ARMA
ARMED ASSAULT

EDITING GUIDE

SE-Version 1.02

EDITING GUIDE

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 BI, Morphicon 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. Furthermore, I would like to say thank you to the Morphicon Team, especially to Morris Hebecker and Alexander Harlander for their powerful support while realizing and marketing this book.

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 for the connection to Morphicon and Andreas Holzwart, Rastatovich and Wolle for their amazing support. Real special thanks to the translator of this guide MemphisBelle and his helping hands Metal0130 an 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

Table Of Contents

Chapter 4: The Mission

Chapter 5: Mission Accessories

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.

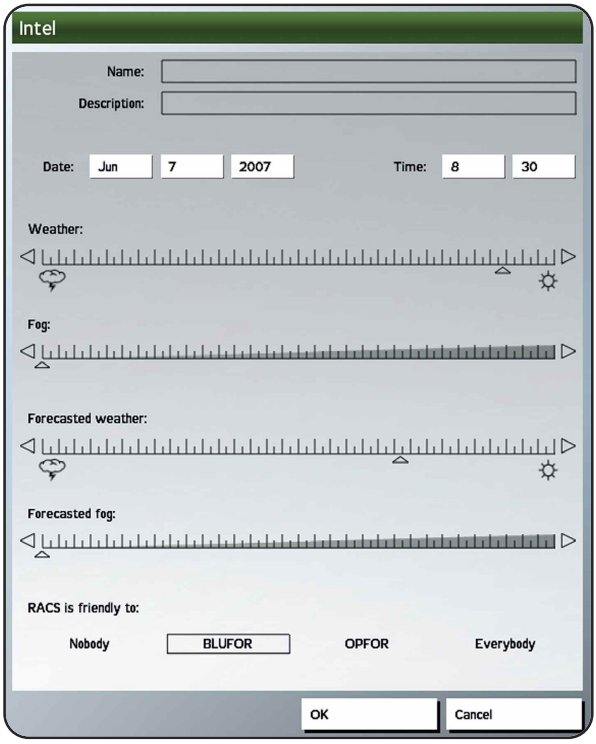
Arctic (Ronny Krischker)

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 one 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

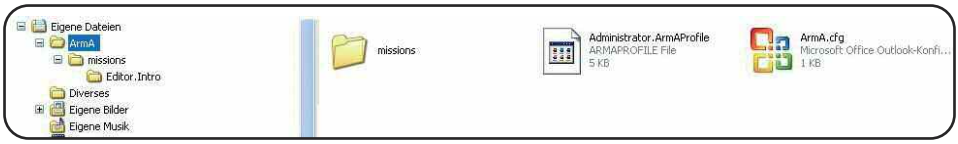
Merging is similar to importing. Merging makes it possible to import other missions or units, markers, objects, triggers and waypoints from another map into the current mission. By using the **Merge**-Button everything on the map will get imported, but not the contents of the mission folder.

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. She 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. He will be able to destroy it now 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

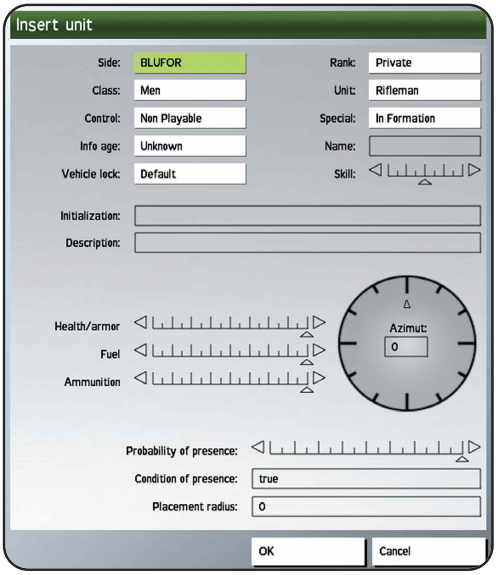
To close the editor and return to the main menu, just click on the **Exit**-Button.

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	**Choice of side**
East	East Units
West	West Units
Independent	Resistance Units
Civilian	Civilian Units
Empty	Empty Vehicles
Logic	Game Logic

Class	**Kind of units**
Air	Helicopter, Airplane
Ammo	Weapons, Ammo
Armored	Armored Vehicles
Car	Cars, Motorbikes
Men	Soldiers
Mines	Mines
Objects	Static Objects
Ship	Ships
Sounds	Sounds
Static	Guns, Machineguns
Support	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, playable units needed. If the gamer wants to use character switch, the player is able to switch between the units to bring them to different positions.

Player	The player himself
Playable	Playable unit
Non Playable	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
Sergant
Lieutenant
Captain
Major
Colonel

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

```
Player setRank "Sergant"
```

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

This sets the direction of view for a unit when it has been placed on the map. The unit will look in the predefined direction after the user has pressed the **OK**-Button.

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 with pressed 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 immediately. Scripts can be defined here. They will be executed by the game immediately as well. There is a possibility for players / users to make it easier. They only need to create a text file called **Init.sqs** in the missions folder. The **Init.sqs** will be executed by the game without a predefined Syntax. Every definition of the unit can be placed into the **Init.sqs**. You will find a more accurate explanation in **Chapter 2**.

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 setdammage 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 setdammage 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 percent. 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"

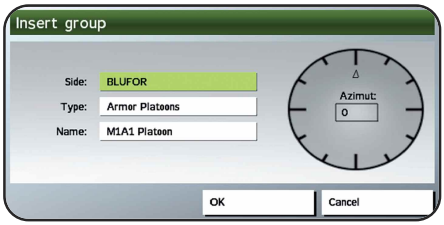
It is also possible to define the info-age by using a syntax. This is what it would look like:

Player setTargetAge "10 MIN"

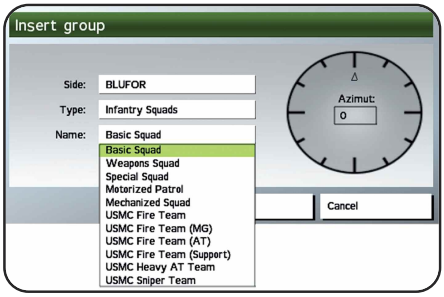
1.3 - Adding Groups (F2)

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 one has decided on a side for the units, one has 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 APC
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**".

