Name=" Font1";                          // Name
        Controls[]={ "Control01" };

        class Control01: RscText
        {
            ColorText[]={1,1,1,1};            //Text Color
            Font="TahomaB";                   // Typeface
            Size=0;
            Text="OPERATION SNAKEBITE";       // Text
            x = 0.30;                          // X-Axis
            y = 0.50;                          // Y-Axis
            w = 0.40;                          // Window Width
            h = 0.1;                          // Window Height
        };
    };
};
```

10.7 – Fading scope views

The regular scope views of the weapons can be used for sequences like intros, outros and others, but the binocular view is the most simple to create, the only thing which is needed is the following syntax:

cutRsc ["Binocular", "PLAIN",0.1] or **titleRsc** ["Binocular", "PLAIN",0.1]

All the other scope views need to be predefined with in the Description.ext. Once the definition has been made, one has the possibility to select this resource just by selecting the respective entry in the **Effects** category of a **Trigger** under Type/Resources. Ideally a script will work also.

The list below will show a small selection of the usable scopes with their respective source path. But be careful, some weapons are using the same scope. That's why not all weapons have been listed right here, but a small selection only.

Scope Types

Weapon	Source
G36	\ca\Weapons\G36_optics
M4SPR	\ca\weapons\optika_sniperw
M4	\ca\Weapons\optika_ACOG
M4GL	\ca\Weapons\optika_ACOG
SVD	\ca\weapons\optika_snpiere
Javeline	\ca\Weapons\optika_TOW
Binocular	\ca\weapons\optika_dalekohled
Laserdesignator	\ca\weapons\optika_SOFLAM
NVGoggles	\ca\weapons\optika_night
M119	\ca\weapons\optika_M119
Stryker	\ca\Wheeled\optika_stryker_driver \ca\Tracked\optika_stryker_gunner
BRDM	\ca\wheeled\optika_BRDM
AH1ZGunner	\ca\air\optika_heli_gunner \ca\air\optika_AH1Z
Ka50	\ca\air\optika_Ka50_rocket.p3d
Tankdriver	\ca\Tracked\optika_tank_driver
Tankgunner	\ca\Tracked\optika_tank_gunner
M1A1	\ca\Tracked\optika_M1A1_commander
ZSU Gunner	\ca\Tracked\optika_zsu_gunner
ZSUCommander	\ca\Tracked\optika_tanke_auxiliary
T72Gunner	\ca\Tracked\optika_T72_gunner

Description.ext

```
#define CT_OBJECT            80
class RscObject
{
    type = CT_OBJECT;
    scale = 1.0;
    direction[] = {0, 0, 1};
    up[] = {0, 1, 0};
};
class RscTitles
{
    titles[] = {SNIPER};
    class SNIPER
    {
        idd=-1;
        movingEnable = false;
        duration=10;
        name = "SNIPER";
        objects[] = {SNIPER};
        class SNIPER : RscObject
        {
            model= "\ca\weapons\optika_SOFLAM";
            idc=-1;
            position[] = {0,0,0.065};
            direction[] = {sin 0, sin 0, cos 180};
            up[] = {0, 1, 0};
        };
    };
};
```

The defined resource called SNIPER will now get called as follows:

titleRsc ["SNIPER", "Plain"]



If one wants to fade out this resource again later, up to the selection it works by using:

titleRsc ["default", "plain",2] or **cutRsc ["default", "plain",2]**

10.8 – An own tactical map

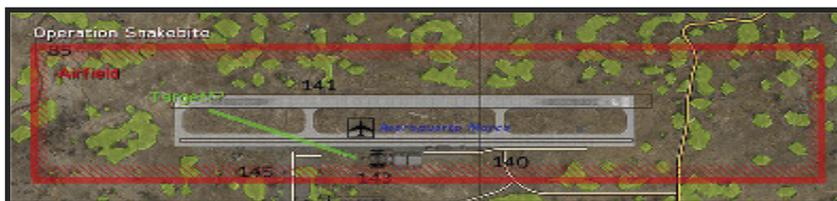
The possibility to insert custom images offers many possibilities. A tactical map, secret plans, a mug shot or much more than a simple logo could get realized now. The following example will deal with the tactical map. The way to get it called should be up to the liking of the user, whether it's a radio trigger, an action menu entry or whatever. But the syntax to call the effect has to be there:

```
titleRsc ["MyTacticalMap", "PLAIN"];
```

Description.ext

```
class RscPicture
{
    type = CT_STATIC;
    idc = -1;
    style = ST_PICTURE;
    colorBackground[] = {0, 0, 0, 0};
    colorText[] = {1, 1, 1, 1};
    font = Zeppelin32;                                     //Typeface
    sizeEx = 0;
};
class RscTitles{titles[] = {MyTacticalMap};
    class MyTacticalMap
    {
        idd=-1;
        movingEnable = true;
        duration=10;                                       // Fade Duration
        fadein=2;                                          // Fade Time
        name = "MyTacticalMap";                            // Name in Editor
        controls[]={Picture};
    }
    class Picture : RscPicture
    {
        x = 0.10;                                         // X-Axis
        y = 0.40;                                         // Y-Axis
        w = 0.80;                                         // Window Width
        h = 0.05;                                         // Window Height
        text = "pics\map.jpg";                             // Graphic Direction
        sizeEx = 0.04;
        style=48;
    };
};
```

As one can see this Description.ext is exactly the same as was used in **Chapter 10.5**, but only the values have been changed.



10.9 – Defining a button

To define a button is much more complicated than adding a simple image or some text. The Defines are definitely needed to be used again here and much more as well as you will note in a few moments. One has unlimited possibilities. But this example is focusing on a few basics, respectively, one simple default button only.

Description.ext

```
#define CT_BUTTON          1
#define Color_White       {1, 1, 1, 1}
#define Color_Black       {0, 0, 0, 1}
#define Color_Grey        {0.5, 0.5, 0.5, 1}
#define FontHTML          "BitStream"

class RscButton
{
    type = CT_BUTTON;
    style = ST_CENTER;

    // Text Definitions
    font = FontHTML;
    colorText[] = Color_White;
    colorDisabled[] = Color_Black;

    // Background Configuration
    colorBackground[] = Color_Black;
    colorBackgroundDisabled[] = Color_Black;
    colorBackgroundActive[] = Color_Grey;

    offsetX = 0.001;
    offsetY = 0.002;
    offsetPressedX = 0.003;
    offsetPressedY = -0.003;
    colorFocused[] = Color_Black;

    // Shadow Configuration
    colorShadow[] = Color_Black;

    // Rahmen Configuration
    colorBorder[] = Color_White;
    borderSize = 0.02;

    // Sounds
    soundEnter[] = {"\ca\ui\sound\pageopen", 0.1, 1};
    soundPush[] = {"\ca\ui\sound\pageclose", 0.1, 1};
    soundClick[] = {"\ca\ui\sound\offbutton", 0.1, 1};
    soundEscape[] = {"\ca\ui\sound\offbutton", 0.1, 1};
};
```

This script will go on the next page. The frame data for the buttons have been defined within the first part of the Description.ext. The second part will handle the button itself.

