BAS f Manual

For BAS f Version: 1-3-1

Authors	Fer Kegetys kevb0
Editors	Kronzky Serclaes Messiah (aka Messiah2)
Translators	Serclaes Messiah (aka Messiah2)

Table of Contents

INTRODUCTION

A problem with mission-making, and multi-player missions in particular, is that the 'learning-curve' is steep. The mission designer has to learn about many issues, and ensure that several key components (such as briefing files, gear selection snippets, automatic removal of dead bodies etc.) are created, correctly configured and tested in order to build a quality mission with high levels of re-playability and performance.

Often the overall quality of a designer's early missions suffers because s/he is having to 're-invent the wheel', writing and testing his/her own library of personally developed scripts, and finding and learning how to use scripts and code snippets written by other designers.

BAS f is an attempt to help the new mission designer take advantage of a library of pre-tested components that will increase the quality, re-playability and performance of his/her missions, whilst allowing him/her to focus on making his/her own unique ideas come to life.

BAS f is a *framework*: an MP mission folder containing a library of scripts, functions and template files, plus a manual (this document). The framework is designed to provide the mission designer, after minimal additional configuration, with a selection of pre-tested features and functionality intended to improve the overall quality and re-playability of his/her mission.

Since BAS f is a framework, and not a template, the design of the actual mission is completely open; the framework is intended only to save time for the designer by providing components such as weather selections that work with join-in-progress (JIP), or pre-configured gear selections for re-equipping soldiers during the mission briefing. For many components care has also been taken to localise messages and texts (where used) into several languages.

Importantly, all components of the framework are fully explained in this document, feature extensive commenting within script files, and each can be disabled if desired.

This manual is designed to guide the mission designer through the full configuration process, as well as provide instructions for optional components (found in section B).

WHO IS THE FRAMEWORK FOR?

BAS f is aimed at the new ArmA mission designer, although it is not intended for complete beginners. To use BAS f the mission designer should have a *basic* understanding of:

- How to open the MP mission editor
- How to place and edit units, triggers, waypoints and markers in the editor
- The ArmA scripting syntax (for **`.sqf`** files)
- The roles of key files: **`description.ext`** and **`init.sqf`**
- The role of script files (**`.sqf`** files)

If a mission designer has already created his/her first few missions, everything in BAS f should be relatively straightforward.

In addition, mission designers with intermediate experience may also find BAS f useful as a time-saver, or as the basis for their own personal base framework(s).

BAS, and its partners, plans to continually evolve this framework, and will also explore the creation of specialised versions catering for specific mission types. For links to the most recent version of BAS f please see the BI Community wiki:

http://community.bistudio.com/wiki/BAS_f

SECTION A

CORE COMPONENTS

The following mini-guides, core components and pre-configured mission settings are included in this version of the BAS f framework:

- Select Your Island
- Naming Your Mission
- Load Screen Text
- Respawn Settings
- Configurable Playable Slots
- Conditions Selector
- Gear Snippets
- Automatic Body Removal
- Multiplayer Ending Controller
- Sample Markers
- Debug Mode
- BAS Server Logic
- BAS f Common Local Variables
- Briefing File Template
- ReadMe File Template

Some of these components require minor amounts of configuration, however the majority are ready for use without any further editing. This manual will step the mission designer through all components, and indicate any edits necessary for final configuration (including how to disable a specific component).

SELECT YOUR ISLAND

BAS f supports a wide range of official and community-generated islands in ArmA. All the files for a mission are stored within a single folder, which is specific to a particular Island. To begin using the BAS f framework you must first decide which island you want to use for you mission, and then use the appropriate folder.

Note: For some islands, the pre-placed units, markers and games logics may not be positioned over dry land by default. These items are always found in the lower left-hand side of the map.

Official Islands

Island	Template Folder	Notes
Porto	BAS_f_v1-3-1.Porto	Requires Queen's Gambit expansion for ArmA.
Rahmadi	BAS_f_v1-3-1.Intro	
Sahrani	BAS_f_v1-3-1.Sara	
South Sahrani	BAS_f_v1-3-1.SaraLite	
United Sahrani	BAS_f_v1-3-1.Sara_dbe1	Requires Queen's Gambit expansion for ArmA.

Community Islands

Island	Template Folder	Notes
Afghan Village	BAS_f_v1-3-1.afghan_village	By Opteryx, for more information please search the BI forums.
Avgani	BAS_f_v1-3-1.Avgani	By Opteryx, for more information please search the BI forums.
LDDK Training Island	BAS_f_v1-3-1.LDDK_Isle	By Goeth of LDD Kyllikki, this island is also known as Isla de Pollo
Podaga	BAS_f_v1-3-1.FDF_Isle1	Part of the FDF Mod, or more information please search the BI forums.
Sakakah Al Jawf	BAS_f_v1-3-1.Sakakah	By Opteryx, for more information please search the BI forums.
Schmalfelden	BAS_f_v1-3-1.Schmalfelden	By Nicholas Bell, for more information please search the BI forums.
Uhao	BAS_f_v1-3-1.Uhao	By OFman, for more information please search the BI forums.

ACE Islands

Island	Template Folder	Notes
ACE Nogova	BAS_f_v1-3-1.ACE_Island_Noe	
ACE Nabukonodexa	BAS_f_v1-3-1.ACE_Island_nabukonodexa	
ACE Messor	BAS_f_v1-3-1.ACE_Island_Messor	
ACE Leusderheide	BAS_f_v1-3-1.ACE_Island_leusderheide	
ACE Lake Martin	BAS_f_v1-3-1.ACE_Island_lakemartin	
ACE Ivtiliac	BAS_f_v1-3-1.ACE_Island_ivtiliac	
ACE Isla de Stella	BAS_f_v1-3-1.ACE_Island_isladestella	
ACE Occasus	BAS_f_v1-3-1.ACE_Island_Occasus	
ACE Highlands	BAS_f_v1-3-1.ACE_Island_highlands	
ACE OFP World	BAS_f_v1-3ace_island_ofp_world	
ACE Havelte	BAS_f_v1-3-1.ACE_Island_havelte	

ACE Gaia	BAS_f_v1-3-1.ACE_Island_Gaia	
ACE Freya	BAS_f_v1-3-1.ACE_Island_freya	
ACE Elephant Head	BAS_f_v1-3-1.ACE_Island_elephanthead	
ACE Everon	BAS_f_v1-3-1.ACE_Island_Eden	
ACE Clarck Island	BAS_f_v1-3-1.ACE_Island_clarkisland	
ACE Canyonda	BAS_f_v1-3-1.ACE_Island_canyonda	
ACE Kolgujev	BAS_f_v1-3-1.ACE_Island_Cain	
ACE Samak Hills	BAS_f_v1-3-1.ACE_Island_samakhills	
ACE Sandy Rocks	BAS_f_v1-3-1.ACE_Island_sandy_rocks	
ACE Saru	BAS_f_v1-3-1.ACE_Island_saru	
ACE Skye	BAS_f_v1-3-1.ACE_Island_skye	
ACE Sontonagh district	BAS_f_v1-3-1.ACE_Island_sontonagh_district	
ACE Torment Valley	BAS_f_v1-3-1.ACE_Island_torment_valley	
ACE Trinity	BAS_f_v1-3-1.ACE_Island_trinity	
ACE Uwar desert	BAS_f_v1-3-1.ACE_Island_uwar_desert	
ACE 73 Eastings	BAS_f_v1-3-1.ACE_Island_73eastings	
ACE Malden	BAS_f_v1-3ace_island_abel	
ACE Anilym	BAS_f_v1-3-1.ACE_Island_anilym	
ACE Atlantis Gold	BAS_f_v1-3ace_island_atlantis_gold	
ACE Avignon	BAS_f_v1-3-1.ACE_Island_avignon	
ACE Virovitia	BAS_f_v1-3-1.ACE_Island_virovitica	

NAMING YOUR MISSION

Before editing can commence you will want to name your mission by following these steps (make sure you have not started ArmA yet, and if necessary quit the game first):

Note: The following steps assume that the island being used is Sahrani. If you decide to use another island, the steps are the same, but the folder name will be different (e.g. **`BAS_f_v1-3-1.Intro`** for missions set on Rahmadi, or **`BAS_f_v1-3-1.Porto`** for mission set on Porto).

1. Pick a name for your mission (see notes below).

2. Before starting ArmA, copy the scenario folder **`BAS_f_v1-3-1.Sara`** and rename the copy **`YourMissionName.Sara`** (be careful not to change the letters after the '.') - you will want to put this folder into the location:

   `C:\Documents and Settings\YourName\My Documents\ArmA Other Profiles\YourName\MPMissions`

3. In the mission folder, open the file **`mission.sqm`** and look for the line:

   ```
   briefingName="*** Insert name here. ***";
   ```

   Change the line to read:

   ```
   briefingName="YourMissionName";
   ```

   This must *exactly* match the name you gave to the mission folder in step 2 (but without the **`.Sara`** part of the folder name).

4. Start ArmA, and open the scenario for editing. Do this via the menu options: **Play > Multiplayer > New > Sahrani > YourMissionName** (missions made with BAS f are designed to be edited in native MP mode).

5. In the editor, open the **Intel** dialog (by clicking on the date in the top-right) and change the value for the **Description:** field from **`MISSION DESCRIPTION`** to your chosen description. An example might be: "Clear the town of enemy troops"

6. Open the file **`description.ext`** and look for the code segment entitled:

   ```
   // BAS f - Mission Header
   ```

   You should notice that in the code segment values are set for **`gameType`**, **`minPlayers`** and **`maxPlayers`**.

7. In the version of the file **`description.ext`** that comes with BAS f, the value for **`gameType`** is set as **`Coop`**. If your mission is not going to be a co-operative one, you should change this value to **`CTF`**, **`Team`** etc.

8. The values for **`minPlayers`** and **`maxPlayers`** reflect the minimum and maximum number of human players that can participate in the mission. In the version of the file **`description.ext`** that comes with BAS f, the values for **`minPlayers`** and **`maxPlayers`** are set at **`1`** and **`10`** respectively. You should change these values to reflect the number of playable slots in your mission.