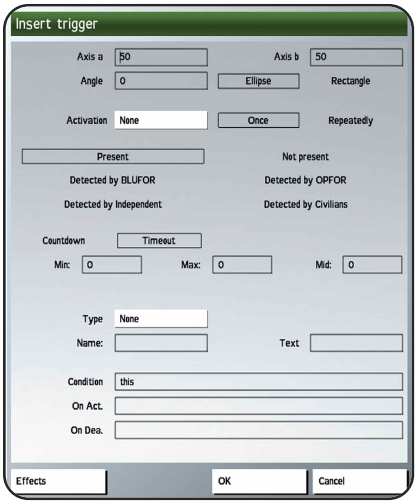
The direction of view needs to be adjusted by setting the **Azimut** again.

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 is entering a 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 on **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 on "**true**" in the **OnActivation**-Box of the trigger, waypoint, or written in a script. The action defined will be activated now.

Verifying a condition

A further possibility is 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 looks 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 is entering a trigger area, then a further option, called "entry", will be add 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

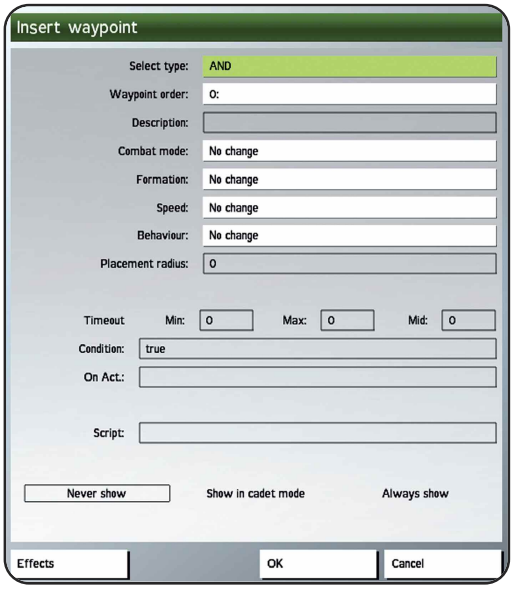
When it's needed, the user can move or delete a trigger. But to make sure that it works, the trigger has to be equipped with a name first. That's important to speak to the trigger directly. The following syntax is needed:

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
The user can set individual settings for each waypoint that will be executed when the unit reaches the waypoint.

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 by 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

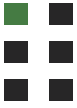
This behaviour is definable by script as well. The syntax is:

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.

Columm



Staggered Columm

Wedge

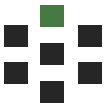
Vee

Echelon Left

Echelon Right

Line

Delta



Column (compact)

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.

Normal / Limited / 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 is reaching 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

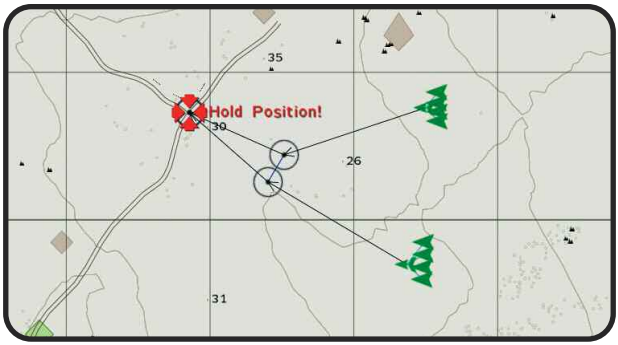
It's possible to show or hide waypoints. The user can define here whether they are only visible in Cadet Mode, general, or not visible.

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 will reach their destinations and are still

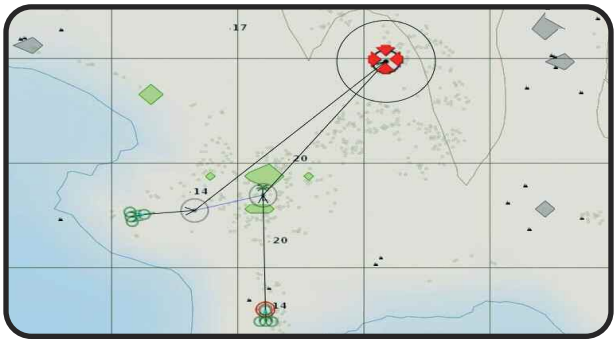
alive, they will come together as one group now. It will only work if the groups are not too big. ArmA allows up to **144 units** to a group including their leaders.



1.6 - Synchronize (F5)

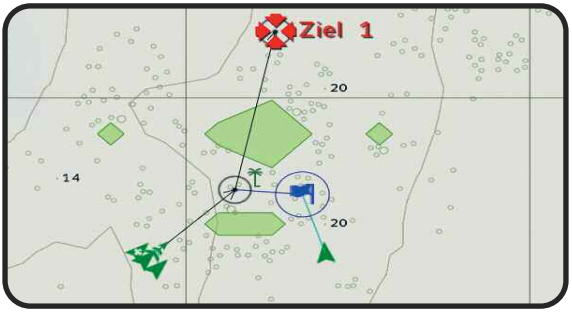
The option "**Synchronize**" offers the possibility to connect waypoints with waypoints or waypoints with triggers. It enables the units to remain in the same formations and the user does not have to use too many variables.

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 her waypoint, she'll wait there until unit 2 arrives at her destination.

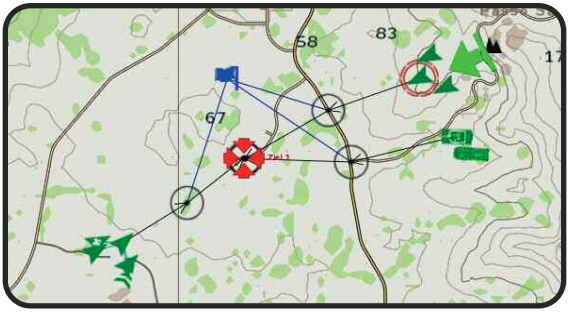


The trigger-waypoint combination works the same way. The groups will only move on when the trigger is executed. It's not necessary to allocate a variable to the unit at the waypoint (like **Grp1go** for example). Enter the command **Grp1go=true** into the OnActivation box of the trigger so that the group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



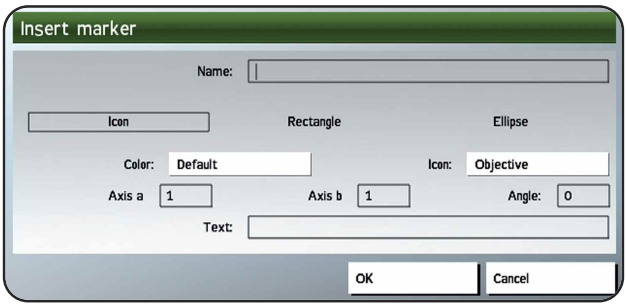
Three groups and a radio trigger will get synchronized with each other in the picture below. If all 3 groups have reached their positions the user only has to give the order to attack by using **Radio 0-0-1**. The units will move to the next position, the target.