```

class DemoButton
{
    idd = 100;
    movingEnable = false;
    controls[] = { My_BUTTON };

    class My_BUTTON : RscButton
    {
        idc = 100;
        sizeEx = 0.018;
        text = "Click Me!";
        // Position
        x = 0.4;
        y = 0.4;
        // Size
        w = 0.15;
        h = 0.04;
        // Action
        action = "ctrlSetText [100, ""Thanks!""; hint ""TEST"" ";
    };
};

```

The button has been defined here right now. He will receive the frame data from the predefined **RscButton** which is located in the first part of the Description.ext. This button can get called now simply by using this syntax:

```
ok = createDialog "DemoButton"
```

This syntax can be used everywhere to call the button. A simple radio trigger is a good and easy way to test the button.

The Action line could look different. The reason why it was shown that way is to demonstrate that the text of the button can change when it gets clicked. By default the commands are used there which has to be executed by using this button.

A further way about how the action line could look like:

```
action = "closeDialog 0; [] exec ""script.sqs"" ";
```

As one can see so the action line is pretty much freely definable. It enables one to get things caused or something else. But it's important to take care about things like brackets, signs and quotes, etc.

The command **closeDialog 0** will finally close the dialog

Now it's possible to decorate this button with a additional frame or much more. This should just be a simple example to an explain how to create a button.

10.10 – Defining a frame

The following example will deal with the way about how to create a simple Frame. This Frame will include a special feature which is represented by a small text. But at first the Defines have to be defined again within the head which will be reordered later.

Description.ext

```
#define CT_STATIC          0
#define Color_White      {1, 1, 1, 1}
#define Color_Black      {0, 0, 0, 1}
#define FontHTML         "BitStream"
#define ST_FRAME         64

class RscText
{
    Idc = -1;
    Type = CT_Static;
    Font = FontHTML;
    ColorBackground = Color_White;
};

class My_Frame : RscText
{
    Style = ST_FRAME;
    x = 0.22;
    y = 0.12;
    w = 0.40;
    h = 0.20;
};

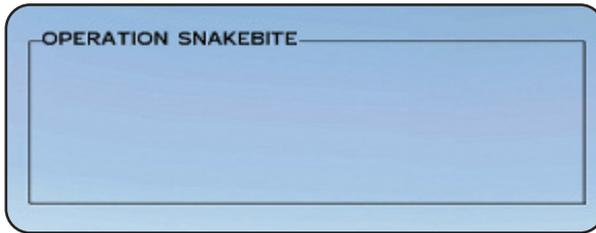
class RscTitles {Titles [] = {"Frame1"};

    class Frame1
    {
        Idc = -1;
        MovingEnable = False;
        Duration = 10;                // Fade Duration
        FadeIn = 1;                   // Fade Time
        Name = "Frame1";            // Name
        Controls[] = {"FontFrame1"}; // List

        class FontFrame1 : MY_Frame
        {
            ColorText[] = Color_Black; //Font Color
            Text = "OPERATION SNAKEBITE"; // Text
            SizeEx = 0.030;
        };
    };
};
```

In game it will look like this. The Frame will get called by using:

```
titleRsc ["Frame1", "PLAIN"];
```



10.11 – The video sequence

The final highlight to this dialog chapter shall be the way to create your own video sequences. Such a sequence will get created by using a bunch of single frames, so lot of single images and a small script. But the Description.ext has to be prepared of course, like always while working with dialogs.

Description.ext

```
class RscPicture
{
    Idc = -1;
    Type = 0;
    Style = 48;
    ColorBackground[] = {0, 0, 0, 0};
    ColorText[] = {1, 1, 1, 1};
    Font = BitStream;
    SizeEx = 0;
};

class Videodialog
{
    Idd = -1;
    MovingEnable = true;

    Controls[] = {Picture_1};

    class Picture_1 : RscPicture
    {
        idc = 200; // The ID
        x = 0.0; y = 0.0; w = 1.0; h = 1.0; // Size and Position
        text = "video\frame1.jpg"; // Source
        size = 0;
    };
};
```

Once the Description.ext has been defined, a script is still needed which will call and control the sequence. The script looks as follows:

Video.sqs:

```
playMusic "ATrack8"
TitleCut [" ", "BLACK IN"]; titleFadeOut 4
_video = createdialog "Videodialog"
_x = 1
_delay = 1/24

#Loop
ctrlSetText [200, format ["video\frame%1.jpg", _x]]
~_delay
_x = _x + 1
? _x < 150 : goto "Loop"

closedialog 0

TitleCut [" ", "BLACK IN"]; titleFadeOut 4
exit;
```

As one can see some effects were used here as well. So the sequence will fade in slowly because of the **Titlecut** definition. Additionally to this, music will be played also. The video dialog will get created at the same time as the images are getting started. But for this, some additional explanations.

- _x** - Will be used for the implanted counter. So long as the value is smaller than 150, the loop will always be replicated. 150 is the number of used images, which are saved in the folder Video, but much more could be used also.
- _delay** - The speed of getting the images played can be adjusted here. If the value will be changed to **1/12**, the images will run slower and a stop motion picture effect appears. The value 1/48 makes the images run much faster.
- video\frame%1.jpg** - All the images are saved within the folder Video, which are named Frame1, Frame2,... In this example 150 images were used. So if one wants to use more, the script just has to be adjusted.

The images resp. frames

Once the scaffolding has been created, just the content is missing. As already mentioned, single images need to be used. The length of the sequences defines itself by the number of images. Special freeware tools can be downloaded which enables one to extract single images out of a whole movie. But I won't give some links or tell their names here for legal reasons.

Depending on the project the user wants to create, it might be that a ton of images will be used. In the example script around 150 images were used, although the sequence has a total length of a few seconds. So the user should really think about the reason why to use such a feature.

There's also the possibility given to resize the images again, so that only a small part of the screen is covered by the video sequence. For example, the Action hud or an info video or something similar.

But if one wants the whole screen to be covered by the video, the images also need to be saved in a respective resolution. For example, 512x512 Pixels. But make sure not use too big files. That will make the total size of the mission too huge at the end of the whole mission project.

Folder

It is recommended to create your own folder where all the images will be saved because saving all images right into your Mission root folder would be very unprofessional. In this case the folder was named **video**. This is the path:

video\frame%1.jpg

Finally, a small image which represents the named and numbered images.



The Engine is replacing the value **%1** with the next highest number. But it'll work only by using the correct syntax. The script would look like this (see left hand picture).

ctrlSetText [200, format ["video\frame%1.jpg", _x]]

Chapter 5.66 contains some self explaining syntaxes of this. That should provide enough to get a handle on this.

So now, just one thing is missing:

And Action!

Chapter 11

- General Informations -

This chapter, which is also the last one, will explain some of the general parts of the game. E.g. To create and use mod folders, the use of addons in your missions or what you should do before releasing your mission to the public and last but not least the Nato Alphabet which is always recommended to know as an Arma® soldier.

11.1	Own profile	299
11.2	The Arma cheats	301
11.3	The MOD-Folder	302
11.4	The use of addons	303
11.5	The missions release	304
11.6	The Arma.rpt	305
11.7	The NATO Alphabet	306
11.8	The ranks and their badges	307
11.9	The Squad.xml	308
11.10	The Start Parameters	311
11.11	Key combinations, tips and tricks	313



11.1 – Own profile

Your profile is where all stored game saves and the saved Editor Missions are all basically located.

C:\My Documents\ArmA\User\Missions

Armed Assault® automatically stores your user profile into the system drive where your windows operating system is located which is usually the C:/ Drive. This can be bad in some ways which I will explain below.

Armed Assault® automatically creates the necessary folders on your operating system drive. Unfortunately, if a virus or something else bad happens to your PC and nothing works anymore (worst case), Windows would have to be reinstalled. All the missions, game saves, ideas and all the hours of work would be gone; basically your whole profile.

So if ArmA® is not installed on the system partition you have the possibility to save the profile directory in the ArmA® folder of your choice. To do this a shortcut to the ArmA.exe is needed first. Then hit right mouse button and select the properties. Then just change the already given path to:

D:\ArmA\Arma.exe -profiles=D:\ArmA\ -name=Username

The path still needs to get modified to your settings for what ever you like best. Additional parameters could get used also:

- window** - Will run ArmA® in a Window only
- nosplash** - Will run ArmA® without the introduction screen (fast run)

So the modified line would look like this:

C:\ArmA\Arma.exe -nosplash -window -profiles=D:\ArmA\ -name=Mr-Murray

By using the above line the profile will get called from the profile folder in

D:\ Drive\ArmA Folder

and also the game would get started in windowed mode and without a splash screen.

Tip:

If you only want to mission edit, it's recommended to get ArmA® run in windowed mode, so it's easier and even faster to switch between the game and the scripts without the worry of a crash to desktop.

Username.ArmAProfile

The Profile folder contains a file called Username.ArmAProfile, where all settings of the Player is stored in. Therefore belongs the content listed below (the most important things only)

- Graphic settings (Brightness, Gamma, Shadow, Shading etc.)
- Sound settings
- Key combinations
- Mouse settings
- Last played SP Missions
- Last played MP Missions
- Last played Campaign Missions
- Selected Campaign
- Multiplayer settings (Filter, etc.)
- Degree of difficulty
- User defined degree of difficulty
- Own Identity (like in Description.ext)
- Active Missions keys
- Active Vehicle keys (armory)
- Blood (on/off)

This file can be opened quite simply with the text editor. So you have the possibility to change the settings right there and then.

ArmA.cfg

A further important file is the **ArmA.cfg**. Direct System settings are located in there, things like Screen resolution and RAM also.

winX=0;	- Screen Position X-Axis
winY=5;	- Screen Position Y-Axis
winW=1024;	- Screen resolution X-Axis
winH=746;	- Screen resolution Y-Axis
winDefW=1028;	- Screen resolution relative to X-Axis
winDefH=746;	- Screen resolution relative to Y-Axis
FSAA=2;	- Anti Aliasing
HDRPrecision=8;	- HDR
lastDeviceId="4318,661,107156578";	- contains all information
localVRAM=527429632;	- Graphics card RAM Store
nonlocalVRAM=260046847;	- Shader store in Ram allocated by Motherboard

11.2 - The ArmA-Cheats

The following list of cheats for Armed Assault® are not regular cheats. They are not really giving an advantage to the player, except for the save game cheat. But there are also nice features which can be useful in certain situations.

All listed cheats need a special way to become executed. To do this press and hold the **left Shift key** and hit the **minus key** on the **Num-Pad**, Then enter the code using the keyboard. If this process was successful a little hint will appear on the left upper corner which will confirm the code. It's important to know that no input line will appear to enter the Code, so you are just typing blind.

- CAMPAIGN** - This code allows you to unlock all the campaign missions without having to complete the campaign all the way through so you can select the missions you want to do. This code must be entered on the main menu screen.
- MISSIONS** - To unlock all of the single player missions without completing the first set type this code in on the main menu screen.
- ENDMISSION** - This code has to be used within a mission and will only work in single player. Its pretty self explanatory, once the code has been imputed the mission will end.
- SAVEGAME** - This code saves the current game status while playing a SP Mission only.
- TOPOGRAPHY** - This generates a Map in EMF-Vector Format and stores it the Drive C:\. This map will be generated if one is changing back to the map next time. Be careful while using American Keyboards. In this case enter **TOPOGRAPHZ**.
- FLUSH** - Flush refreshed all the Textures and Objects out of the RAM. So for e.g. when you start to get texture glitches you will use FLUSH to sort them out.

Unlock a Single player Mission

Single player Missions can get unlocked on the one or other way. But this requires the edit of the own ArmA-Profile. To do this just change within the own profile folder, open the file **Username.ArmAProfile** with **Text Editor** and enter the not yet saved keys in there.

That all should finally look this way:

```
activeKeys[]= { "M00", "M01", "M02", "M03", "M04", "M05", "M06", "M07", "M08",  
               "M09", "M10", "TT01", "TT02", "TT03", "TT04", "TT05", "TT06" };
```

Once this file will get closed and saved again, so these changes will take effect immediately.

11.3 -The MOD-Folder

The use of Mod Folders offers a better overview while using Mods and Addons. One has also the choice which Addons should get loaded as well and which ones not. Some Addons from the Community are containing corrupted Configs, which will get entered automatically within the Mission.sqm without really loading and using this Addon. Additionally to this, the loading time will increase unnecessarily if all addons are loaded. Also, most of them are not needed. That's why the smart ArMA® player is using Mod folders! Always keep in mind:

Never touch the Addons Folder in the Game directory!!

Mod folder can be used in different ways. A little incitation:

Related to the Mission **@MyMission**

Related to a Theme **@MAPFACT**

Related to a Mod **@ECP-MOD**

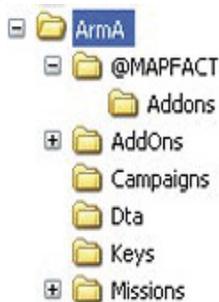
Those names are examples only, and one can define them as he wants to do, so they are variable. If a complete Mod should receive an own Mod Folder so it's recommended to use the predefined Name. But even in this case it's up to the player.

Now all Addons can get loaded right from this folder. A further advantage is that these Addons can also get offered now for a download to avoid the player having to search too much for those ones.

Now the player would have the choice, whether he wants to get this or that addon loaded with the game or not. If he doesn't want to use some special Addons, so these ones don't need to get loaded also. That saves time and performance.

The creation of a Mod Folder

The folder always needs to be created right in the main ArMA® directory. Give him a name and open it and create a new folder which has to be named Addons. All Addons (pbo files) have to be stored right in there. The next time ArMA® will get started one can select now the Addons which should get loaded to use in a Mission or what ever.



The image above shows a screenshot from a directory example. As you can see the @ sign was placed right before the Folder called Mapfact. That's the only way to make sure that the new folder will be placed at the very first position in the directory list. If all additional Mod folders will get renamed like this, so they will get placed right at the beginning of the list, which keeps the overview. It's also possible to use other special signs other than @. That's only an incitation

To load a Mod-Folder

If one or more Addons should get launched so there are several ways to do this. The default way is to use ArmA® shortcuts for your Addons which have an additional line within the command line. The other way is to use one of the available ArmA® launchers which can get downloaded on many Community sites. This way is of course the most elegant and more powerful solution. But now, here's the Parameter which is needed to complete the short cut. To add this Parameter to the shortcut just right click on it and add this part right behind the command line:

-mod=@MAPFACT

Combined with the command line it should look like this:

D:\Arma\arma.exe -mod=@MAPFACT

To avoid getting confused with several ArmA® shortcuts, so you can rename that one as you want. In case of the Mapfact Mod a possible name for a short cut would be ArmA-Mapfact.

11.4 - The use of addons

Addons are a nice feature, expanding ArmA® a lot and are giving a more interesting face to ArmA® for a long time. So it should be clear that Addons shouldn't be missed within self made Missions. But there's much to take care about which will get explained here right now.

If Addons are used within a mission watch out for those ones which are not to friendly to the performance. Many addon builders like to forget that fact and create addons with a lot of polygons and unnecessarily large textures for parts which are not even seen in game. Mostly it's possible to realize that with less polygons and a smaller texture size to save performance to the machine.

If all the correct Addons were selected editing can get started. But now take care not to use too many Addons and Mods for only one Mission. The player will mostly need to search a lot for all these Addons and also download them which often takes a lot of time and destroys the motivation to go on. After he has downloaded all that stuff very often follows the disappointment that the mission if it was bad or is just not really playable cause too many Addons were used.

So always keep in mind....less is more! Not the number of used Addons within a Mission is the important part, but more the realization, functionality, Story and of course the fun to play factor!

11.5 - The missions release

If the mission has finally been created with a mental brakedown and some more or less invested time, the publication is close. Finally we present our work to the public and also to get some good feedback. But there're also some things to take care about. It's not important whether the mission will be released today or tomorrow, it's better to use one further day to test the mission to receive a rational result than a half finished project.

Test, test, test

Your mission should get tested for functionality, playability and fun factor by you, your friends and your acquaintances.

Story and logic

The Mission should have a good Story and a logical course. The player always should know what happens next so they don't get lost and bored period. Make it keep it interesting.

The Readme

Each mission should include one readme. The readme should give a little bit of information about the mission, who created it and what scripts and addons were used in the mission. If possible working links should be in there as well for the addons.

Used Addons

Ideally a Mod folder would be a great additional feature. But that's up to the total size of all used Addons. Use links for bigger Addons.

The download file

At the end pack all the files into a rar or zip and make sure that these things are included:

the **Mission.pbo**, one **Readme** and if possible the **Mod-Folder**

If you have followed these points, then the mission can get released, and there should'nt be any barrier for the Mission to become successful.

So now have fun for your first Mission release!

11.6 - The ArmA.rpt

The ArmA.rpt is a Error report file of Armed Assault®. The Engine stores important information right here. I.e. if ArmA® causes a CTD again (Crash to Desktop), you can try to figure out why that's happened. The best reason for a CTD is mostly the Description.ext which causes errors where some users found themselves back on the Desktop and asking why that happens. Looking into the ArmA.rpt, sometimes helps to find the mistake.

This file will store among others errors of:

- Missions
- Scripts
- Functions
- Addons

That may be missing signs like quotes or wrong Textures, Models or even Config Mistakes. But the ArmA.rpt is not only a error report file, It can also be seen as a kind of Log file. It also stores information like:

- Status changes of Units and Objects
- Status changes of the player
- Multiplayer activities

And lots of more, so it's the optimal road sign for Errors and more

So if you are close to releasing a Mission, it would always be recommended to check the rpt whether any Errors are still existing, which are directly or indirectly visible.

It's further recommended deleting the ArmA.rpt from time to time, especially when it comes to figuring out a mistake. ArmA® will always recreate that file and locating errors in the rpt when its's not new can become messy and a lot of work.

The image below will give a short look inside the saved text.