Something that you may want to do when naming your mission – and particularly when re-naming the mission folder – is follow a popular naming convention. Missions names that follow the most popular conventions can look this:

```
co_ace_10_Victory_Rose
```

Naming conventions are designed to allow players and administrators to quickly learn a lot about the mission just from the name itself. For example, the above example reveals several bits of information:

- The mission is called **Victory Rose**
- The maximum number of players is 10
- The mission is co-operative
- Addons from the ACE mod are required

With two key differences (see below), BAS suggests using the naming convention followed by makers of missions for Operation Flashpoint (OFP, the game which came before ArmA). You can read about this convention at:

http://www.flashpoint1985.com/cgi-bin/ikonboard311/ikonboard.cgi?act=ST;f=2;t=26694

Due to differences in the way that ArmA handles file and folder names, the two key differences suggested by BAS are:

1. Use underscores (_) instead of spaces ( ) in file and folder names.

2. Use dashes (-) instead of periods (.) when including version numbers in names (e.g. `v1-1` instead of `v1.1`), because ArmA will only allow one period (.) in a file name.

3. Do not use the (@) character.

LOADING SCREEN TEXT

When the mission is loading it is possible to display some text on the screen, such as the name of the mission, or the name of its designer. To use the loading screen text:

1. Open the file **description.ext** and look for the code segment entitled:

   ```
   // BAS f - Loading Screen Text
   ```

2. Edit the following line, inserting your desired text between the inverted commas:

   ```
   onLoadMission="";
   ```

   An example would be:

   ```
   onLoadMission="Get Ready...";
   ```

Please note that a mission that takes very little time to load will only display the message for a short time, and your message may not be readable in that time-frame.

To disable this component completely simply skip the steps above (in the version of the file **description.ext** that comes with BAS f, the value for **onLoadMission** is left blank).

RESPAWN SETTINGS

When a player dies several options for respawning are available, from becoming a seagull to spawning into an AI within the same group. The options available are:

1. BIRD - Respawn as a seagull.
2. INSTANT - Respawn just where you died (you will not keep your gear).
3. BASE - Respawn at base marker (these must be placed in the editor).
4. GROUP - Respawn in your group (if no AI slots are left, you become a seagull).

To create the desired type of respawn a segment of code must be placed in the **description.ext** file. In the version of the file **description.ext** that comes with BAS f, the default option for respawn is for players to become a seagull.

To change the type of respawn:

1. Open the file **description.ext** and look for the code segment entitled:

   ```
   // BAS f - Respawn Settings
   ```

2. Edit the following line, changing the value of **respawn** to the desired respawn type (see list above):

   ```
   respawn=BIRD;
   ```

3. If you have selected BASE respwn then you must ensure that markers with the following names exist (place the markers on the map where you want each side to respawn):

   ```
   respawn_west
   respawn_east
   respawn_guerrila
   respawn_civilian
   ```

4. If you have selected BASE or INSTANT respawn you can also control the time it takes for the player to move into the free AI. Edit the following line, changing the value of **respawndelay** to the desired number of seconds:

   ```
   respawndelay=3;
   ```

This component cannot be disabled. As described above, in the version of the file **description.ext** that comes with BAS f, the default option for respawn is for players to become a seagull.

CONFIGURABLE PLAYABLE SLOTS

By default, a mission created with BAS f contains 1 BLUFOR player group, with 10 units in the group. All individual units are playable. The group is pre-named **`f_GrpBLU11A`**, using the following line in each individual unit's **Init:** field:

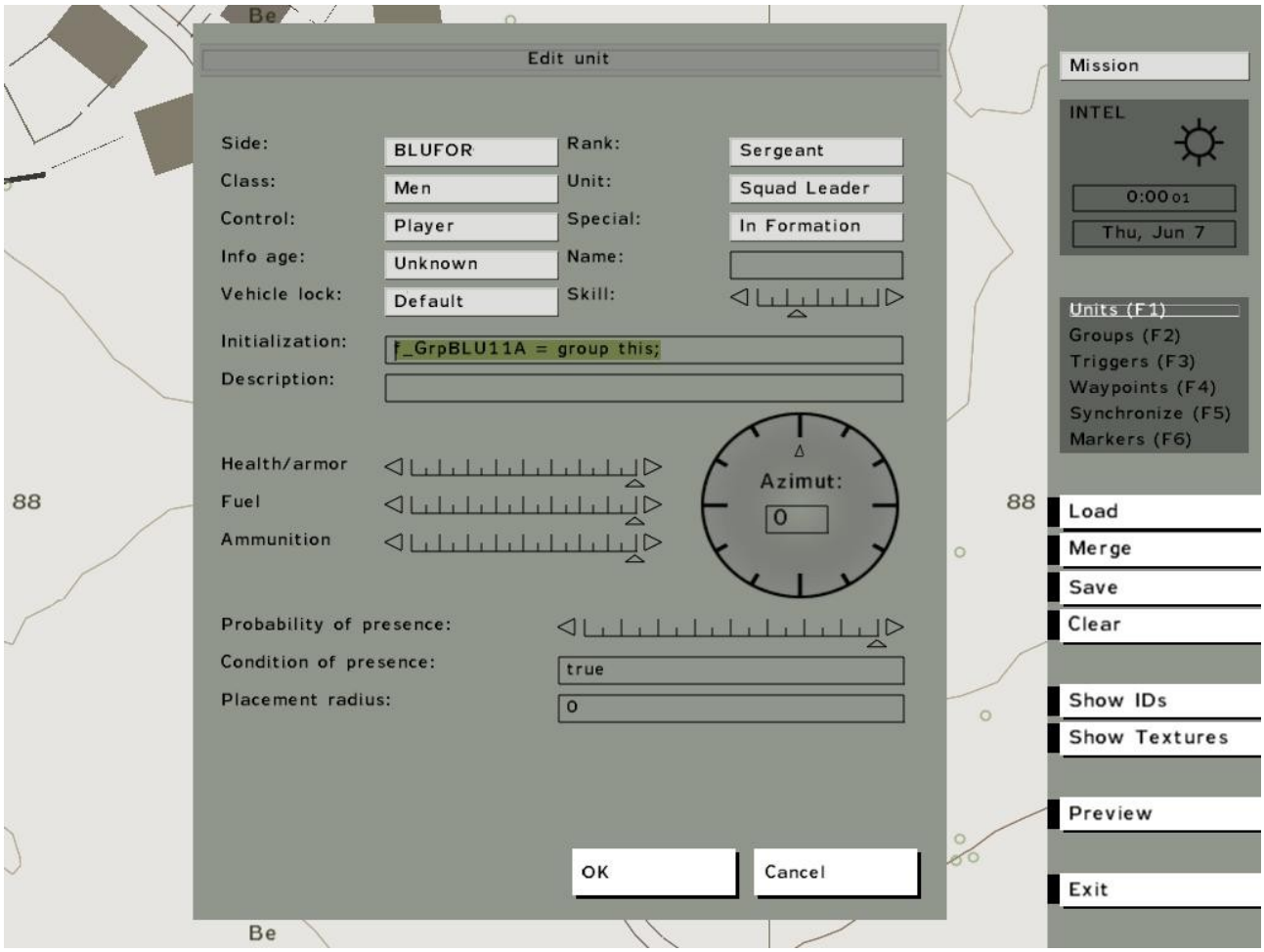
```
f_GrpBLU11A = group this;
```

The reason this line is present in the **Init:** field of every individual unit is so that regardless of whether one, some or all units are used in the mission, the group is always named **`f_GrpBLU11A`** (if the line is only placed in the **Init:** field of the group leader, the group will not be named *unless* the leader slot is used by a human player, which is not guaranteed).

Initially, the group is composed of BLUFOR soldiers from the standard ArmA core (a basic squad, plus a medic) - you can of course change this by selecting an individual in the editor, double-clicking, and altering the value of the **Unit:** drop-down. There are some important rules to remember when you are configuring the group and/or individuals:

a) Try to change individual units by editing an existing individual, since this helps to preserve the contents of the **Init:** line.

b) If you accidentally remove an individual and replace it, ensure its **Init:** field is the same as the other group members (e.g. **`f_GrpBLU11A = group this;`**).



CONDITIONS SELECTOR

In order to allow different time and weather conditions to be selected each time the mission is played, a selector is made available in the mission set-up screen. To create this selector segments of code are placed in the following files:

- **description.ext**
- **init.sqf**
- **stringtable.csv**
- **f\common\f_setMissionConditions.sqf**

By default, the following time and weather options are available:

Option	Time	Weather
21	Early Morning	Clear
22	Early Morning	Overcast
23	Early Morning	Storm
24	Early Morning	Light Fog
25	Early Morning	Heavy Fog
26	Morning	Clear
27	Morning	Overcast
28	Morning	Storm
29	Morning	Light Fog

30	Morning	Heavy Fog
1	Noon	Clear
2	Noon	Overcast
3	Noon	Storm
4	Noon	Light Fog
5	Noon	Heavy Fog
31	Afternoon	Clear
32	Afternoon	Overcast
33	Afternoon	Storm
34	Afternoon	Light Fog
35	Afternoon	Heavy Fog
36	Evening	Clear
37	Evening	Overcast
38	Evening	Storm
39	Evening	Light Fog
40	Evening	Heavy Fog
6	Dusk	Clear
7	Dusk	Overcast
8	Dusk	Storm
9	Dusk	Light Fog
10	Dusk	Heavy Fog
11	Night	Clear
12	Night	Overcast
13	Night	Storm
14	Night	Light Fog
15	Night	Heavy Fog
16	Dawn	Clear
17	Dawn	Overcast
18	Dawn	Storm
19	Dawn	Light Fog
20	Dawn	Heavy Fog
99	Debug Mode	Debug Mode

The default selection is *Noon, Clear*. The *Debug Mode* option will set the same time and weather conditions as *Noon, Clear*, but will also turn on debug mode (for more information please see the section in this document on how to use the BAS f debug mode).

The options have also been translated into English, Czech, German, Polish, Spanish, French and Russian (using text strings contained in the file **`stringtable.csv`**); players using copies of ArmA released in those languages will automatically see the options in translated form.

No configuration or editing of this component is required.