1.7 - Adding Markers (F6)

The markers are necessary to create a tactical style on the map. The markers are showing the player the course of the mission, the targets, and more information that 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 **cycle**. With the last two choices it's possible to mark whole enemy or friendly areas.

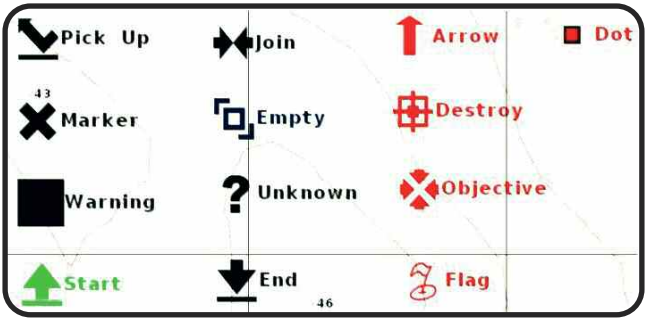
Color

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.

Objective (**Flag**)	**Join**
Flag1	**PickUp**
Dot	**Unknown**
Destroy	**Marker**
Start	**Arrow**
End	**Empty**
Warning	

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**.

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"

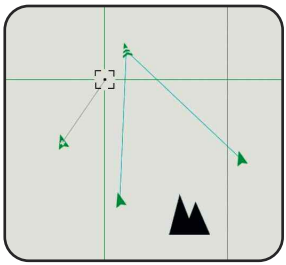
1.8 - Rotating Units And Objects

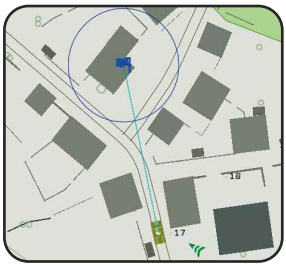
To twist whole groups, objects or single units it's necessary to select them first. Then the user needs to press the **Shift-Key**, click and hold the **Left Mouse Button** while moving the crosshair by using the mouse until the object is turned towards the desired direction. It's possible to turn more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

1.9 - Merging Units

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.

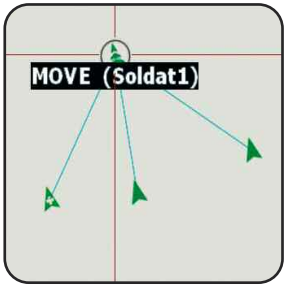


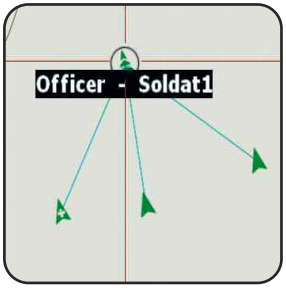


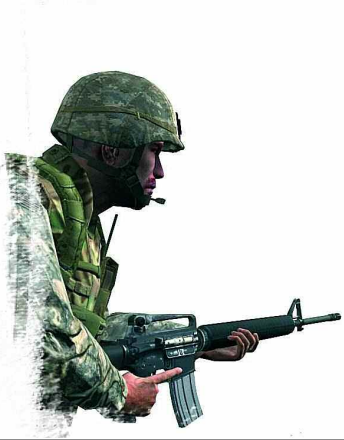
1.10 - Edit Units With Allocated Waypoints

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 she 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.

Legislator (Daniel Bötticher)

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 (.sqs)
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.sqs"

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-add-ons.

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
Forcasted weather
Forcasted 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 Item1**
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	YXZ-Coordinates of the player in order X ZY
Azimut	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	YXZ-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 Excel!

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 implant 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 quiet 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= B I S T U D I O  p r o u d l y  p r e s e n t s
onLoadMission= A R M E D  A S S A U L T
// === 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_Titel;

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_Titel**
Out of the description or the config:	**$STR_Titel**

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´tsleep 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 funktion
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 the time of edit 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 run through the scripts 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 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 accept 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 add-ons. 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 -->
```

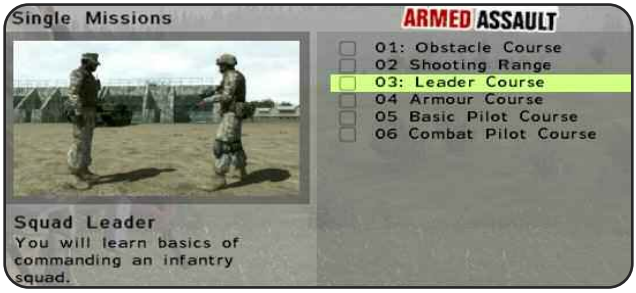
```
0.400, 0
0.440, 1
0.480, 3
0.520, 0
0.560, 1
        4 Frame-Pauses
0.720, 2
        2 Frame-Pauses
0.800, 1
0.840, 2
0.920, 1
1.000, 2
```

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"><img src="Title.paa" width="230" height="115"></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>
```

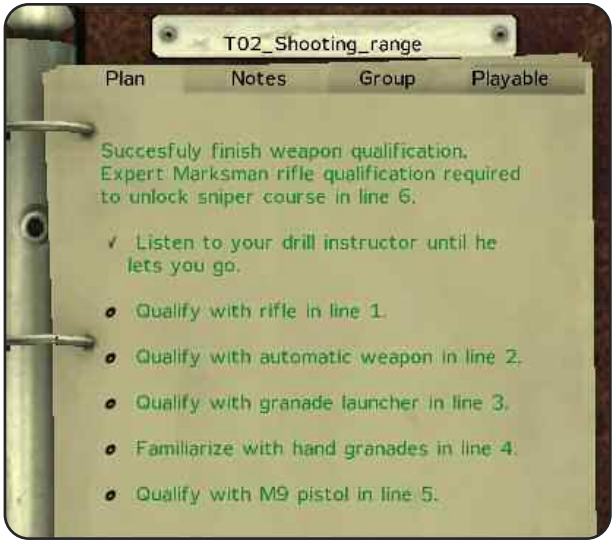
2.13 - The Briefing

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 granade launcher in line 3.
</p><hr>
<p><a name="OBJ_5"></a> Familiarize with hand granades 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 both 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 sentense 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

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.