```
No more slot to add connection at De61 (6952.9,8228.9)
Client: Object 8:57 (type UpdatePositionVehicle) not found.
*** Remote: Identity Mr-Murray transferred from 8:98 to 8:120
File update_v1\RscDefine.hpp, line 0: '!100': Missing ';' at the end of line
File update_v1\RscDefine.hpp, line 0: '!120': Missing ';' at the end of line
Updating base class ->Helicopter, by ca\air\config.bin\CfgVehicles\UH60MG/
```

The ArmA.rpt is located, when one is working with **-Profile** as explained in **Chapter 11.1**, usually in the ArmA® root directory, otherwise you can also find it in:

C:\Documents and Settings\User\AppData\Local\Arma

11.7 - The NATO Alphabet

A	Alpha
B	Bravo
C	Charlie
D	Delta
E	Echo
F	Foxtrott
G	Golf
H	Hotel
I	India
J	Julienn
K	Kilo
L	Lima
M	Mike
N	November
O	Oskar
P	Papa
Q	Quebec
R	Romeo
S	Sierra
T	Tango
U	Uniform
V	Viktor
W	Whisky
X	X-Ray
Y	Yankee
Z	Zulu

11.8 - The ranks

Now for a list of Ranks used by different NATO Forces. There're of course many different Ranks in different armies but that would be definitely too much right now.

Germany	USA	UK	France
Gefreiter	Private	Private	Première Classe
Obergefreiter	---	---	Caporal
Hauptgefreiter	Private First Class	Lance Corporal	Caporal Chef
Stabsgefreiter	---	---	---
Oberstabsgefreiter	---	---	---
Unteroffizier	Corporal	Corporal	Sergent
Stabsunteroffizier	---	---	Sergent Chef
Feldwebel	Sergeant	Sergeant	Adjudant
Oberfeldwebel	Staff Sergeant	Staff Sergeant	---
Hauptfeldwebel	Sergeant First Class	---	Adjudant Chef
Stabsfeldwebel	Master Sergeant	Warrant Officer Class 2	Major
Oberstabsfeldwebel	Sergeant Major	Warrant Officer Class 1	---
Fahnenjunker	---	---	---
Fähnrich	Cadet	Officer Cadet	Aspirant
Oberfähnrich	---	---	---
Leutnant	Second Lieutenant	Second Lieutenant	Sous Lieutenant
Oberleutnant	First Lieutenant	Lieutenant	Lieutenant
Hauptmann	Captain	Captain	Capitaine
Major	Major	Major	Commandant
Oberstleutnant	Lieutenant Colonel	Lieutenant Colonel	Lieutenant Colonel
Oberst	Colonel	Colonel	---
Brigadegeneral	Brigadier General	Brigadier	Général de Brigande
Generalmajor	Major General	Major General	Général de Division
Generalleutnant	Lieutenant General	Lieutenant General	Général de Corps d'Armee
General	General/Field Marshal	General/General of Army	Général d'Armee

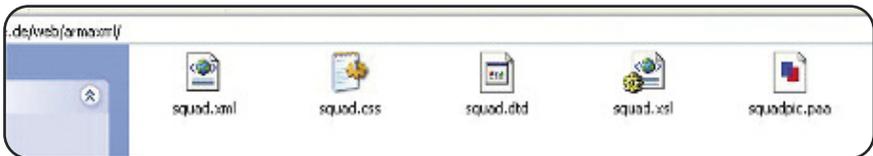


11.9 - The Squad.xml

The Squad.xml is basically made for becoming members of a clan which you can be identified on the Battlefield. This enables you to allocate your own logo to each member. So it makes sense when all members of a clan have the same logo (Tag). But even if you're not a member of a clan, it's possible to get a logo, just to get identified yourself. That logo will always get displayed on the right arm of a unit and any vehicle he boards, as shown in the image below.



To use the squad.xml, some things are required first. A webserver is needed, where all the following files can be transferred to. Just create a new directory on your server and name it as you want. Then modify the files (the ones shown below) and save them in that new directory.



Needed files

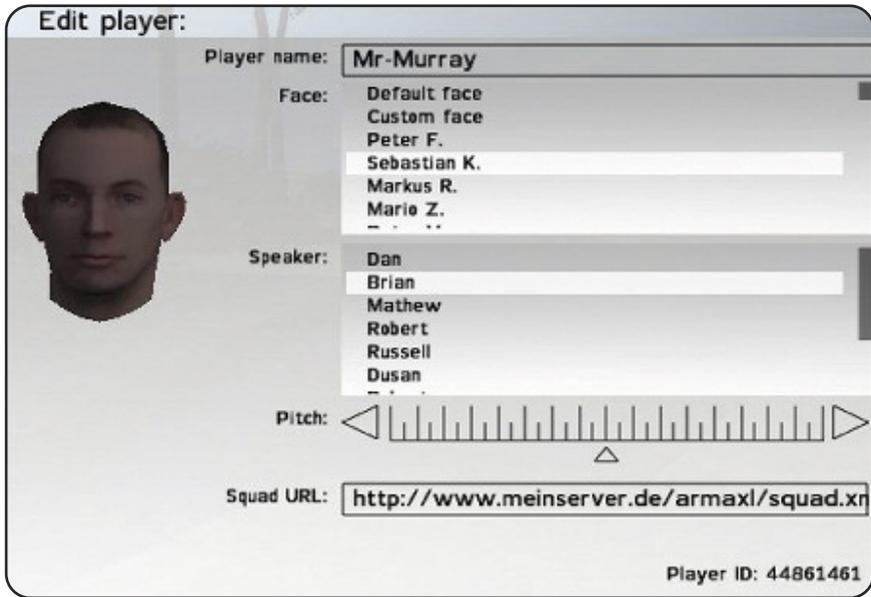
The needed files are:

squad.xml
squad.css
squad.dtd
squad.xsl
squadpic.paa

The URL needs to be noted to put it into a special part of the profile.

<http://www.myserver.de/armaxml/squad.xml>

This part is the line called Squad URL right in your profile.



Player ID

Each game/player has an individual **ID**, there's no 2 of the same ID's existing. And this ID is needed within the Squad.xml. Otherwise it won't work.

The Configuration

The only files which need to be modified is the **Squad.xml** and the **Squadpic.paa**. The other files don't need to be modified, but they are also very important for the xml to work and so they also need to be stored onto your webserver

Squad.xml

The actual configuration will be done in the squad.xml. This file is separated into 2 different parts. The upper part is defining the squad, or the clan, while the part below is only defining the player himself.

<Squad Nick>	- Squad shortcut (Clan-Tag)
<Name>	- Squad name
<EMail>	- Squad contact
<Web>	- Squad website
<Picture>	- Image (Squadpic.paa)
<Title>	- Squad name

It's important to make sure that this file definitely doesn't get edited with Excel but with the Text Editor only!

This part defines the player:

<Member ID>	- Player ID (see Profile image)
<Nick>	- Nick name (Profile name)
<Name>	- Name
<EMail>	- Player mail contact
<ICQ>	- IM-Contact
<Remark>	- Notification

When its finally edited the squad.xml looks like this:

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<!DOCTYPE squad SYSTEM "squad.dtd">
<?xml-stylesheet href="squad.xsl?" type="text/xsl"?>

<squad nick="MAP">
  <name>Mr-Murray</name>
  <email>mr-murray@bossmail.de</email>
  <web>www.mr-murray.de.vu</web>
  <picture>squadpic.paa</picture>
  <title>Mr-Murray</title>

  <member id="12345678" nick="Mr-Murray">
    <name>Mr-Murray</name>
    <email>mr-murray@bossmail.de</email>
    <icq>N/A</icq>
    <remark></remark>
  </member>
</squad>
```