To change the default selection:

1. Open the file **description.ext** and look for the line:

   defValueParam1 = 1;

2. Change **1** for any number between **1** and **20** (see the list of options above).

To disable this component completely:

1. Open the file **description.ext** and look for the code segment entitled:

   // BAS f - Mission Conditions Selector

2. Delete everything from the above line (including the line itself), to the next instance of:

   // ==

3. Open the file **init.sqf** and look for the code segment entitled:

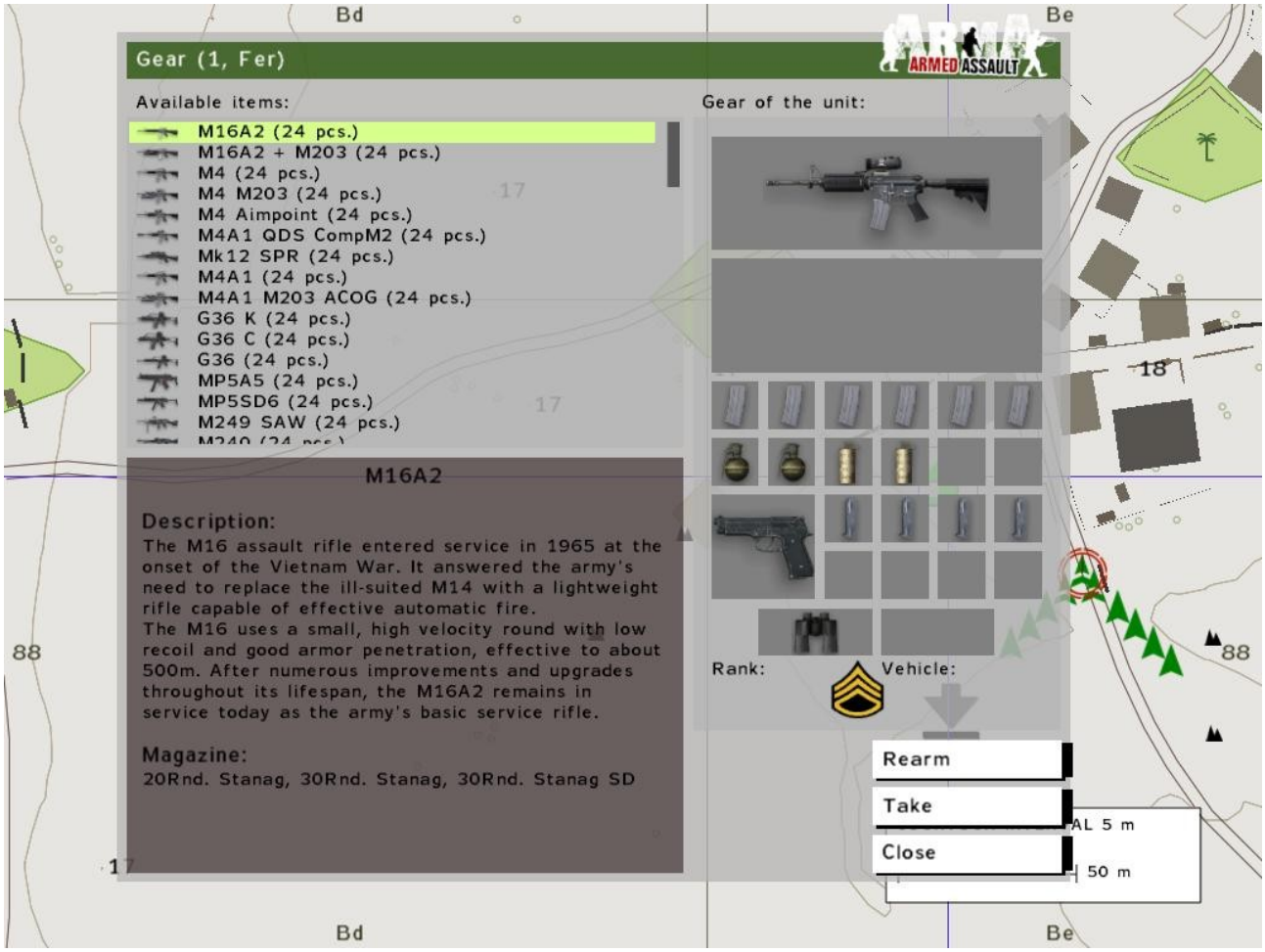
   // BAS f - Mission Conditions Selector

4. Delete everything from the above line (including the line itself), to the next instance of:

   // ==

5. In the editor, open the **Intel:** dialog (by clicking on the date in the top-right) and change the values for the **Date:** and **Time:** fields, as well as the **Weather:** and **Fog:** sliders, to reflect your desired mission conditions.

GEAR SNIPPETS



In order to allow changes to be made to the gear selection of individuals during the mission briefing, weapons, magazines and equipment must be specified as being available in the 'description.ext' file. By default, the 'description.ext' file that comes with BAS f contains references for all weapons, magazines and equipment from:

- ArmA - Weapons PBO (all gear from the basic game)

Obviously, unless modified this may allow players too much choice in terms of available weapons, magazines and equipment. In order to edit the available gear:

1. Open the file **`description.ext`** and look for the code segment entitled:

   ```
   // BAS f - Gear Snippets
   ```

   You should notice that in the first class weapons are grouped by side, and in the following class magazines are grouped correspondingly.

2. To make a weapon unavailable for selection, delete its line or comment it out by placing `//` at the start of the line. For example, to make the M4 unavailable, edit its line to:

   ```
   //    class M4          {count = 24;};
   ```

3. To make a particular magazine type unavailable for selection, use the same approach as in step #2.

4. To alter the number of weapons and magazines available, simply alter the value of

count. By default, 24 of each weapon, and 24 of each magazine type are provided. You may want to limit the availability of certain items such as sniper rifles or rockets.

When editing the weapons and ammunition code segment remember that certain weapons share magazine types. Thus, when you have decided upon the weapons that will be made available make sure that the appropriate magazine types are also available - detailed references covering weapons and their possible magazine types can be found on the BIS Community Wiki, also know as the Biki (http://community.bistudio.com/wiki/).

To disable this component completely:

1. Open the file **description.ext** and look for the code segment entitled:

   ```
   // BAS f - Gear Snippets
   ```

2. Delete everything from the above line (including the line itself), to the next instance of:

   ```
   // ======================================================================
   ```

AUTOMATIC BODY REMOVAL

In order to reduce lag in large missions a common technique is to remove dead bodies from the battlefield. This is accomplished by adding an event handler to each unit: when the unit is killed, a script is run that will pause for a certain amount of time, then delete the body.

BAS f includes a component that will automatically add such an event handler to all units in the mission. To create this functionality segments of code are placed in the following files:

- **init.sqf**
- **f\common\f_setLocalVars.sqf**
- **f\common\f_addRemoveBodyEH.sqf**
- **f\common\f_removeBody.sqf**

By default, this component is configured to remove *all* dead bodies 180 seconds (3 minutes) after the unit has been been killed.

To change length of time before a dead body is removed:

1. Open the file **init.sqf** and look for the code segment entitled:

   ```
   // BAS f - Automatic Body Remover
   ```

2. Edit the following line, changing the value of **f_removeBodyDelay** to the desired number of seconds for the delay before a body is deleted:

   ```
   f_removeBodyDelay = 180;
   ```

Because the gear on a dead body is also deleted, you may not want to apply this feature to some groups of soldiers (such as the players' group). To make a group exempt from this feature, and never delete its units' bodies:

1. Open the file **init.sqf** and look for the code segment entitled:

   ```
   // BAS f - Automatic Body Remover
   ```

2. Edit the following line, changing the value of **f_doNotRemoveBodies** from **[]** to include the name of the group(s) you want to exempt.

   ```
   f_doNotRemoveBodies = [];
   ```

   For example, to make the default players' group (which is named **f_GrpBLU11A** by default in BAS f) exempt from this feature, change the line to:

   ```
   f_doNotRemoveBodies = [f_GrpBLU11A];
   ```

   To make more than one group exempt, use commas to separate the group names:

   ```
   f_doNotRemoveBodies = [f_GrpBLU11A,GroupTwo,GroupThree];
   ```

A key limitation of this component is that it cannot automatically add the event handler to units which are created dynamically *during* the mission (for example, if you use a script to generate enemies or civilians dynamically). However, you can add the event handler by ensuring that any dynamically-created units have the following code in their **Init:** line:

```
this addEventHandler ["killed", {_this execVM "f\common\f_removeBody.sqf"}];
```

To disable this component completely:

1. Open the file **`init.sqf`** and look for the code segment entitled:

   `// BAS f - Automatic Body Remover`

2. Delete everything from the above line (including the line itself), to the next instance of:

   `// ======================================================================`

MULTIPLAYER ENDING CONTROLLER

For various reasons, it is not always easy to get a mission to end gracefully across all clients and the server - not all players see the correct debriefing (from the **briefing.html** file). The Multiplayer Ending Controller component allows a desired mission ending to be invoked on the server *after* the clients – which means all players see the appropriate debriefing. To create this functionality segments of code are placed in the following files:

- **init.sqf**
- **f\common\f_mpEndSetup.sqf**
- **f\common\f_mpEndReceiver.sqf**
- **f\server\f_mpEndBroadcast.sqf**

In your mission, whenever you want the mission to end (such as in a trigger or a script), use the following code (where **myEnd** is anything you choose, and **n** is the number of the scenario ending you wish to invoke; possible values are: **1**,**2**,**3**,**4**,**5**,**6**):

```
myEnd = [n] execVM "f\server\f_mpEndBroadcast.sqf";
```

Please note that the above code can be executed on any machine, but it will only cause the component to work (and thereby invoke an ending) if it is also run on the server.

You can put the above code in the **On Activation** field of a normal trigger, or within a custom script (.sqf file). It is important to note that if you use triggers, you should use the type **Switch**, and never the type **End1** / **End2** etc. (because the multiplayer ending controller automatically creates and uses an **End** trigger).

The component will automatically ensure the desired ending occurs on the clients first, then on the server. Your mission will now end gracefully across all machines, displaying the correct debriefing for all players (make sure your **briefing.html** file contains text for each ending you plan to use – see the Briefing File Template section of this document for more information).