Laggingape (Till Breuer)

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
100Rnd_127x99_M2

SearchLight

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

Waffe: **D30**
Magazin: 30Rnd_122mmHE_D30

DSHKM
50Rnd_127x107_DSHKM

AGS
29Rnd_30mm_AGS30

Equipment (general)

Weapon: **Handgrenade**
Magazine: Handgrenade

HandGrenadeTimed
HandGrenadeTimed

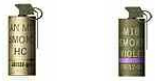
Pipebomb
Pipebomb

Weapon: **Mine**
Magazine: Mine

MineE
MineE

Binocular

Weapon: **NVGoggles**
Magazine:



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 M16A4 M16A4_ACG M4 M4A1 M4A1SD M4AIM M4SPR	M16A2 M16A4 M16A4 - Scope M 4 - Standard M 4 A1 - Standard M 4 - Silencer M4 - Aimpoint M 4 - Scope	Magazine:	20Rnd_556x45_Stanag 30Rnd_556x45_Stanag 30Rnd_556x45_StanagSD
M16A2GL M16A4_GL M16A4_ACG_GL M4GL M4A1GL	Guns with Grenade Launcher	Magazine: Grenades: Flares:	20Rnd_556x45_Stanag 30Rnd_556x45_Stanag 30Rnd_556x45_StanagSD 1Rnd_HE_M203 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 G36C G36K	G 36 – Standard G 36 - Commando G 36 – Commando II	Magazine:	30Rnd_556x45_G36
M24	Sniper Rifle	Magazine:	5Rnd_762x51_M24
M107	Heavy Sniper Rifle	Magazine:	10Rnd_127x99_M107
MP5A5 MP5SD	MP5 - StandardMP5 - Silencer	Magazine:	30Rnd_9x19_MP5 30Rnd_9x19_MP5SD
M9 M9SD	Pistol Pistol - Silencer	Magazine:	15Rnd_9x19_M9 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: Grenade: Flares:	30Rnd_545x39_AK 1Rnd_HE_GP25 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_AK AK74PSO
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"

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:

```
// The part class Weapons begins right here
class Weapons
{
        class M4A1SD                    The kind of weapons
        {
        count = 6;                      The number of weapons
        };
        class M136                      The kind of weapons
        {
        count = 2;                      The number of weapons
        };
 };
// The part Class Weapons ends right here

// The part class Magazines begins right here
class Magazines
{
        class 30Rnd_556x45_StanagSD     The kind of magazines
        {
        count = 20;                     The number of magazines
        };
        class M136                      The kind of magazines
        {
        count = 6;                      The number of magazines
        };
        class HandGrenade               The kind of magazines
        {
        count = 20;                     The number of magazines
        };
};
// The part Class Magazines ends right here
```

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

One can see a list of the vehicles below. This list contains each single class with their editor names, a description, and the class name which is needed to place the vehicle on the map by using a syntax.

WEST

Vehicle	Description	Class Name
Land		
M1Abrams	Tank	M1Abrams
M113	Armoured Personnel Tank	M113
M113Ambul	Ambulance Tank	M113Ambul
Vulcan	Anti Aircraft Tank	Vulcan
Stryker ICV M2	Light Tank with M2-Maschine 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-Maschine 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-Maschine 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
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 Maschine Gun	RHIB
RHIB 2 Turret	Patrol Boat with Maschine Gun, Grenade Launcher	RHIB2Turret

EAST

Vehicle	Description	Class Name
	Land	
T72	Tank	T72
BMP2	Armoured Personnel Tank	BMP2
BMP2Ambul	Ambulance Tank	BMP2Ambul
ZSU	Anti Aircraft Tank	ZSU
BRDM2	Light Tank	BRDM2
BRDM2_ATGM	Light Tank with AT-Launcher	BRDM2_ATGM
UAZ	Jeep	UAZ
UAZMG	Jeep with Maschine Gun	UAZMG
Ural	Truck	Ural
UralOpen	Truck – open	UralOpen
UralRepair	Truck – Repair Truck	UralRepair
UralReammo	Truck – Reammo Truck	UralReammo
UralRefuel	Truck – ReFuel Truck	UralRefuel
Motorcycle	Motorcycle	TT650G
	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 Maschine 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	
M113 RACS	Armoured Personnel Tank	M113_RACS
Vulkan RACS	Anti Aircraft Tank	Vulkan_RACS
4x4	Jeep (closed)	Landrover
4x4 MG	Jeep with Maschinen Gun M2	LandroverMG
4x4 Open	Jeep (open)	Landrover_Closed
	Air	
AH6 RACS	Little Bird Helicopter armed	AH6_RACS
MH6 RACS	Little Bird	MH6_RACS
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 weiß	Skoda
Skoda (Blue)	Skoda blue	SkodaBlue
Skoda (Red)	Skoda red	SkodaRed
Skoda (Green)	Skoda green	SkodaGreen
Policecar	Police Jeep	Landrover_Police
Bus	City Bus	Bus_city
UralCivil	Truck yellow (closed)	UralCivil
UralCivil 2	Truck blue (open)	UralCivil2
Tractor	Tractor	Tractor
Motorcycle	Motorcycle	TT650C
Air		
Parachute	Parachute	ParachuteC

3.8 - 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 Maschine Gun M249	SoldierWAR
Camel Pilot	Biplane Pilot	BISCamelPilot
Crewman	Vehicle Crew	SoldierWCrew
Engineer	Engineer	SoldierWMiner
Grenadier	Grenadier	SoldierWG
Machinegunner	Soldier with Maschine Gun M240	SoldierWMG
Medic	Corpsman	SoldierWMedic
Officer	Officer	OfficerW
Pilot	Pilot	SoldierWPilot
Rifleman	Soldier with M4AIM	SoldierWB
SF Assault	Special Forces with M4A1GL	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	SoldierWSniper
Squad Leader	Squad Leader with M4 AIM	SquadLeaderW
Team Leader	Team Leader with M4 AIM	TeamLeaderW

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 Maschinen Gun PK	SoldierEMG
Medic	Corpsman	SoldierEMedic
Officer	Officer	OfficerE
Pilot	Pilot	SoldierEPilot
Rifleman	Soldier	SoldierEB
Sniper	Sniper with Dragunov (SVD)	SoldierESniper
Squad Leader	Squad Leader	SquadLeaderE
Team Leader	Team Leader	TeamLeaderE

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 Maschinen Gun M240	SoldierGMG
Medic	Corpsman (Medic)	SoldierGMedic
Officer	Officer	SoldierG
Pilot	Pilot	SoldierGPilot
Rifleman	Soldier	SoldierGB
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 M24	SoldierGSniper
Squad Leader	Squad Leader	SquadLeader
Team Leader	Team Leader	TeamLeader

CIVILIAN

Unit	Description	Class Name
Civilian to Civilian21	Nearer description is not necessary. Numbered from Civilian to Civilian21.	Civilian
N/A	King	King
N/A	War Reporter	FieldReporter
N/A	Bodyguard	Anchorman
N/A	Prime Minister	NorthPrimeMinister
N/A	Female Reporter	MarianQuandt
N/A	Female Reporter	MarianQuandt02
N/A	Female Reporter	MarianQuandt03
N/A	Female Reporter	MarianQuandt04
N/A	Zombie	Civil_Undead_1
N/A	Zombie	Civil_Undead_2
N/A	Zombie	Civil_Undead_3
N/A	Zombie	Civil_Undead_4

Units 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**.