Squadpic.paa

The `squadpic.paa` is the logo which will be displayed later in the profile, on the arm of the player's character and also on each vehicle the player boards as driver. But the logo is not needed, so you could leave it out as well. The name **Squadpic** is a variable name. It's only important to make sure that the image name and the definition in the `Squad.xml` are exactly the same.

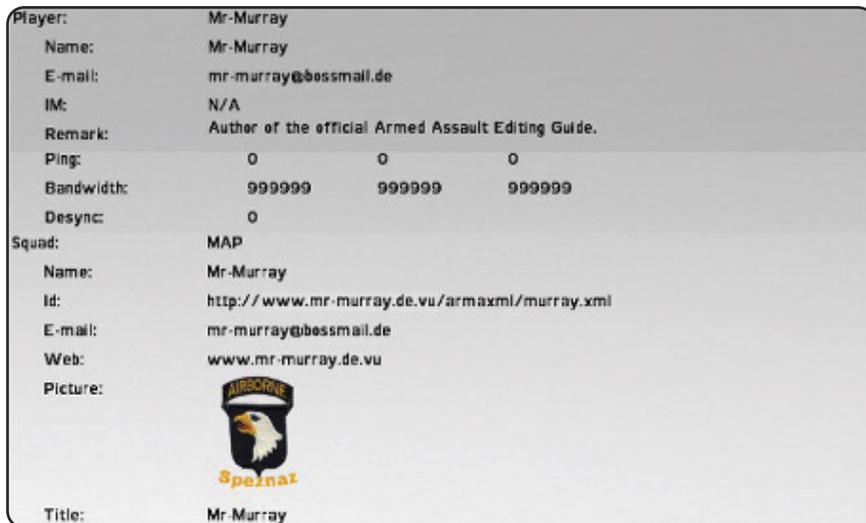
If you want you can create your own logo with a graphic program like Gimp or a Test version of Photoshop. Create a new image in TGA Format and define the size which must be like one of these:

16/16, 32/32, 64/64, 128/128, 256/256

Then create the logo how you want to and save it. In the next step this image needs to be saved in `paa` format or `PNG` format. To do this just get the BIS Tool `TextView2` and load your image right in there, then save in `paa/png` format, that's all. Make sure to use the same name as defined in the `squad.xml`.

Result

If you press the P-Key later in MP game, whether hosted or dedicated Mission, the following image will be displayed then.



Template

You can find a downloadable template at:

www.mr-murray.de.vu

But it would have to be adjusted to your needs.

11.10 - The Start Parameters

There are endless start-up parameters existing for Armed Assault®, which can be added to the ArmA.exe or just a shortcut of the ArmA.exe. You can get lots of advantages by using these parameters. A very good feature is the possibility to run ArmA® in windowed mode, or to skip the splash screen. The following list will explain the most interesting parameters, more can be found in the BIS Wiki..

Screen options

- window
- nosplash
- x=Value
- y=Value
- Runs ArmA® in windowed mode
- Runs ArmA® without a splash screen
- Resolution (width)
- Resolution (height)

11.11 – Key combinations, tips and tricks

As in each program, there are some key combinations needed to enable a faster editing so here are some tricks.

Key Combinations

- | | |
|---|--|
| [CTRL] and [C] | - Copy (Units/Objects) |
| [CTRL] and [X] | - Cut (Units/Objects) |
| [CTRL] and [V] | - Paste (Units/Objects) |
| [Delete] | - Deleting (Units/Objects) |
| [Shift] and [Delete] | - Deleting all marked units and objects |
| [Shift] and [left Mouse Button] | - Turning around (Units/Objects)
(press and hold key while moving the mouse) |
| [Shift] and [left Mouse Button] | - Edit unit with waypoint
(press F1, press and hold shift and press left mousebutton) |
| [Left Mouse Button] | - Move unit/object or mark area
(press and hold mousebutton and move the mouse) |
| [Left] and [right Mouse Button] | - Move over the map
(press and hold mousebutton and move mouse) |
| [Mouse Wheel] | - Map zoom in zoom out |

Templates

It can be necessary to create a template for your mission. So if you spend a lot of time building a position, you will then be able to use it again for other missions. If a rough template was created for every single mission feature you will have the possibility to get them imported quite fast again for later Missions. You can save a lot of time by working this way. To do this just create your feature and save it by using a regular editor name like Temp_Bunker. So you can load it when it's needed. Use the Key combos shown above to copy and paste the template from one Mission into the other.

Templates II

If you have created a huge mission it may happen that a great idea suddenly appears to add a further special feature. Maybe a special kind of fortress or what ever, But if you created that fortress in the mission when testing it you would have to preview the hole mission over and over which can be annoying.

So the creation of a Map (template) would be helpful right here. Just build your base or what ever on a new map and once it has been finished, just save it with regular mission name. Then go back to the main mission and use the merge function to merge it into the Main Mission.

Keyword Index

A

Description	Chapter	Page
Action command	5.55	136
Actionmenu entry	6.3	181
Airstrike	6.12	198
Airvehiclecreator	6.13	201
Alert	5.32	119
Animationcommand	5.56	139
Arma.rpt	11.6	305
Array	9.12	271
Artillery	6.7	189
Armament within Multiplayer	7.21	250
Adjusting date and time of day	5.43	123
Assigning ranks	5.76	171
Adding Units/vehicles	1.2	20
Arm and Equip Units	3.3	70
Assigning a target to a unit	5.21	113
Allocate a flag to a flag staff	5.35	120
Adjusting radio menu	5.48	127
Accurate helicopter landing	5.18	153
Adjust height of an Object	5.16	111
Adding Markers	1.7	36
Assigning respective display score in MP	7.9	231
Allocate a call-sign to a group	5.49	128
All about vehicles	5.71	165
Add or remove switchable units	5.37	121
Adjust distance of view	5.41	122
Adding Waypoints	1.5	30
Adjusting the weather	5.42	122
Assigning a target to a unit	5.21	113

B

Description	Chapter	Page
Briefing	2.13	59
Burning fire	5.36	121
Basic knowledge about Functions (SQF)	9.13	274
Building positions	5.62	148
Brackets (properties)	9.10	270
Backpack	6.4	181