If you want to make further use of the component to perform clean-up tasks, such as setting objectives to pass/fail, or invoking scripted cut-scenes when the mission ends:

1. Open the file **f\common\f_mpEndReceiver.sqf** and look for the code segment entitled:

   ```
   // CLEAN-UP OBJECTIVES & TRIGGER CUT-SCENES ETC.
   ```

2. If you read through this section, you will see spaces marked:

   ```
   // Ending #1
         case 1:
         {
   // Place any custom code for ending #1 after this line:



         };
   ```

3. There will be a space for each of the possible endings (1 to 6). Simply insert your ending-specific code in the space provided. An example is:

   ```
   // Ending #1";
         case 1:
         {
   // Place any custom code for ending #1 after this line:
               "1" objStatus "DONE";
               "2" objStatus "DONE";
   ```

```
            "3" objStatus "DONE";
        };
```

To disable this component completely:

1. Open the file **init.sqf** and look for the code segment entitled:

   // BAS f - Multiplayer Ending Controller

2. Delete everything from the above line (including the line itself), to the next instance of:

   // ==

SAMPLE MARKERS

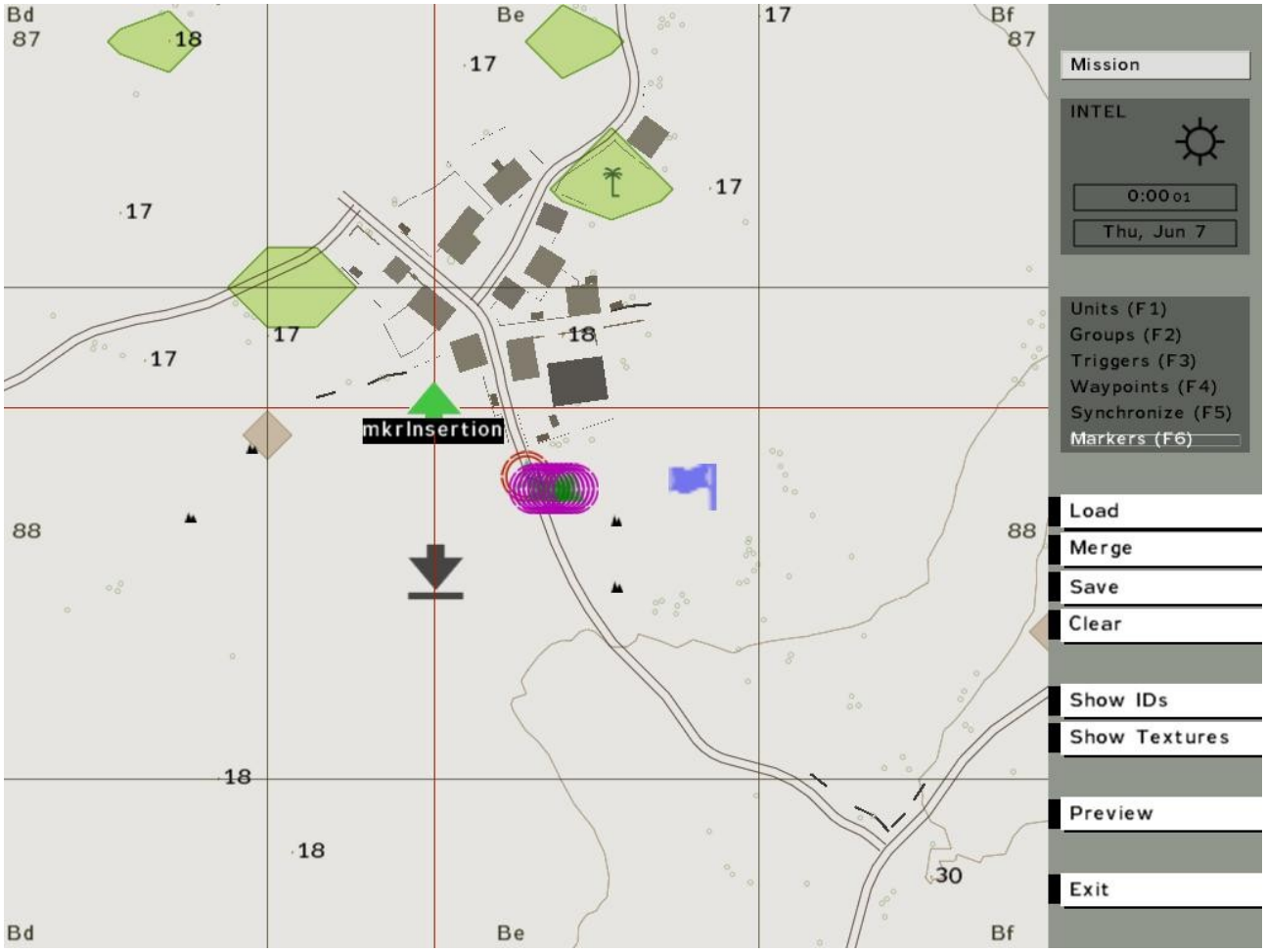
A quality mission will usually include several markers placed on the map, which help the players to understand key locations such as insertion and extraction points. As a start, the mission file that comes with BAS f contains two pre-placed markers.

To view and place the two markers:

1. In the ArmA editor press **F6** to make markers visible.

2. By the default player group you will see two markers:

   - **`mkrInsertion`** (a green up arrow)
   - **`mkrExtraction`** (a grey down arrow)

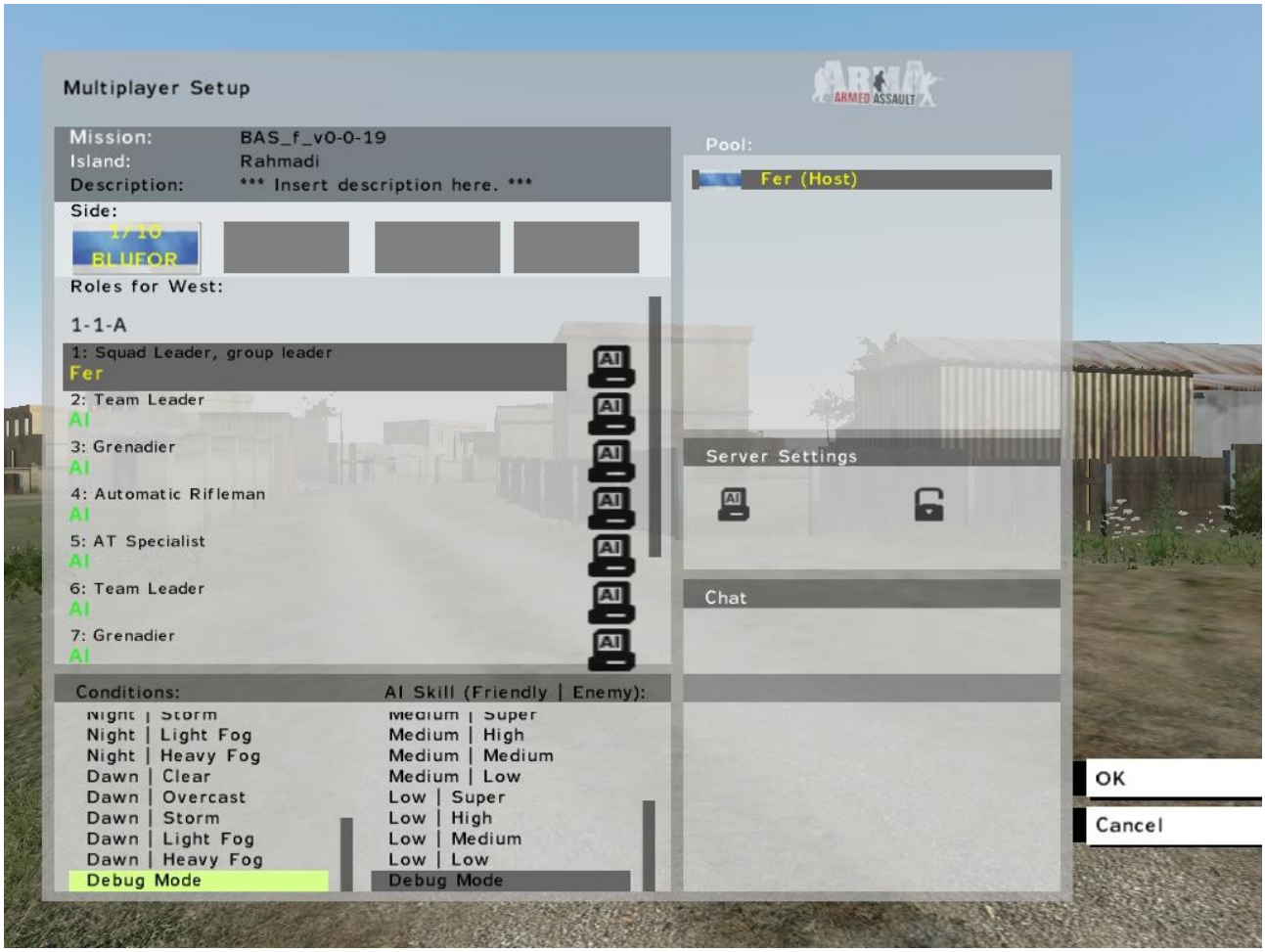


3. Click and drag a marker to change its position on the map.

Generally, the marker **`mkrInsertion`** should be placed where the players start, whilst the marker **`mkrExtraction`** should be placed where the players end the mission. However, there are absolutely no fixed rules!

Markers can also be used in the briefing – see the section of this manual covering the briefing file template for more information.

DEBUG MODE



To assist the mission designer with testing new scripts and functionality, BAS f includes a debug mode. By default, the debug mode is activated by selecting 'Debug Mode' in either the Conditions Selector or the AI Skill Selector (during mission set-up). To create this functionality segments of code are placed in the following files:

- **`init.sqf`**
- **`description.ext`**
- **`f\common\f_setAISkill.sqf`**
- **`f\common\f_setMissionConditions.sqf`**

At the start of the file init.sqf is a segment of code entitled:

```
// BAS f - Debug Mode
```

This segment of code will check to see if the value of either **`Param1`** (which is set by the Conditions Selector) or **`Param2`** (which is set by the AI Skill Selector) equals **`99`**. If this is the case, the value of a global variable, **`f_var_debugMode`**, is automatically set to **`1`** on all machines, including the server.