INSECTS

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

3.9 - 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];
```

3.10 - 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:

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]}]
```

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.

Switcher83 (Matthias Schulz)

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 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 above 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 accessory the number **1** or **0** respective **false** or **true** are needed. 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

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 "Mission plan 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 he 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 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:

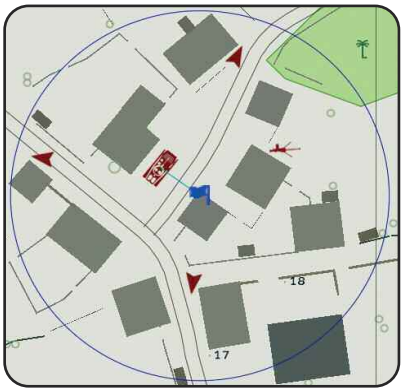
Hidden	- The mission target will be hidden
Visible	- The mission target will be visible again
Active	- The mission target is active
Done	- The mission target is done
Failed	- The mission has been failed.

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:

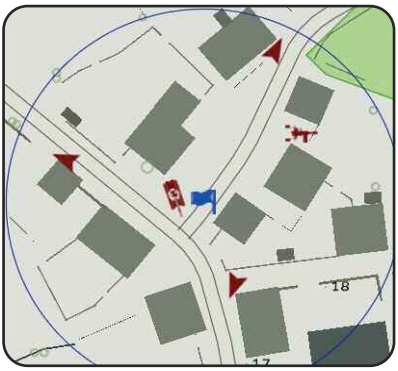
! (alive Tank1) AND ! (alive MG1) AND ! (alive Soldier1)

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: End #1
Activation: EAST
Not present

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):

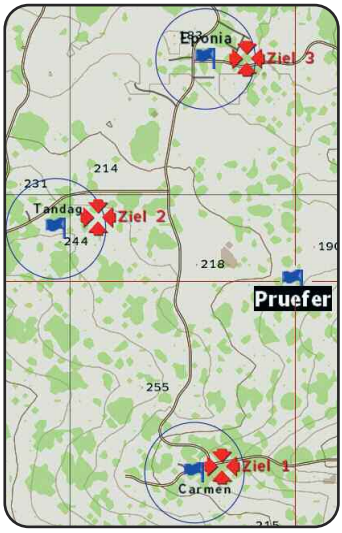
Axis a/b: 150
Activation: EAST (not present)
onActivation: Target2=true

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 on 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 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.

Xsive (Pierre Wirsik)

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 more than 60 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

To place an empty vehicle on the map just presses the F1 -Key and double click on the map. When the menus 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 will not to 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 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

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 is moving only 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 Fahrzeug1	- Unit is leaving vehicle
{unAssignVehicle _x} forEach units Grp1	- 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:

Name1 setSpeedMode "Limited"	**-** slow
Name1 setSpeedMode "Normal"	**-** medium
Name1 setSpeedMode "Full"	**-** fast

5.9 - Make units move or stop

In this example a unit or group is used which has been assigned several waypoints. It might be necessary that this unit stops between 2 waypoints. To make this work, just use this syntax:

Name1 stop true

The named unit will keep its position now until the syntax will be set to false again.

Name1 stop false

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 is following him when he is changing its 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 it is 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**
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 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

5.16 - Placing objects higher or lower

Nearly all objects which were placed on the map can be set higher or deeper. 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 appears. 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 glaceAt 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 getDammage will work only by using two **m**.

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

Name setDamage 1 or **Name setDammage 1**

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

Name1 setDamage getDammage Name2

Name1 setDammage getDammage Name2

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

? damage Name >= 0.5 or **? getDammage Name >= 0.5**

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 - Checking of 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:

Player sideChat format ["%1",list Area1]

If the player is using the radio in game, then the units of **Area1** will be displayed.

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.

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 with rather than 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 flagstaffs 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

Russia
rus_vlajka_co.paa

Sovjet union
rus_vlajka_co.pac

SLA
sever_vlajka.paa

Racs
jih_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 that unit unplayable again:

removeSwitchableUnit Name

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

selectPlayer Name

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 at 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

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]

Einheitsbezogen:
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 rang 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**.

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.

```
GrpOne = Creategroup EAST;
_Leader="SquadLeaderE" createUnit [getMarkerPos "GrpOneM", GrpOne, "", 1, "Sergant"];
_Unit2="SoldierEB" createUnit [getMarkerPos "GrpOneM", GrpOne, "", 1, "Corporal"];
_Unit3="SoldierEB" createUnit [getMarkerPos "GrpOneM", GrpOne, "", 1, "Corporal"];
_Unit4="SoldierEG" createUnit [getMarkerPos "GrpOneM", GrpOne, "", 1, "Corporal"];
_Unit5="SoldierEMG" createUnit [getMarkerPos "GrpOneM", GrpOne, "", 1, "Corporal"];
_Unit6="SoldierEAT" createUnit [getMarkerPos "GrpOneM", GrpOne, "", 1, "Corporal"];
_Unit7="SoldierESniper" createUnit [getMarkerPos "GrpOneM", GrpOne, "", 1, "Corporal"];
 exit
```

As one can see, the first unit gets renamed as leader and she 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:

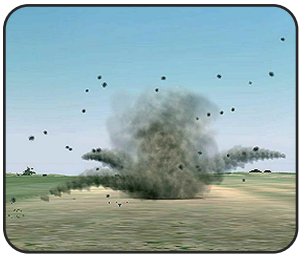
Smoke1="Smokeshell" createVehicle position Name

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 all which has a 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"]
```

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 normaly - 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!"

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[] = {\music\track2.ogg, db+0, 1.0};
	};
};

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

Continued on next page.

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

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 a 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};
                    };
 };
```

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.