C

Description	Chapter	Page
Camscripting	8.1	255
Capture The Flag	7.17	240
Cheats	11.2	301
Checking of an area	5.26	115
Class Header	7.11	233
Change behaviour of a unit in an area	5.27	115
Create Fire (script version)	5.75	169
Camera effects	8.7	262
Controlling the camera	8.1	255
Camera coordinates	8.2	256
Creating a camera	8.3	257
Camera is fixed on a vehicle or unit	8.5	260
Create a light source (Script version)	5.72	167
Create marker	5.70	162
Create smoke (script version)	5.74	168
Create dust	5.73	167
Creating weapons and magazines	3.21	94
Create waypoints	5.68	159
Counter	9.5	268

D

Description	Chapter	Page
Dead as condition	5.33	120
Degree of familiarity of a unit	5.29	117
Deathmatch	7.6	227
Delay	9.7	269
Description.ext	2.3	48
Dialog	10.1	279
Distance of two units or objects	5.34	120
Driver/Passenger of a vehicle	5.2	106
Display the speed of a unit	5.9	108
Disable AI units	5.57	144
Deleting units and objects	5.45	124
Defining the Multiplayer Area	7.7	228
Disallow reloading	3.19	93
Damage value	5.24	114
Deleting killed units and vehicles	6.8	194
Defining a death zone	5.25	115

Different text displays	5.66	157
Disable environment sound	8.7	262
Dynamic start-points	6.5	185

E

Description	Chapter	Page
Edit units with allocated waypoints	1.10	40
Eventhandler	5.65	155
Error log file / ArmA.rpt	11.6	305
Escape behaviour of a unit or a group	5.14	110
Empty or locked vehicle	5.1	106
Empty searchlight with light	5.86	177

F

Description	Chapter	Page
Fired ammo type as text message	3.16	92
Finishing a mission	4.6	101
Friendly forces	5.31	118
Flexible respawn points	7.3	225
Friendly enemy	5.30	117
Flag basic information's MP	7.16	238
Force the map on the screen	5.40	121
Friendly forces becomes enemy	5.31	118

G

Description	Chapter	Page
Generate bombs	5.46	126
Generating units and objects	5.45	124
Get in/get out of a vehicle	5.7	108
Generate flares, smoke and explosions	5.46	126
GPS-System	6.2	180
Grafic Formats (Paa/Pac)	2.8	55
Create a group	5.45	124
Group already in vehicle when the mission begins	5.6	108
Getting position displayed	5.64	153
Getting weapon and magazine types displayed	3.15	92
Getting XYZ position displayed	5.64	155

H

Description	Chapter	Page
Height of a unit	5.17	112
House-Patrolling-Script	6.16	205
Hand weapons and static weapons	3.1	64

I

Description	Chapter	Page
If-Then-Else	9.6	268
Information Text	5.59	144
Init.sqs	2.5	53
Insect script	6.20	215
Indestructible Objects	5.61	145

J

Description	Chapter	Page
--------------------	----------------	-------------

K

Description	Chapter	Page
--------------------	----------------	-------------

L

Description	Chapter	Page
Load and unload vehicles	3.5	71
Lip-Files	2.11	57
Logical operators	9.3	267
Locked vehicle	5.1	106
Logical values	9.2	266

M

Description	Chapter	Page
Mission accessories	4.3	98
MP general information	7.19	247
Moving units, objects, triggers and markers	5.15	111
Merging units and markers	1.9	39
Map animation	8.9	263
Mapclick	6.6	187
Merging units and markers	1.9	39
Mine-Script	6.17	208
Mimics	5.54	135
Missions folder	2.1	42
Mission.sqm	2.2	43

Mission name	4.1	97
Mission start	4.2	97
Mission accessories	4.3	98
MP-Description.ext	7.4	226
Missions release	11.5	287
MOD-Folder	11.3	302
Mission appraisal	4.4	99
Mission targets	4.5	99
Move objects	5.16	111
MP score	7.8	229
Moving units, objects, triggers and markers	5.15	111

N

Description	Chapter	Page
Nato Alphabet	11.7	306

O

Description	Chapter	Page
Own profile	11.1	299
Object and building classes	3.11	81
Overview	2.12	58
Own sound/music	5.51	129
Opress player input	5.39	121

P

Description	Chapter	Page
Paratroopers	6.1	179
Preface information for MPMissions	7.19	247
PBO-file	2.9	56
Plant classes	3.12	88
Primary or secondary weapon	3.18	93
Player related text messages	7.22	251
Public variable	7.18	246

Q

Description	Chapter	Page
-------------	---------	------

R

Description	Chapter	Page
Ranks	11.8	307
Running patrol, drive or fly	5.13	110
Respawn positions	7.2	224
Respawn different kinds	7.5	227
Respawn dialog	7.12	233
Random	9.8	269
Read out and display player	5.38	121
Rock classes	3.13	90
Random animation	5.56	139
Random music	5.52	130
Random position mission start	1.9	39
Random weather	5.42	122

S

Description	Chapter	Page
Saboteur	6.21	216
Scout	6.22	217
Start or stop unit/vehicle	5.11	111
Save or load a unit status	5.28	116
Send a radio message	5.50	129
Speed of a unit	5.8	108
Set identity	5.53	134
Saving a mission	4.7	103
Seagull script	6.19	213
Semicolon	9.11	81
Set velocity	5.58	144
Shell classes	3.10	80
Script (.sqs)	2.6	54
Saving	4.7	103
Suppressing gaming speed constantly	6.9	181
Squad.xml	11.9	308
Start parameters	11.10	311
Stringtable.csv	2.4	51
Stringtable basic values	5.67	158
Stringtable basic values in MP	7.13	234
Synchronize	1.6	35
Searchlight	6.14	203
Sign Classes	3.14	91
Synchronize waypoints and trigger	1.6	35
Slow motion or time sprint	5.44	123

T

Description	Chapter	Page
Trigger synchronize with waypoint	1.6	35
Trigger insert	1.4	27
Trigger create	5.69	160
Trigger check its area	5.26	115
Trigger merge several areas	5.26	115
Track down enemy units	6.11	197
Text information	5.59	144
Text and blending effects	8.6	261
Time limit within MP	7.8	229
Time Counter	6.15	204

U

Description	Chapter	Page
User interface	1.1	16
Unit is moving into a building	5.19	112
Unit keeps standing	5.10	109
Unit is moving to its destination	5.12	110
Unit is capitulating itself	6.23	218
Unit has a weapon	3.17	92
Unit classes	3.9	77
Unit in vehicle	5.4	107
Unit/Group already in vehicle	5.6	108
Unit is leaving / joining group	5.20	113
Unit is selecting weapon	5.23	114
Unit turns to another unit	5.22	114
Unit is moving to desired house position	5.63	153
Unit is not allowed to enter a vehicle	5.3	106
Unit is using binoculars	5.77	172
Using ID's – disable lamps	5.61	145
Units keeps lying or keeps standing	5.60	144
Using own sounds	5.52	130
Use own sound/music	5.52	130
Use of addons	11.4	303
Unit is selecting weapon	5.23	114
Unit turns to another Unit	5.22	114

V

Description	Chapter	Page
Vehicle classes	3.7	73
Vehicle weapon classes	3.8	76
Vehicle is waiting for unit	5.6	108
Vehicle respawn	7.14	235
Vehicle respawn (advanced) by Mr-Murray	7.15	236
Variable	9.1	265
Vehicle transport script	6.18	209
Vehicle weapon classes	3.8	76

W

Description	Chapter	Page
Weapon and ammo crates	3.4	71
Weapon classes	3.2	68
Weapon and ammo crates	3.4	71
Weapon selection in the briefing	3.6	72
WaitUntil	9.9	269
While-Do-Loop	9.4	268