In your scripts you may want to include **`hint`** or **`sideChat`** commands (or other segments of code) which only execute if the mission is being run in debug mode.

To take advantage of this feature its is suggested that you use the following code in your scripts:

```
// DEBUG
if (f_var_debugMode == 1) then
{
// Place code to run ONLY in debug mode after this line:



};
```

To disable this component completely:

1. Open the file **`init.sqf`** and look for the code segment entitled:

   ```
   // BAS f - Debug Mode
   ```

2. Delete everything from the above line (including the line itself), to the next instance of:

   ```
   // ======================================================================
   ```

3. Open the file **`description.ext`** and look for the code segment entitled:

   ```
   // BAS f - Mission Conditions Selector
   ```

4. Find the line that begins:

   ```
   valuesParam1[] =
   ```

Remove the final value, **99**. Remember to remove the comma (,) before it as well.

5. Find the line that begins:

   textsParam1[] =

   Remove the final value, **$STR_f_ConditionsSelector_Option21**. Remember to remove the comma (,) before it as well.

6. Still in the file **description.ext**, look for the code segment entitled:

   // BAS f - AI Skill Selector

7. Find the line that begins:

   valuesParam2[] =

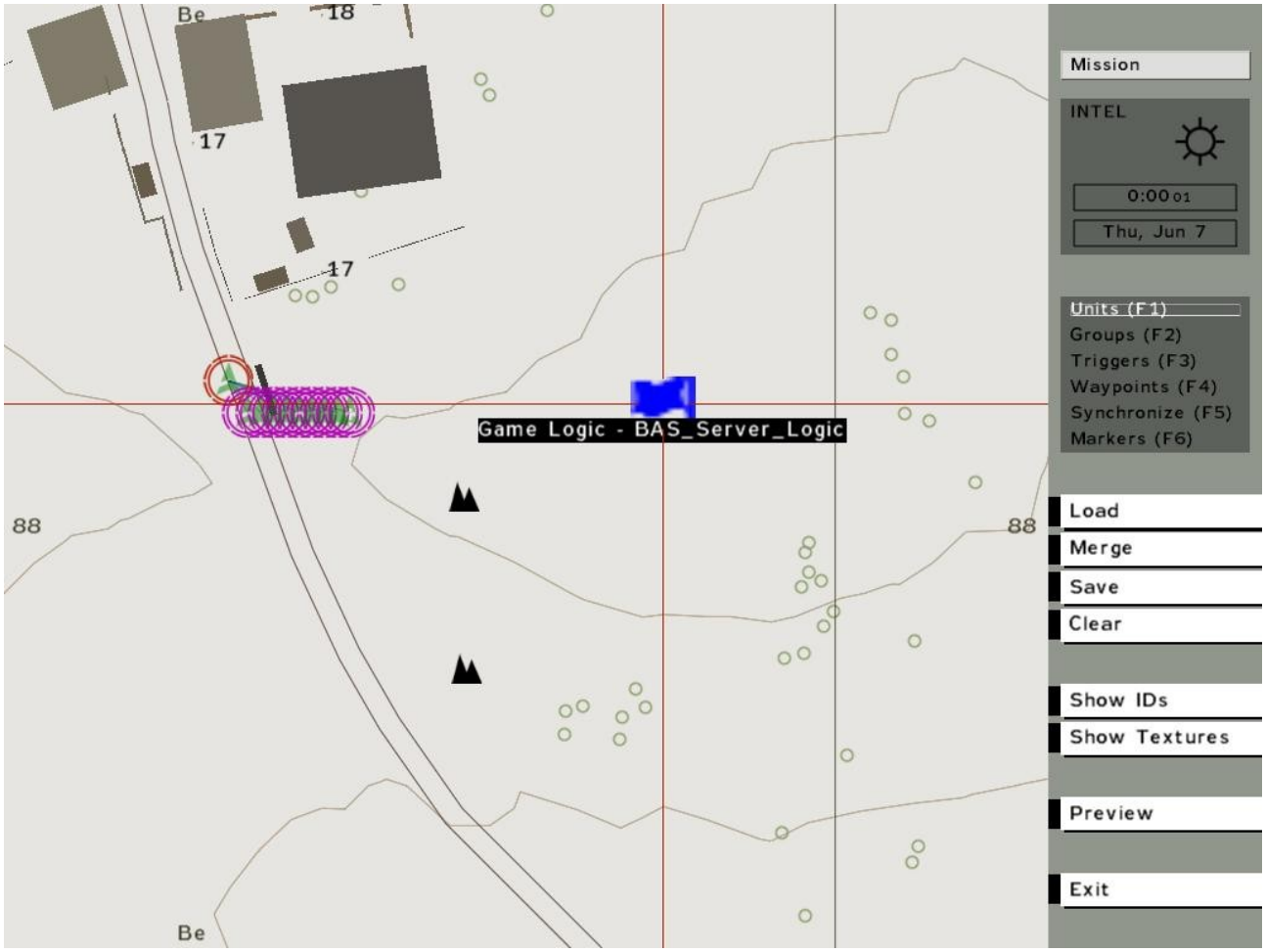
   Remove the final value, **99**. Remember to remove the comma (,) before it as well.

8. Find the line that begins:

   textsParam2[] =

   Remove the final value, **$STR_f_AISkillSelector_Option25**. Remember to remove the comma (,) before it as well.

BAS SERVER LOGIC



Whilst editing your mission you may encounter a server logic named **`BAS_Server_Logic`** placed on the map. Please do not delete this game logic, as it is required by many components.

The **`BAS_Server_Logic`** can be used to determine if a script is being executed on the server, or on a client. Within a script you can run different blocks of code on the server using the following structure:

```
if (local BAS_Server_Logic) then
{
// Place code to run ONLY on the SERVER after this line:

}
else
{
// Place code to run ONLY on CLIENTS after this line:

};
```

Note: You can also use the command **`isServer`** instead of the **`BAS_Server_Logic`**.

Please be aware that, due to the way ArmA works, the following code will *not* work (i.e. it will not end the script if it is not being executed on a server):

```
if (!local BAS_Server_Logic) then {exit;};
```

BAS F COMMON LOCAL VARIABLES

To help mission designers writing custom scripts, BAS f contains a component that automatically generates a selection of useful common local variables. These variables provide information such as arrays containing all the groups on a particular side, or all OPFOR men.

To create this functionality segments of code are placed in the following files:

- **`init.sqf`**
- **`f\common\f_setLocalVars.sqf`**

The script **`f_setLocalVars.sqf`** is executed at the start of the mission, on the server and every client machine. The variables created are accurate *only* for the local machine. This means that the precise value of any of the variables described below may differ from machine to machine. However, for most of the variables (such as arrays containing all units in the mission), the values are usually the same on all machines.

Once the script **`f_setLocalVars.sqf`** (which has the handle **`f_script_setLocalVars`** when run from the **`init.sqf`** file) has completed, the following variables are available:

Variable	Type	Description
f_var_units	Array	Contains all units, regardless of side etc.
f_var_units_BLU	Array	Contains all BLUFOR units.
f_var_units_RES	Array	Contains all resistance units.
f_var_units_OPF	Array	Contains all OPFOR units.
f_var_units_CIV	Array	Contains all civilian units.
f_var_men	Array	Contains all men, regardless of side etc.
f_var_men_BLU	Array	Contains all BLUFOR men.
f_var_men_RES	Array	Contains all resistance men.
f_var_men_OPF	Array	Contains all OPFOR men.
f_var_men_CIV	Array	Contains all civilian men.
f_var_men_players	Array	Contains all players (excluding JIP players).
f_var_groups	Array	Contains all groups, regardless of side etc.
f_var_groups_BLU	Array	Contains all BLUFOR groups.
f_var_groups_RES	Array	Contains all resistance groups.
f_var_groups_OPF	Array	Contains all OPFOR groups.
f_var_groups_CIV	Array	Contains all civilian groups.
f_var_vehicles	Array	Contains all vehicles, regardless of side etc.
f_var_vehicles_BLU	Array	Contains all BLUFOR vehicles.
f_var_vehicles_RES	Array	Contains all resistance vehicles.
f_var_vehicles_OPF	Array	Contains all OPFOR vehicles.
f_var_vehicles_CIV	Array	Contains all civilian vehicles.

To use any of these variables in a custom script:

1. Ensure your custom script does not start until the variables have been set. This is done by placing the following block of code at the start of your script (or at least before you need to use the variables):

   ```
   waitUntil {scriptDone f_script_setLocalVars};
   ```

Note: Do not disable, remove or alter the script **`f_setLocalVars.sqf`**, or stop it being called

by **`init.sqf`**, as the common local variables are essential for several core components within the BAS f framework.

BRIEFING FILE TEMPLATE

A major configuration task is to create the briefing for the mission. The briefing is contained in a separate file, **briefing.html**, which must be edited with a plain text or HTML editor. The BAS f framework comes with a template **briefing.html** file for coop missions (there is also a template for multi-side briefings – see optional component: Multi-Side Briefing File Template).

To create your briefing, open up the file **briefing.html** and complete the following sections:

- Notes
- Plan
- Debriefings
- Mission Credits

Throughout the **briefing.html** file the sections which you should edit have been labelled:

***** Insert [specific information] here. *****

Replace the text starting and ending with ******* using your own content (delete the ******* as well).

In the last section, *Mission Credits*, the suggested format for mission version is *n-n-n (DD MMM CCYY)*. An example of a mission that has reached version 1.7 on the 30th of April, 2007, would be: 1-7-0 (30 APR 2007).

The format of the **briefing.html** file is similar to HTML, although it is not exactly the same. Only a few HTML tags will work, but here are the key ones:

**
** - Will give a carriage return (new line).

**

** - Will give a blank line between paragraphs.

Text - Will create a link that, when clicked, will automatically centre the map over the marker named **mkrName** (be sure to name the marker in the ArmA editor).