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 **deleteIdentity "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:

Name1 setMimic "Smile"

Normal - **Surprise** - **Agresive** - **Hurt** - **Ironic** - **Smile** - **Cynic** - **Angry** - **Sad**

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

Name1 setFaceAnimation 0.5

5.55 - The Action Order

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 granade

["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 of 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 "Animation command"

Name switchMove "Animation command"

A list with the most used commands will be shown on the list below:

Animation	Bezeichnung
AmovPercMstpSnonWnonDnon_carCheckPush	Vehicle control
AmovPercMstpSnonWnonDnon_carCheckWheel	Vehicle control
AmovPercMstpSnonWnonDnon_carCheckWash	Washing Vehicle
AmovPercMstpSnonWnonDnon_exerciseKata	Martial Arts
AmovPercMstpSnonWnonDnon_exerciseKneeBendA	Knee-bend slow
AmovPercMstpSnonWnonDnon_exerciseKneeBendB	Knee-bend fast
AmovPercMstpSnonWnonDnon_exercisePushup	Pushup
AmovPercMstpSlowWrflDnon_Salute	Starts saluting
AmovPercMstpSlowWrflDnon_SaluteIn	Strats saluting
AmovPercMstpSrasWpstDnon_SaluteIn_end	Ends saluting
AmovPercMstpSlowWrflDnon_SaluteOut	Ends saluting
AmovPercMstpSrasWpstDnon_SaluteOut_end	Ends salutíng
AmovPercMstpSnonWnonDnon_seeWatch	Watching the clock
AmovPercMstpSnonWnonDnon_talking	Is talking
AmovPercMstpSlowWrflDnon_talking	Is talking
ActsPercMstpSnonWnonDnon_MarianQ_shot1man	Is talking
ActsPercMstpSnonWnonDnon_MarianQ_shot3man	Is talking
ActsPercMstpSnonWnonDnon_MarianQ_shot4man	Secure
ActsPercMstpSnonWnonDnon_MarianQ_shot5man	Secure 2
ActsPercMstpSnonWnonDnon_MarianQ_TVstudioMan_Loop1	Sitting
AmovPsitMstpSlowWrflDnon_Smoking	Sitting, smoking
AmovPsitMstpSlowWrflDnon_WeaponCheck1	Sitting,/checking weapon
AmovPsitMstpSlowWrflDnon_WeaponCheck2	Sitting/checking weapon
AmovPsitMstpSnonWnonDnon_ground	Sitting, hands back
AmovPercMstpSnonWnonDnon_AmovPsitMstpSnonWnonDnon_ground	Sit down /stand up
AmovPsitMstpSnonWnonDnon_ground_AmovPpneMstpSnonWnonDnon	Sit down /stand up

Animation	Bezeichnung
AmovPsitMstpSlowWrflDnon_AmovPercMstpSlowWrflDnon	Stands up
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_start	Lying, throwing granate
AwopPpneMstpSgthWnonDnon_throw	Lying, throwing granate
AwopPpneMstpSgthWnonDnon_end	Lying, throwing granate
AwopPercMstpSgthWrflDnon_Throw1	Throwing granate
AswmPercMrunSnonWnonDf_AswmPercMstpSnonWnonDnon	Swimming
DeadState	Dying
SprintBaseDf	Running straight ahead
SprintBaseDfl	Running left a circle
SprintBaseDfr	Running right a circle
AwopPercMstpSoptWbinDnon_rfl	Using Binoculars
AmovPercMstpSnonWnonDnon_turnL	Standing around
AmovPercMstpSnonWnonDnon_turnR	Standing around
AmovPercMstpSnonWnonDnon_Ease	If attitude assumes
AmovPercMstpSnonWnonDnon_EaseIn	If attitude assumes
AmovPercMstpSnonWnonDnon_EaseOut	If attitude assumes
AmovPercMstpSnonWnonDnon_AmovPknlMstpSnonWnonDnon	Kneeing on one knee
AmovPercMstpSsurWnonDnon	Hands behind the head
AmovPercMstpSnonWnonDnon_AmovPpneMstpSnonWnonDnon	Lying, standing up
AmovPercMstpSnonWnonDnon_AinvPknlMstpSnonWnonDnon	Putting some things down
AmovPercMstpSlowWrflDnon_AmovPsitMstpSlowWrflDnon	Sitting on the ground
AinvPknlMstpSnonWnonDnon_AmovPknlMstpSrasWpstDnon	Knees down and gasping
AinvPknlMstpSlayWrflDnon_AmovPknlMstpSrasWrflDnon	Knees down and gasping
AinvPknlMstpSlayWrflDnon_healed	Healing animation
AinvPknlMstpSlayWrflDnon_healed2	Healing animation
AinvPknlMstpSnonWnonDnon_healed_1	Healing animation
AinvPknlMstpSnonWnonDnon_healed_2	Healing animation
AinvPknlMstpSlayWrflDnon_medic	Bandaging a victim
AinvPknlMstpSnonWnonDnon_medic_1	Bandaging a victim
AinvPknlMstpSnonWnonDnon_medic_2	Bandaging a victim
AidlPknlMstpSnonWnonDnon01	Takes cover
AmovPercMrunSlowWrflDf_AmovPpneMstpSrasWrflDnon	Takes cover
AidlPercMstpSnonWnonDnon08	Shouldered the weapon
AinvPknlMstpSnonWnonDnon_1	Kneeing down, ammobox
AinvPknlMstpSnonWnonDnon_2	Kneeing down, ammobox
AinvPknlMstpSnonWnonDnon_3	Kneeing down, ammobox
AinvPknlMstpSnonWnonDnon_4	Kneeing down, ammobox
AmovPercMstpSnonWnonDnon_AwopPercMstpSoptWbinDnon	Shouldered the weapon, Taking Binoculars
AmovPercMstpSnonWnonDnon_Dancing	N/A
AmovPercMstpSnonWnonDnon_flipflop	N/A

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

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 on-screen information (hint)

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

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.