X

Description	Chapter	Page
--------------------	----------------	-------------

Y

Description	Chapter	Page
--------------------	----------------	-------------

Z

Description	Chapter	Page
	114	

Syntax Index

A

Description	Chapter	Page
Active	4.5	99
Action	5.55	136
AddAction	6.3	181
AddEventHandler	5.65	155
AddRating	4.4	99
AddSwitchableUnit	5.37	121
AddWeapon	3.3	70
AddMagazine	3.3	70
AddWeaponCargo	3.4	71
AddWaypoint	5.68	159
AddMagazineCargo	3.4	71
AIKills	7.4	226
Alive	5.71	165
AllowGetIn	5.3	106
AllowFleeing	5.14	110
And	9.3	267
Animate	5.61	145
AnimationState	5.56	139
AssignAsCargo	5.78	172
AssignAsCommander	5.78	172
AssignAsDriver	5.78	172
AssignAsGunner	5.78	172
AssignTeam	5.79	173
AssignToAirport	5.82	175
AssignedVehicle	5.78	172
AssignedVehicleRole	5.78	172
AVGScore	4.4	99

B

Description	Chapter	Page
BuildingPos	5.62	148

C

Description	Chapter	Page
Call	9.13	274
CamUseNVG	8.7	262
CanMove	5.81	174
CanFire	5.81	174
CanStand	5.81	174

Case	9.3	267
Ceil	9.3	267
ClearWeaponCargo	3.4	71
ClearMagazineCargo	3.4	71
CloseDialog	10.9	292
CommandTarget	5.21	113
CommandFire	5.80	174
CommandMove	5.80	174
CommandStop	5.80	174
CommandTarget	5.80	174
Compile	9.13	274
ComposeText	8.6	261
Count	9.3	267
CreateDialog	10.9	292
CreateGroup	5.45	124
CreateMarker	5.70	162
CreateMine	3.20	93
CreateSoundSource	5.51	129
CreateTrigger	5.69	160
CreateVehicle	5.45	124
CreateVehicleLocal	5.45	124
CreateUnit	5.45	124
Crew	5.71	165
CutRsc	10.7	289
CutText	8.6	261

D

Description	Chapter	Page
Damage	5.24	114
Default	1.2	20
DefValueParam	7.8	229
DeleteMarker	5.70	162
DeleteStatus	5.28	116
DeleteVehicle	5.47	127
DeleteWaypoint	5.68	159
Driver	5.71	165
Done	4.5	99
DoFire	5.21	113
DoGetOut	5.47	127
DoMove	5.12	110
DoStop	5.10	109

DoTarget	5.21	113
DoWatch	5.22	114
DisableAI	5.57	144
DisableUserInput	5.39	121
DissolveTeam	5.79	173
Distance	5.34	120

E

Description	Chapter	Page
EnableAI	5.57	144
EnableEnvironment	8.7	262
EnableRadio	5.50	129
EnableReload	3.19	93
Enemy	5.31	118
ExecVM	9.13	274
ExitWith	9.13	274

F

Description	Chapter	Page
FadeMusic	5.52	129
FadeRadio	5.52	129
FadeSound	5.52	129
Failed	4.5	99
Fire	5.21	113
False	9.2	266
Flag	7.16	238
FlagOwner	7.16	238
Floor	9.3	267
FlyInHeight	5.17	112
Format	5.66	157
ForEach	9.3	267
ForceMap	5.40	121
ForceSpeed	5.71	165
Fuel	5.71	165

G

Description	Chapter	Page
GameType	7.11	233
GetDammage	5.24	114
GetDir	5.22	114

GetMarkerPos	1.7	36
GetPos	5.22	114
GetPosASL	5.64	153
GetSpeed	5.8	108
GlanceAt	5.22	114
GlobalChat	5.50	129
Goto	9.3	267
Group	5.6	108
GroupChat	5.50	129
GrpNull	5.20	113
Gunner	5.71	165

H

Description	Chapter	Page
HandsHit	5.81	174
HasWeapon	3.17	92
Hidden	4.5	99
Hint, HintC, HintCadet	5.59	144

I

Description	Chapter	Page
If then else	9.6	268
In	5.71	165
InFlame	5.36	121
IsEngineOn	5.71	165
IsNull	5.71	165
IsServer	7.19	247

J

Description	Chapter	Page
Join	5.20	113

K

Description	Chapter	Page
KnowsAbout	5.29	117

L

Description	Chapter	Page
Land	5.18	112
LandAt	5.82	175

Leader	5.79	173
LimitSpeed	5.71	165
List	9.3	267
Localize	5.66	157
Local Player	7.19	247
Local Server	7.19	247
Lock, Locked	5.1	106
LookAt	5.22	114
LoadIdentity	5.53	134
LoadStatus	5.28	116

M

Description	Chapter	Page
Magazines	3.15	92
MapAnimAdd	8.9	263
MapAnimClear	8.9	263
MapAnimCommit	8.9	263
MapAnimDone	8.9	263
MaxPlayers	7.11	233
MaxScore	4.4	99
MinPlayers	7.11	233
MinScore	4.4	99
ModelToWorld	5.15	111
Move	5.12	110
MovelnCargo	5.2	106
MovelnCommander	5.2	106
MovelnDriverMovelnGunner	5.2	106
MovelnTurret	5.2	106

N

Description	Chapter	Page
NearestBuilding	5.61	145
NearestObject	5.61	145
Not	9.3	267

O

Description	Chapter	Page
ObjNull	5.21	113
ObjStatus	4.5	99
OnLoadIntro	4.2	97

OnLoadIntroTime	4.2	97
OnLoadMission	4.2	97
OnLoadMissionTime	4.2	97
OnPlayerConnected	7.22	252
OnMapSingleClick	6.6	187
OrderGetIn	5.78	172

P

Description	Chapter	Page
Player	9.1	265
PlayMove	5.56	139
PlayMusic	5.52	130
PlaySound	5.52	130
Position	5.64	153
PreLoadCamera	8.8	262
PreLoadMusic	8.8	263
PreLoadObject	8.8	262
PreLoadSound	5.52	130
PreLoadTitleRsc	8.8	263
PreLoadTitleText	8.8	263
PreProcessFile	9.13	274
PrimaryWeapon	3.18	93
PublicVariable	7.18	246

Q

Description	Chapter	Page
-------------	---------	------

R

Description	Chapter	Page
Random	9.8	269
Rank	5.76	171
Rating	4.4	99
RemoveAction	6.3	181
RemoveEventHandler	5.65	155
RemoveWeapon	3.3	70
RemoveAllWeapons	3.3	70
RemoveMagazine	3.3	70
RemoveSwitchabeUnit	5.37	121
Respawn	7.4	226

RespawnDelay	7.4	226
RespawnDialog	7.12	233
RespawnVehicle	7.14	235
RespawnVehicleDelay	7.14	235
Reveal	5.29	117

S

Description	Chapter	Page
Saving	4.7	103
SaveGame	4.7	103
SavelIdentity	5.53	134
SaveStatus	5.28	116
SaveVar	9.1	265
Say	5.51	129
ScriptDone	8.8	263
SecondaryWeapon	3.18	93
SelectLeader	5.37	121
SelectPlayer	5.37	121
SelectWeapon	5.23	114
Server	7.19	247
SetAccTime	5.44	123
SetAirportSide	5.82	175
SetAmmoCargo	1.2	20
SetAperture	8.7	262
SetBehaviour	1.5	30
SetCaptive	5.30	117
SetCombatMode	1.4	27
SetDamage, SetDammage	5.24	114
SetDate	5.43	123
SetDir	1.2	16
SetFace	5.53	134
SetFace Animation	5.53	134
SetFlagOwner	7.16	238
SetFlagSide	7.16	238
SetFlagTexture	5.35	120
SetFog	5.42	122
SetFormation	1.5	30
SetFormDir	1.2	20
SetFriend	5.31	118
SetFuel	5.71	165
SetFuelCargo	5.71	165