If you would like to make the briefing available in more than one language, follow these steps:

1. Complete the original version of the **`briefing.html`** file in English, and save it.

2. Make a copy of the file, and rename it:

   `briefing.german.html`

   This will create a version of the file that is opened automatically by German language versions of ArmA.

3. Open the file **`briefing.german.html`** and translate your inserted texts into German.

4. Repeat steps 2 and 3 for the languages: **`Czech`**, **`Polish`**, **`French`**, **`Spanish`**, **`Italian`**, **`French`** and **`Russian`**. Note that if a non-English version of ArmA cannot find a copy of the **`briefing.YourLanguage.html`** file in its language, it will use the file **`briefing.html`** (which should be in English).

Please note that if you are creating a version in Russian, you must ensure that the file is saved in Unicode format.

The version of the file **`briefing.html`** that comes with BAS f is simple, and intended for use with co-operative missions where all players are on the same side. There is also a template for multi-side briefings – see optional component: Multi-Side Briefing File Template.

To disable this component simply delete the file **`briefing.html`** (or replace it with your own version). However, BAS recommends that all missions include a briefing!

README FILE TEMPLATE

Although not vital, it is good practice to create a readme file for the mission. This file is particularly useful if your mission requires the player to certain addons, as it should detail which addons are necessary, and where they can be downloaded from. The readme file is a separate file, **readme.txt**, which can be edited with a plain text editor. The BAS f framework comes with a template **readme.txt** file.

To create your readme, open up the file **readme.txt**. Throughout the file the sections which you should edit have been labelled:

```
[ Insert specific information here. ]
```

Replace the text starting and ending with **[ ]** using your own content (delete the **[ ]** as well).

The version of the file **readme.txt** that comes with BAS f is simple, and intended for use with co-operative missions. You should feel free to modify the template to suit your requirements.

To disable this component simply delete the file **readme.txt** (or replace it with your own version). BAS recommends that all missions include a readme file.

WHAT DO I HAVE NOW?

You do not yet have a mission. What you have is a *platform* upon which you can now build your mission. Why is the platform useful? Assuming you have used all the core components, you can already say the following things about your mission:

- It is named in a proper fashion, not just at the file/folder level, but in all the other places (such as in the editor).
- When the mission is loading, players see some meaningful custom text instead of just a blank loading screen.
- The way in which respawning is handled has been considered carefully by you, the mission designer.
- The playable slots are in groups with meaningful names (so that any third-party scripts you choose to use will be able to identify and work with the soldiers).
- The mission can be played in 20 different combinations of time and weather conditions, which means lots of re-playability.
- The weapons, ammunition and equipment use by players can be chosen prior to the mission start, although the actual selection has been considered carefully by you.
- Mission performance is greatly enhanced, as all dead bodies are automatically removed from the battlefield after a short pause.
- The handling of endings, and in particular the way in which de-briefings are presented to clients, is handled more gracefully than the default ArmA end triggers alone.
- A debug mode is available for you to use – not only with BAS f components, but with any third party scripts and functions you write or choose to import.
- The BAS Server Logic is pre-placed, giving you a quick method of checking if a script is running on the server, or on a client machine.
- Your mission features a proper briefing file.
- Your mission is accompanied by a detailed readme file that will help players and server admins to understand what addons are required (if at all), and where to get them.

Underpinning all of the above features are some key aspects:

- The components have been tested and developed specifically for MP.
- Where necessary, text strings, hints and messages have already been translated into Czech, German, Polish, Russian, Spanish, French and English.

As you build your mission, you will also be able to use the optional components (described in Section B of this manual), the ShackTactical Optional components (described in Section C of this manual), and the LDD Kyllikki Optional components (described in Section D of this manual) with the same ease and benefits as the core features listed above.

All you have to do now is write *your* mission ;)

SECTION B

OPTIONAL COMPONENTS

The following mini-guides and optional components are intended to provide your mission with extra features such as automatic team-killer punishment, or restricted vehicle crews. Each component has been pre-integrated within BAS f, but is disabled by default. Included in this version of the BAS f framework:

- AI Skill Selector (Coop Version)
- AI Skill Selector (Attack & Defend Version)
- Authorised Crew Check
- Authorised Crew Type Check
- Kegetys Spectator Script for ArmA
- Dynamic View Distance
- Multi-Side Briefing File Template
- Hide Enemy Objectives
- Casualties Cap
- Casualties Cap (Advanced)
- Automatic Body Removal (FIFO Version)
- Configurable Playable Slots (ACE Version)

AI SKILL SELECTOR (COOP VERSION)

A selector is made available in the mission set-up screen that allows players to change the relative skill levels of friendly and enemy AI units. To create this selector segments of code are placed in the following files:

- **description.ext**
- **init.sqf**
- **stringtable.csv**
- **f\common\f_setLocalVars.sqf**
- **f\common\f_setAISkill.sqf**

By default, the following skill options are available:

Option	Friendly	Enemy
6	Super	Super
7	Super	High
8	Super	Medium
9	Super	Low
11	High	Super
12	High	High
13	High	Medium
14	High	Low

16	Medium	Super
17	Medium	High
18	Medium	Medium
19	Medium	Low
21	Low	Super
22	Low	High
23	Low	Medium
24	Low	Low
99	Debug Mode	Debug Mode

You may note that the numbering sequence used above is incomplete – the missing numbers are reserved for a future version of this component.

The default selection is *Super, Super*. The *Debug Mode* option will set the same AI skill levels as *Low, Low*, but will also turn on debug mode (for more information please see the section in this document on how to use the BAS f debug mode).

Additionally, the options have also been translated into English, Czech, German, Polish, Spanish, French and Russian (using text strings contained in the file **`stringtable.csv`**); players using copies of ArmA released in those languages will automatically see the options in translated form.

To use this optional component:

1. Open the file **`init.sqf`** and look for the code segment entitled:

   ```
   // BAS f - AI Skill Selector (Coop Version)
   ```

2. Edit the following line, removing the `//` at the start:

   ```
   // [] execVM "f\common\f_setAISkill.sqf";
   ```

3. Open the file **`description.ext`** and look for the code segment entitled:

   ```
   // BAS f - AI Skill Selector (Coop Version)
   ```

4. Edit the following lines, removing the `//` at the start of each line:

   ```
   // titleParam2 = $STR_f_AISkillSelector_Title;
   // valuesParam2[] = {6,7,8,9,11,12,13,14,16,17,18,19,21,22,23,24,99};
   // defValueParam2 = 6;
   // textsParam2[] = {$STR_f_AISkillSelector_Option06, ...
   ```

Sides (**`west`**, **`resistance`**, **`east`** and **`civilian`**) are defined as *friendly* or *enemy* using four lines in the **`init.sqf`** file. In the version of the file **`init.sqf`** that comes with BAS f, the default option is for **`west`**, **`resistance`** and **`civilian`** to be *friendly*, whilst **`east`** is *enemy*.

To change the *friendly* and *enemy* designations for each side:

1. Open the file **`init.sqf`** and look for the code segment entitled:

   ```
   // BAS f – AI Skill Selector (Coop Version)
   ```

2. Edit the following line, changing the value **`f_isFriendlyBLU`** to `1` if you want the side west to be friendly, or `0` if you want west to be enemy:

```
f_isFriendlyBLU = 1;
```

3. Repeat step 2 for the variables **f_isFriendlyRES** (the **resistance** side), **f_isFriendlyOPF** (the **east** side) and **f_isFriendlyCIV** (the **civilian** side).

To change the default selection:

1. Open the file **description.ext** and look for the code segment entitled:

   ```
   // BAS f - AI Skill Selector (Coop Version)
   ```

2. Within the same code segment, look for the line:

   ```
   defValueParam2 = 6;
   ```

3. Change **6** for a number between **6** and **24** that is used in the table above.

A key limitation of this component is that it cannot automatically set the skill level of units which are created dynamically *during* the mission (for example, if you use a script to generate enemies or civilians dynamically). However, you can ensure that the skill level of any dynamically-created units is set according to the same level as other units on their side by inserting some code in their **Init:** line. The code required is different for each side:

- For **west** (BLUFOR) units, use: **this setSkill f_skillBLU;**
- For **resistance** (Independent) units, use: **this setSkill f_skillRES;**
- For **east** (OPFOR) units, use: **this setSkill f_skillOPF;**
- For **civilian** units, use: **this setSkill f_skillCIV;**

AI SKILL SELECTOR (ATTACK & DEFEND VERSION)

A selector is made available in the mission set-up screen that allows players to change the relative skill levels of BLUFOR and OPFOR AI units. To create this selector segments of code are placed in the following files:

- **description.ext**
- **init.sqf**
- **stringtable.csv**
- **f\common\f_setLocalVars.sqf**
- **f\common\f_setAISkillAD.sqf**

By default, the following skill options are available:

Option	BLUFOR	OPFOR
6	Super	Super
7	Super	High
8	Super	Medium
9	Super	Low
11	High	Super
12	High	High
13	High	Medium
14	High	Low

16	Medium	Super
17	Medium	High
18	Medium	Medium
19	Medium	Low
21	Low	Super
22	Low	High
23	Low	Medium
24	Low	Low
99	Debug Mode	Debug Mode

You may note that the numbering sequence used above is incomplete – the missing numbers are reserved for a future version of this component.

The default selection is *Super, Super*. The *Debug Mode* option will set the same AI skill levels as *Low, Low*, but will also turn on debug mode (for more information please see the section in this document on how to use the BAS f debug mode).

Additionally, the options have also been translated into English, Czech, German, Polish, Spanish, French and Russian (using text strings contained in the file **stringtable.csv**); players using copies of ArmA released in those languages will automatically see the options in translated form.