Churchill (Anton Voß)

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 Gruppe1;
_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 "script\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 Soldat1;
#START
; Checking whether Sold 1 is still alive, if not script will go to the Label END
? (!(alive Soldat1)) : goto "END";
;Set Marker
#MARKER
"S1-Symbol" setMarkerPos getPos Soldat1;
~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 Soldat1)Then{"S1-Symbol" setMarkerPos getPos Soldat1}
Else{"S1-Symbol" setMarkerType "Empty";exit};
 goto "Start";
```

Explanation:
(**If**) **Soldat1** is still alive (**Then**) set **S1-Symbol** on **Soldat1** 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 ["Own Entry", "scriptName.sqs"]

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 ["Own Entry", "scriptName.sqs"]

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 ["Own Entry", "script.sqs"]
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.

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	**firstaid.sqs**
save.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 on 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 on **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"
; Entrys 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
RIIIID = 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!

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 on **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 on **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 on **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 "AinvPknlMstpSlayWrflDnon_healed";
~1
Playsound "Sanipack";
~2
Player switchmove "AinvPknlMstpSlayWrflDnon_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 "CloseBackpack";
Player removeAction RIID;
Player removeAction RIIID;
Player removeAction RIIIID;
RID = Player addAction ["Open Backpack","backpack\backpack.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 then **1**

The script would go to **#P2** if the value is smaller then **2**

The script would go to **#P3** if the value is smaller then **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 been 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:

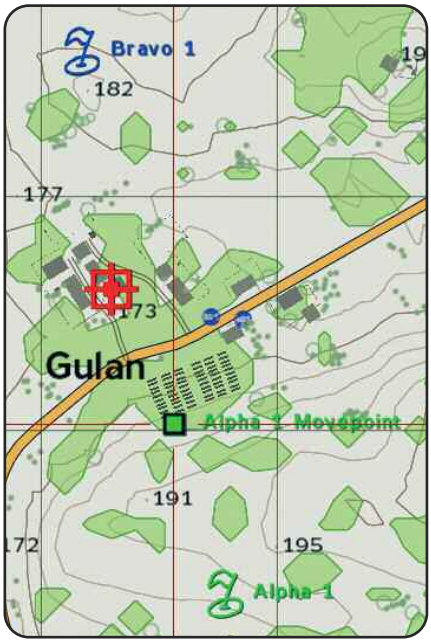
Name:	Alpha1
Initline:	Alpha1=group this
Size of the Group:	Random

Radio trigger:

Aktivierung:	Radio Alpha Repeatdetly
Axis a/b:	0
Text:	0-0-1 Alpha 1
On Activation	[] exec "skripts\alpha.sqs"

Marker:

Name:	AmoveP
Text:	Alpha 1 Movepoint
Symbol:	Dot
Axis a/b:	1



Alpha.sqs

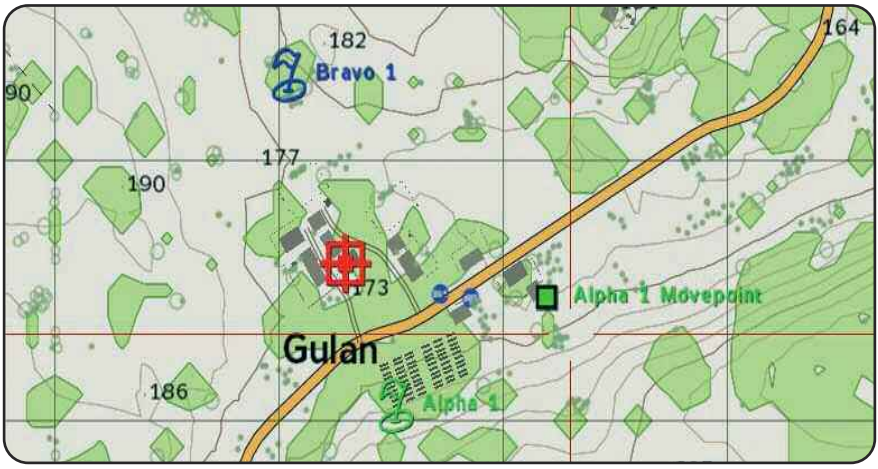
The following things will happen when this script gets executed: The variable **alphaclick** will bet set on **true** and a text line will appear on the screen which asks the player to click on the map. ArmA notes the definition at **onMapSingleClick** and the script will break at **@!alphaclick** until 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 "artillery\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 + + +
[W1,ATarget] exec "artillery\fire.sqs"
[W2,ATarget] exec "artillery\fire.sqs"
[W3,ATarget] exec "artillery\fire.sqs"
[W4,ATarget] exec "artillery\fire.sqs"
[W5,ATarget] exec "artillery\fire.sqs"
[W6,ATarget] exec "artillery\fire.sqs"
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 script call (**W6**) 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

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;
[E1,_Target] exec "enemyArty\fire.sqs";
[E2,_Target] exec "enemyArty\fire.sqs";
[E3,_Target] exec "enemyArty\fire.sqs";
[E4,_Target] exec "enemyArty\fire.sqs";
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"
```

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 - A script to 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 for 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 "End";
signalcounter = signalcounter+1;
~0.8
"Target1" setMarkerColor "ColorRed";
~0.8
"Target1" setMarkerColor "ColorBlack";
goto "Start"
#End
~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 = createGroup 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"
```

```
;=========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 "END";
goto "CHECK2"

;=========DELETE=======================

#END
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.

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.

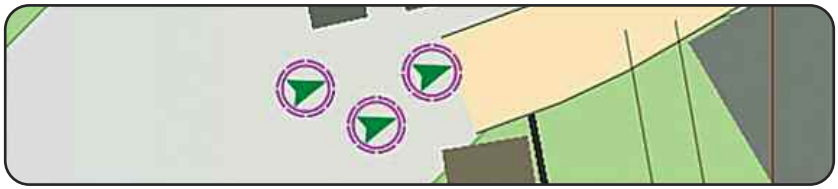
AlexXx (Alexander Zanft)

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 missions. The information given here can be used in more complex missions later on.

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.



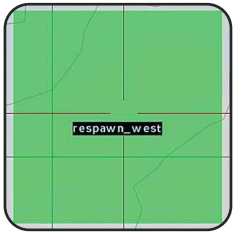
7.2 - The Respawn Points

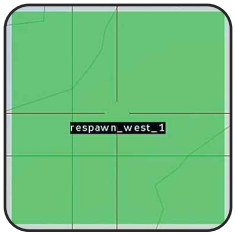
Markers are the best way to define respawn points. Respawn markers have to be renamed to match the side using the respawn point:

West:	**Respawn_west**	Resistance:	**Respawn_guerrila**
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 object 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:

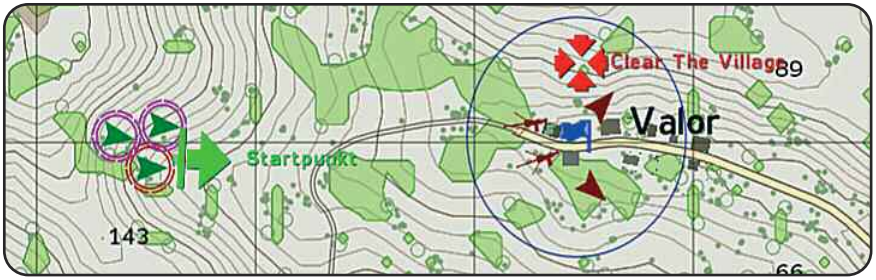
Respawn Marker

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 "Congratulation - 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: "**Congratulation - 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
AIkills=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;
AIkills=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**.