SetGroupID	5.49	128
SetIdentity	5.53	134
SetLightAmbient	5.73	167
SetLightBrightness	5.73	167
SetlightColor	5.73	167
SetMarkerBrush	5.70	162
SetMarkerColor	5.70	162
SetMarkerDir	5.70	162
SetMarkerPos	1.7	36
SetMarkerShape	5.70	162
SetMarkerSize	5.70	162
SetMarkerText	5.70	162
SetMarkerType	5.70	162
SetMusicEffect	5.69	160
SetMimic	5.54	135
SetOverCast	5.42	122
SetPos	5.15	111
SetRadioMessage	5.48	127
SetRain	5.42	122
SetRank	5.76	171
SetRepairCargo	1.2	20
SetSkill	1.2	20
SetSoundEffect	5.69	160
SetSpeedMode	5.8	108
SetTargetAge	1.2	20
SetTerrainGrid	5.83	176
SetTriggerActivation	5.69	160
SetTriggerArea	5.69	160
SetTriggerStatements	5.69	160
SetTriggerText	5.69	160
SetTriggerTimeOut	5.69	160
SetTriggerType	5.69	160
SetUnitAbility	1.2	20
SetUnitPos	5.60	144
SetVehicleAmmo	3.3	70
SetVehicleArmor	5.24	114
SetVehicleInit	1.2	20
SetVehiclePosition	5.15	111
SetVeloCity	5.58	144
SetViewDistance	5.41	122
SetWaypointType	5.68	159

SetWaypointBehaviour	5.68	159
SetWaypointCombatMode	5.68	159
SetWaypointDescription	5.68	159
SetWaypointFormation	5.68	159
SetWaypointHousePosition	5.68	159
SetWaypointPosition	5.68	159
SetWaypointScript	5.68	159
SetWaypointSpeed	5.68	159
SetWaypointStatements	5.68	159
SetWaypointTimeOut	5.68	159
SetWaypointType	5.68	159
ShowCinemaBorder	8.4	260
ShowCompass	4.3	98
ShowDebriefing	4.3	98
ShowGPS	4.3	98
ShowMap	4.3	98
ShowNotePad	4.3	98
ShowRadio	4.3	98
ShowWatch	4.3	98
ShowWaypoint	5.68	159
Side	5.38	121
SideChat	5.50	129
SkipTime	5.43	123
Sleep	9.7	269
SomeAmmo	3.19	93
Spawn	9.13	274
SpeedIsNull	5.71	165
Stop	5.11	109
SwitchLight	5.61	145
SwitchMove	5.56	139
SwitchCamera	6.22	217

T

Description	Chapter	Page
This	9.1	265
ThisList	9.3	267
Terminate	8.8	263
TextParam	7.8	229
Time	9.1	265
TitleCut	8.6	261
TitleFadeOut	8.6	261

TitleText	5.66	157
TitleParam	7.8	229
TitleRsc	10.3	283
True	9.2	266
TypeOf	5.71	165

U

Description	Chapter	Page
UnAssignTeam	5.79	173
UnAssignVehicle	5.7	108
Unlocked	1.2	20
Unit, Units	5.6	108

V

Description	Chapter	Page
ValuesParam	7.8	229
Variable	9.1	265
Vehicle	5.71	165
VehicleChat	5.50	129
Visible	4.5	99

W

Description	Chapter	Page
WaitUntil	9.9	269
Weapons	3.15	92
WeaponDirection	3.22	95
WeaponHolder	3.21	94
While do	9.4	268

X

Description	Chapter	Page
-------------	---------	------

Y

Description	Chapter	Page
-------------	---------	------

Z

Description	Chapter	Page
-------------	---------	------

IMPRINT

- Author:** Sascha "Mr-Murray" Hoffmann
Website: www.mr-murray.de.vu
- Text, layout and design:** Sascha Hoffmann
Graphics and design: Sascha Hoffmann
Editorial office: Sascha Hoffmann, Daniel Schönyan, Dan Bolan, Matt Rochelle
English translation: Daniel "Memphisbelle" Schönyan
English editorial office: Dan "Metal0130" Bolan and Matt Rochelle
Publisher: Mr-Murray
Game developer: Bohemia Interactive (www.bistudio.com)
Credits: Memphisbelle, Metal0130, Matt Rochelle, Parvus, Wolle, Low Fly, Placebo, Maruk, Suma, Ivan, Pete, Shadow, Jan Hlavatý, Dslyecxi, Hoz, Jahve, Planck, RichUK, Q, Raedor, Rastavovich, Vilem, Jerry Hopper, Deadeye, M-E, Foxhound, C930, Blackland, Vienna, Marco-Polo-IV, MCPXXL, Silola, Chneemann, Lester, Swat, Pit, Sgt Ace, SNKMan, Legislator, Imutep, Crowe, Wolf der Kleine, Berghoff, Al Simmons, Pitti, Sniping-Jack, JörgF, BadAss, Unterfeld, OneManGang, Flashpoint_K, Kriegerdaemon, Wüstenfuchs, LockheedMartin\$ch, Burns, Simba, LordOfTheFlames, Spinor, OFPEC-Team, Deadeye, Luemmel, Zenshin, Moses, Hardrock, TheArmALord, Bolek, Flimmi, Lima, NoFu, Tajin, John Silver, T_D, Redfish, Buliwyf, Big X, Teufelsklaue, Clausewitz, Vektorboson, Cervo, Mandoble, NeoArmageddon, Mondkalb, Colonel Klink, Woody
- Credits Screenshot contest:** Marcus-Ergalla, Burns, Stoned Boy, Laggingape, Swat, Xsive, Woody, Evil Ash, Legislator, Arctic, AlexXx, Blechreiz, Churchill, Switcher83, Blacktiger
- Game version:** Game version 1.15



Copyright

©2008 Sascha "Mr-Murray" Hoffmann. All rights reserved. No part of this publication may be reproduced, stored in a retrieval system, or transmitted in any form or by any means, electronic, mechanical, photocopying, recording or otherwise, without prior permission of the copyright owner.

All resources used in this product as Company names, Names, Logos, Graphics, Trademarks and Product descriptions are serving for identification applications only. They belong to the only legal owner.

Cover artwork and Arma[®] Logo is copyright of Bohemia Interactive[®] (BIS) and it is used with permission.

©2008 Bohemia Interactive[®]. All rights reserved.



EDITING GUIDE - DELUXE EDITION

It doesn't matter whether you are a beginner or a professional editor. This reference is the ultimate tutorial for the Armed Assault Mission Editor. Anyway, whether Single player or even Multiplayer Missions, This tutorial provides detailed explained examples and shows the nearly unlimited possibilities of the Armed Assault Mission Editor.

Create your own dynamic sceneries with all possible Situations of huge battles. Become your ideas realized and inspire the community with your missions. So you will become a member of that huge and fantastic community which this book is devoted to

Detailed class lists of all units, vehicles, objects and weapons are only some of the features provided in this book. It's basically founded upon the game version 1.15 of Armed Assault and also contains the complete content of the official Addon Queens Gambit and the new CTI-Variant Warfare.

The Author, Sascha "Mr-Murray" Hoffmann has already released his very first version of his guide for Operation Flashpoint in 2001. So he used his experiences which he was collected from Operation Flashpoint and Armed Assault while he was writing this book. With this book he has shown you all the tricks and possibilities which can be utilized in the mission editing. So you'll be able to pretty much to create your own fantastic missions.

This guide is also used as a cornerstone for the official successor Armed Assault II, because nearly all basic functions will keep the same.

[www.mr-murray.de.vu]

[www.mapfact.net]

[www.ofpec.com]