To use this optional component:

1. Open the file **init.sqf** and look for the code segment entitled:

   ```
   // BAS f - AI Skill Selector (Attack & Defend Version)
   ```

2. Edit the following line, removing the **//** at the start:

   ```
   // [] execVM "f\common\f_setAISkillAD.sqf";
   ```

3. Open the file **description.ext** and look for the code segment entitled:

   ```
   // BAS f - AI Skill Selector (Attack & Defend Version)
   ```

4. Edit the following lines, removing the **//** at the start of each line:

   ```
   // titleParam2 = $STR_f_AISkillSelector_Title_AD;
   // valuesParam2[] = {6,7,8,9,11,12,13,14,16,17,18,19,21,22,23,24,99};
   // defValueParam2 = 6;
   // textsParam2[] = {$STR_f_AISkillSelector_Option06, ...
   ```

Sides (**resistance** and **civilian**) are defined as *friendly* or *enemy* to BLUFOR using two lines in the **init.sqf** file. In the version of the file **init.sqf** that comes with BAS f, the default option is for **resistance** and **civilian** to to be *friendly* to BLUFOR.

To change the *friendly* / *enemy* designations for **resistance** and **civilian**:

1. Open the file **init.sqf** and look for the code segment entitled:

   ```
   // BAS f - AI Skill Selector (Attack & Defend Version)
   ```

2. Edit the following line, changing the value **f_isFriendlyToBLU_RES** to **0** if you want the **resistance** side to be an enemy of BLUFOR, or **1** if you it to be friendly:

```
f_isFriendlyToBLU_RES = 1;
```

3. Edit the following line, changing the value **f_isFriendlyToBLU_CIV** to **0** if you want the **civilian** side to be an enemy of BLUFOR, or **1** if you it to be friendly:

```
f_isFriendlyToBLU_CIV = 1;
```

To change the default selection:

1. Open the file **description.ext** and look for the code segment entitled:

```
// BAS f - AI Skill Selector (Attack & Defend Version)
```

2. Within the same code segment, look for the line:

```
defValueParam2 = 6;
```

3. Change **6** for a number between **6** and **24** that is used in the table above.

A key limitation of this component is that it cannot automatically set the skill level of units which are created dynamically *during* the mission (for example, if you use a script to generate enemies or civilians dynamically). However, you can ensure that the skill level of any dynamically-created units is set according to the same level as other units on their side by inserting some code in their **Init:** line. The code required is different for each side:

- For **west** (BLUFOR) units, use: **this setSkill f_skillBLU;**
- For **resistance** (Independent) units, use: **this setSkill f_skillRES;**
- For **east** (OPFOR) units, use: **this setSkill f_skillOPF;**
- For **civilian** units, use: **this setSkill f_skillCIV;**

AUTHORISED CREW CHECK

One way of *encouraging* players to stick to their designated roles is to prevent certain units from acting as pilots / drivers / gunners or commanders of specialist vehicles such as tanks and aircraft.

The Authorised Crew Check component is a quick and easy way of ensuring that only players in selected slots are able to act as crew-members on particular vehicles. If a player that is not authorised to act as crew attempts to get in as anything except cargo (passenger), s/he will be automatically ejected with a warning message.

Additionally, the warning message has also been translated into English, Czech, German, Polish, Spanish, French and Russian (using text strings contained in the file **`stringtable.csv`**); players using copies of ArmA released in those languages will automatically see the translated warning.

To create this functionality segments of code are placed in the following files:

- **`init.sqf`**
- **`stringtable.csv`**
- **`f\common\f_isAuthorisedCrew.sqf`**

To activate this component for the a vehicle:

1. In the ArmA editor, select the vehicle and ensure it has a name in the **Name:** field (such as **`MyTank`**).

2. Still in the editor, make sure the unit (soldier) or units (soldiers) that you want to have

as the authorised crew-members also have unique entries in their **Name:** field (such as **MyDriver**, **MyCommander** and **MyGunner**).

3. Open the file **init.sqf** and look for the code segment entitled:

   ```
   // BAS f - Authorised Crew Check
   ```

4. Edit the following line, removing the **//** at the start, replacing **VehicleName** with the name of the vehicle, and the array **[UnitName1,UnitName2]** with an array containing the names of the authorised crew members.

   ```
   // VehicleName addEventhandler ["GetIn", {[_this,[UnitName1,UnitName2]]
   execVM "f\common\f_isAuthorisedCrew.sqf"}];
   ```

   For example, to ensure that only **MyDriver**, **MyCommander** and **MyGunner** can act as the crew of **MyTank**, change the line to:

   ```
   MyTank addEventhandler ["GetIn", {[_this,[MyDriver,MyCommander,MyGunner]]
   execVM "f\common\f_isAuthorisedCrew.sqf"}];
   ```

5. To apply this feature to other vehicle repeat steps 1-4.

There is a known limitation with this feature: if a unit enters a vehicle as a passenger, s/he can sometimes use the action menu to move to a pilot / driver / commander / gunner position.

AUTHORISED CREW TYPE CHECK

The Authorised Crew Type Check component works in exactly the same way as the Authorised Crew Check component, but instead of checking for named player slots, it checks to see if the player is of a certain *type* (such as pilot or armoured vehicle crew).

To create this functionality segments of code are placed in the following files:

- **`init.sqf`**
- **`stringtable.csv`**
- **`f\common\f_isAuthorisedCrewType.sqf`**

To activate this component for the a vehicle:

1. In the ArmA editor, select the vehicle and ensure it has a name in the **Name:** field (such as **`MyTank`**).

2. Still in the editor, make sure the unit (soldier) or units (soldiers) that you want to have as the authorised crew-members are of types such as **Pilot** or **Crew**.

3. Open the file **`init.sqf`** and look for the code segment entitled:

   ```
   // BAS f - Authorised Crew Type Check
   ```

4. Edit the following line, removing the **`//`** at the start, replacing **`VehicleName`** with the name of the vehicle, and the array **`["UnitType1","UnitType2"]`** with an array containing the unit types authorised to act as crew.

   ```
   // VehicleName addEventhandler ["GetIn", {[_this,
   ["UnitType1","UnitType2"]] execVM "f\common\f_isAuthorisedCrewType.sqf"}];
   ```

   For example, to ensure that only units of the type **`SoldierWCrew`** can act as the crew of **`MyTank`**, change the line to:

   ```
   MyTank addEventhandler ["GetIn", {[_this,["SoldierWCrew"]] execVM
   "f\common\f_isAuthorisedCrewType.sqf"}];
   ```

   Note: Remember to put **`""`** around the unit types.

5. To apply this feature to other vehicle repeat steps 1-4.

Common unit types that you may want to use are:

Type	Description
SoldierWCrew	BLUFOR armoured vehicle crew.
SoldierWPilot	BLUFOR pilot.
SoldierGCrew	Resistance armoured vehicle crew.
SoldierGPilot	Resistance pilot.
SoldierECrew	OPFOR armoured vehicle crew.
SoldierEPilot	OPFOR pilot.

There is a known limitation with this feature: if a unit enters a vehicle as a passenger, s/he can sometimes use the action menu to move to a pilot / driver / commander / gunner position.

KEGETYS SPECTATOR SCRIPT FOR ARMA

The spectator script component allows dead players to spectate other (still living) players, replacing the default seagull mode. Features include:

- Free, chase, flyby, top-down and 1st person cameras
- Automatic display of all units in the mission
- Camera control with mouse and keyboard shortcuts
- Unit tags (Colored dots above units) and 3D bullet path indicators (with client addon)
- Night vision and missile camera
- Drop camera feature (Mouse + WSAD keys to move camera)
- Clickable minimap and full screen map with markers indicating unit positions and weapons fire
- Butterfly mode

To create this functionality segments of code are placed in the following files:

- **description.ext**
- **init.sqf**
- **onPlayerRespawnAsSeagull.sqs (onPlayerRespawnAsSeagull.xxx)**
- **f\common\f_spect\common.hpp**
- **f\common\f_spect\specta.sqf**
- **f\common\f_spect\specta_events.sqf**
- **f\common\f_spect\specta_init.sqf**
- **f\common\f_spect\spectating.hpp**

To activate this component:

1. Open the file **`init.sqf`** and look for the code segment entitled:

   ```
   // BAS f - Kegetys Spectator Script for ArmA
   ```

2. Edit the following line, removing the `//` at the start:

   ```
   // [] execVM "f\common\f_spect\specta_init.sqf";
   ```

3. In the mission folder, change the name of the file:

   `onPlayerRespawnAsSeagull.xxx`

   to:

   `onPlayerRespawnAsSeagull.sqs`

   Note: This is an SQS file, not an SQF file.

Restricting visible sides (Coop missions)

By default, the spectator script will track all units in the mission. If you would like to restrict which sides are tracked (i.e. visible), follow these steps:

1. Open the file **`f\common\f_spect\specta_init.sqf`** and edit the following line, removing the `//` at the start:

   ```
   // KEGsShownSides = [west, east, resistance, civilian];
   ```

2. Remove the names of the sides which you do not want visible. For example, if you only want it to be possible for the dead players to see west and civilian units, change the line to:

   ```
   KEGsShownSides = [west, civilian];
   ```

Restricting visible sides (Attack & Defend missions)

If you would like to restrict which sides are tracked (i.e. visible), based on the side of the player, follow these steps:

1. Open the file **`f\common\f_spect\specta_init.sqf`** and find the following lines, removing the `//` at the start of each one:

   ```
   // if (side player == west) then {KEGsShownSides = [west];};
   // if (side player == east) then {KEGsShownSides = [east];};
   // if (side player == resistance) then {KEGsShownSides = [resistance];};
   // if (side player == civilian) then {KEGsShownSides = [civilian];};
   ```

DYNAMIC VIEW DISTANCE

The view distance used by a player can have a dramatic effect on two aspects of ArmA:

- Performance – high view distances will cause low-powered graphics cards to really struggle, and result in poor FPS; low view distances have the opposite effect.

- Player effectiveness – a low view distance will hinder the effectiveness of a player in the role of fast jet pilot, whereas for a player in an infantry role this would not be an issue.

The Dynamic View Distance component allows the mission designer to set a standard view distance for all players, with special view distances for players in the roles of pilots or gunners for helicopters and fixed-wing aircraft, or commanders, drivers and gunners in tanks. The component automatically switches a player's view distance to the appropriate value, depending on whether s/he is in a pilot / gunner / commander / driver slot.

Example: a player's view distance will be normal when s/he is standing on the ground next to a fast jet, but will be changed to a special value the moment s/he gets into the pilot seat – and then changed back to normal when s/he dismounts again.