7.7 - 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 needs to become enemies of each other:

Variant 2
This syntax becomes 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.

7.6 - Defining The Multiplayer Area

Because the island is so large (it's 400 square kilometers), its 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"
```

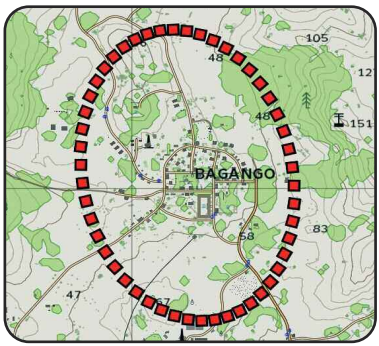
There is no need to script your own **area.sqf**. It´s already implemeted in the engine!

The object, in this case the heli-pad, is renamed to "**Area1**" automatically when the missions 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 sytnax 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 - 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;
};
```

7.9 - 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**.

7.10 - 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]

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 resapawn, 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, …)

Chapter 8
- Camscripting -

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.

Blacktiger (Simon Eichenberger)

8.1 - Controlling the Camera

It's actually quite simple to control the camera because only a few keys of 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 move to this new position. After this position has been defined it will be used in the script and finally adjusted.

Intro.sqs

```
;Intro sequenze

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

Blending black into the sequence with a time of 4 seconds.

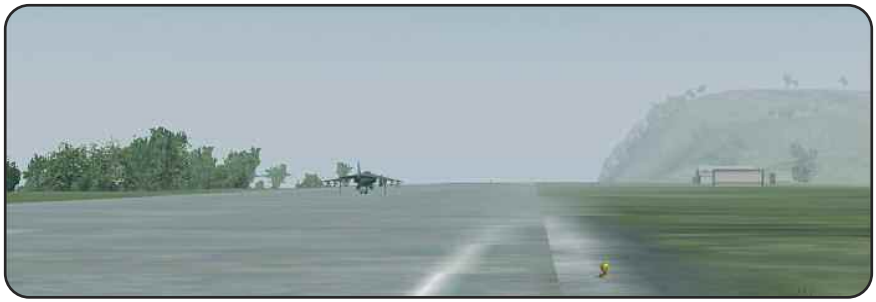
Playmusic "Track1"

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 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 blending 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 wont be audible.

159

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**".

```
;Create Camera
_camera = "camera" camCreate [0,0,0]
_camera camSetTarget Car
_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 = Car

;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 Car < 50 : goto "End"
goto "Loop"

#End
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 syntax:

titleCut ["Hello", "Black Out"]; titleFadeOut 4

or

titleText ["Test", "White In"]; titleFadeOut 4

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:

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.

Woody (Andreas Holzwarth)

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 "Variablenname"

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 on **true**, a predefined action will get started. If the same action gets set back on **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 then **b**. **A** will always receive the value **1** until it's bigger then **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 then **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 Soldat1)Then{"S1-Symbol" setMarkerPos getPos Soldat1}
                    Else{"S1-Symbol" setMarkerType "Empty";exit};
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.

```
_Wert = 0;
waitUntil {_Wert = _Wert +1; _Wert >= 100};
```

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

That's true, without any knowledge in programming! Of course, it wouldn't be bad if you have had some experience working with the Operation Flashpoint Editor. It would be an advantage but it's not necessary. This Guide will explain to you the parts of the editor individually, and many examples will help you to understand the single operations. In addition to this, the Armed Assault Editing Guide will show you the nearly unlimited possibilities which are offered to you by the Editor.

After you've worked with this guide a few times, you'll be able to create your own exciting missions. The only thing you really need is creativity and a lot of ideas which you want to realize. 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 into the game.

Create dynamic missions with different weather, time of day and furthermore, different mission targets. Units, objects or even the player himself can be located at a different position every time a mission begins. All of these possibilities shouldn't be a problem for you anymore.

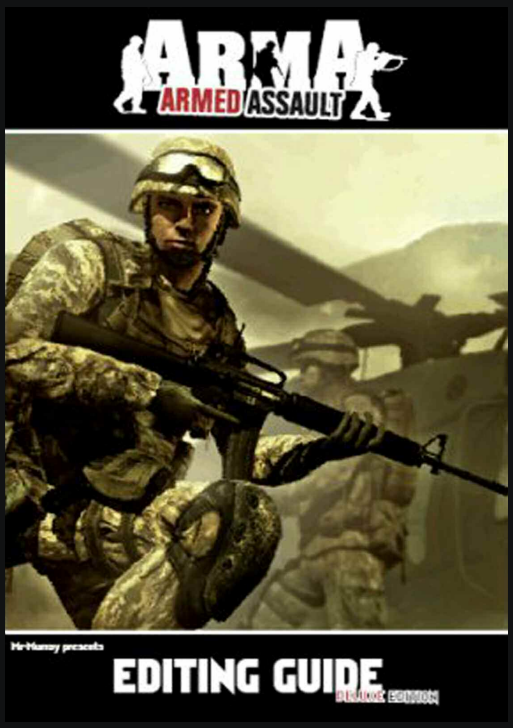
Now it's up to you and your ideas and your creativity to create new good missions. This guide doesn't contain a directing book, scenarios, or any other stories for your missions - that's all up to you. But with this guide, you have all you need to make your ideas come true. If anything doesn't work as you want it to, 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 set up as Operation Flashpoint's successor with its very own scripting language. Here and there some changes were made and many new things were used in ArmA, but the basic concept is still the same. The Editor still enables the user to keep an organized overview, and the missions folder and their contents are still the same as well. So read, try and edit your own missions with this Guide through the world of Armed-Assault.

BI, Morphicon and Mr-Murray wish you good luck and much fun with the Editor.

Available in your store soon (only in Germany!)

Armed Assault Editing Guide Deluxe Edition



The print version of the popular Editing Guide, written by Sascha "Mr-Murray" Hoffmann, will be released soon. It will be a hardcover book containing 335 pages, including over 200 subsections which have been separated in 11 chapters.

The contents of the Deluxe Edition are double that of this guide and also include important additions to all the new stuff which is possible in ArmA, Queens Gambit and the usage of dialogs.

An absolute must for every modding fan of Armed Assault whether a beginner or an advanced mission editor!

Get more information on: www.mr-murray.de.vu

Contact: mr-murray@bossmail.de