To create this functionality segments of code are placed in the following files:

- **init.sqf**
- **f\common\f_setLocalVars.sqf**
- **f\common\f_addSetViewDistanceEHs.sqf**
- **f\common\f_setViewDistanceGetIn.sqf**
- **f\common\f_setViewDistanceGetOut.sqf**

To activate this component:

1. Open the file **init.sqf** and look for the code segment entitled:

   ```
   // BAS f - Dynamic View Distance
   ```

2. Edit the following line, changing the value of **f_viewDistance_default** to the desired view distance for units which are *not* pilots or gunners in helicopters or planes, such as ground infantry or passengers (a default suggested value is 1250):

   ```
   f_viewDistance_default = 1250;
   ```

3. Edit the following line, changing the value of **f_viewDistance_tank** to the desired view distance for units which are commanders, drivers or gunners in a tank (a default suggested value is 2000):

   ```
   f_viewDistance_tank = 2000;
   ```

4. Edit the following line, changing the value of **f_viewDistance_rotaryWing** to the desired view distance for units which are pilots or gunners in helicopters (a default suggested value is 2500):

   ```
   f_viewDistance_rotaryWing = 2500;
   ```

5. Edit the following line, changing the value of **f_viewDistance_fixedWing** to the desired view distance for units which are pilots or gunners in planes (a default suggested value is 5000):

   ```
   f_viewDistance_fixedWing = 5000;
   ```

6. Edit the following line, removing the **//** at the start:

```
// [] execVM "f\common\f_addSetViewDistanceEHs.sqf";
```

There is a known limitation with this feature: if a unit enters a vehicle as a passenger, s/he can sometimes use the action menu to move to a pilot / driver / commander / gunner position without having his/her view distance changed.

MULTI-SIDE BRIEFING FILE TEMPLATE

For team vs. team missions, it is often useful to present different mission briefings to each side. This is easy to accomplish, but requires the use of a slightly different version of the **briefing.html** template.

The BAS f framework comes with a template **briefing.html** file designed for briefing different sides. To use this template:

1. Delete the file **briefing.html** in the mission folder.

2. Change the name of the file **briefing_bySide.html** to **briefing.html**

To create your briefing, open up the file **briefing.html** and complete the following sections:

- Notes - BLUFOR (if you have playable BLUFOR slots)
- Notes - OPFOR (if you have playable OPFOR slots)
- Notes - Resistance (if you have playable Resistance slots)
- Notes – Civilian (if you have playable Civilian slots)
- Plan - BLUFOR (if you have playable BLUFOR slots)
- Plan - OPFOR (if you have playable OPFOR slots)
- Plan - Resistance (if you have playable Resistance slots)
- Plan – Civilian (if you have playable Civilian slots)
- Debriefings
- Mission Credits

Throughout the new version of the **briefing.html** file the sections which you should edit have been labelled:

```
*** Insert [specific information] here. ***
```

Replace the text starting and ending with ******* using your own content (delete the ******* as well).

In the last section, *Mission Credits*, the suggested format for mission version is *n-n-n (DD MMM CCYY)*. An example of a mission that has reached version 1.7 on the 30th of April, 2007, would be: 1-7-0 (30 APR 2007).

The format of the **briefing.html** file is similar to HTML, although it is not exactly the same. Only a few HTML tags will work, but here are the key ones:

**
** - Will give a carriage return (new line).

**

** - Will give a blank line between paragraphs.

Text - Will create a link that, when clicked, will automatically centre the map over the marker named **mkrName** (be sure to name the marker in the ArmA editor).

If you would like to make the briefing available in more than one language, follow these steps:

1. Complete the original version of the **briefing.html** file in English, and save it.

2. Make a copy of the file, and rename it:

   briefing.german.html

   This will create a version of the file that is opened automatically by German language versions of ArmA.

3. Open the file **briefing.german.html** and translate your inserted texts into German.

4. Repeat steps 2 and 3 for the languages: **Czech**, **Polish**, **French**, **Spanish**, **Italian**, **French** and **Russian**. Note that if a non-English version of ArmA cannot find a copy of the **briefing.YourLanguage.html** file in its language, it will use the file **briefing.html** (which should be in English).

Please note that if you are creating a version in Russian, you must ensure that the file is saved in Unicode format.

Side-specific objectives

Each objective has a unique number assigned to it in a **briefing.html** file:

```
<a name = "OBJ_17"></a>*** Insert Resistance objective #5 here. ***</p>
```

In the above example, the objective has the index number **17**. In the **briefing_bySide.html** file you will notice that the example objectives for BLUFOR are numbered 1-6, the ones for OPFOR 7-12 etc. This is because objectives are not naturally side-specific – unless an objective is specially hidden it will be visible to players on all sides.

To have different objectives for each side, you must use different index numbers for each side's objectives, and then hide objectives for the player's enemy side(s). The Hide Enemy Objectives optional component of BAS f is designed to do exactly that.

HIDE ENEMY OBJECTIVES

Due to the way the ArmA reads the **briefing.html** file, some additional work is required to ensure that a player only sees the objectives for his/her side. The Hide Enemy Objectives component is designed to work with the Multi-Side Briefing File Template optional component included with BAS f (see previous section of this manual). The Hide Enemy Objectives component automatically senses the side of the player, and hide all other sides' objectives.

To create this functionality segments of code are placed in the following files:

- **init.sqf**
- **f\common\f_hideEnemyObjectives.sqf**

To activate this component:

1. Open the file **init.sqf** and look for the code segment entitled:

   ```
   // BAS f - Hide Enemy Objectives
   ```

2. Edit the following line, ensuring the array **f_objectives_BLU** contains all the BLUFOR objective index numbers (each number must be surrounded by "" marks):

   ```
   f_objectives_BLU = ["1","2","3","4","5","6"];
   ```

3. Edit the following line, ensuring the array **f_objectives_OPF** contains all the OPFOR objective index numbers (each number must be surrounded by "" marks):

   ```
   f_objectives_OPF = ["7","8","9","10","11","12"];
   ```

4. Edit the following line, ensuring the array **f_objectives_RES** contains all the Resistance objective index numbers (each number must be surrounded by "" marks):

   ```
   f_objectives_RES = ["13","14","15","16","17","18"];
   ```

5. Edit the following line, ensuring the array **f_objectives_CIV** contains all the Civilian objective index numbers (each number must be surrounded by "" marks):

   ```
   f_objectives_CIV = ["19","20","21","22","23","24"];
   ```

6. Edit the following line, removing the **//** at the start:

   ```
   // #include "f\common\f_hideEnemyObjectives.sqf"
   ```

   Note: There is no ; at the end of this line (because it is an **#include**).

CASUALTIES CAP

The Casualties Cap component automatically senses the percentage of casualties taken by a group (or several groups), and triggers an ending when that threshold is reached.

To create this functionality segments of code are placed in the following files:

- **init.sqf**
- **f\server\f_endOnCasualtiesCap.sqf**

To activate this component:

1. Open the file **init.sqf** and look for the code segment entitled:

   ```
   // BAS f - Casualties Cap
   ```

2. Edit the following line, removing the // at the start:

   ```
   // [[f_GrpBLU11A],100,1] execVM "f\server\f_endOnCasualtiesCap.sqf";
   ```

3. On the same line, replace **f_GrpBLU11A** with the name of the group you want to monitor, for example (group is called **MyGrp1**):

   ```
   // [[MyGrp1],100,1] execVM "f\server\f_endOnCasualtiesCap.sqf";
   ```

   If you want to monitor more than one group, ensure they are separated by a comma, for example (groups are called **MyGrp1** and **MyGrp2**):

   ```
   // [[MyGrp1,MyGrp2],100,1] execVM "f\server\f_endOnCasualtiesCap.sqf";
   ```

4. On the same line, replace **100** with the percentage of casualties the group (or groups) will take in order to trigger the desired ending. For example (percentage is 30%):

   ```
   // [[MyGrp1],30,1] execVM "f\server\f_endOnCasualtiesCap.sqf";
   ```

5. On the same line, replace **1** with the desired ending. For example (ending is 3):

   ```
   // [[MyGrp1],30,3] execVM "f\server\f_endOnCasualtiesCap.sqf";
   ```

This component is very useful in Attack & Defend missions, as it can be run for more than one group (or several groups) at the same time, triggering a different ending for each. For example, if you are using the ShackTactical: Baseline Mission File Template component, you might want to use these lines to trigger endings 1 and 2 depending on which side (BLUFOR or OPFOR) is the first to take 30% casualties (note line-wrapping):

```
[[GrpBLU_1Plt_PltHQ,GrpBLU_1Plt_Alpha,GrpBLU_1Plt_A1,GrpBLU_1Plt_A2,GrpBLU
_1Plt_A3,GrpBLU_1Plt_Bravo,GrpBLU_1Plt_B1,GrpBLU_1Plt_B2,GrpBLU_1Plt_B3,Gr
pBLU_1Plt_Charlie,GrpBLU_1Plt_C1,GrpBLU_1Plt_C2,GrpBLU_1Plt_C3,GrpBLU_1Plt
_Delta,GrpBLU_1Plt_D1,GrpBLU_1Plt_D2,GrpBLU_1Plt_D3,GrpBLU_1Plt_Echo,GrpBL
U_1Plt_E1,GrpBLU_1Plt_E2,GrpBLU_1Plt_E3,GrpBLU_1Plt_Fox,GrpBLU_1Plt_F1,Grp
BLU_1Plt_F2,GrpBLU_1Plt_F3],30,1] execVM
"f\server\f_endOnCasualtiesCap.sqf";

[[GrpOPF_1Plt_PltHQ,GrpOPF_1Plt_Alpha,GrpOPF_1Plt_A1,GrpOPF_1Plt_A2,GrpOPF
_1Plt_A3,GrpOPF_1Plt_Bravo,GrpOPF_1Plt_B1,GrpOPF_1Plt_B2,GrpOPF_1Plt_B3,Gr
pOPF_1Plt_Charlie,GrpOPF_1Plt_C1,GrpOPF_1Plt_C2,GrpOPF_1Plt_C3,GrpOPF_1Plt
_Delta,GrpOPF_1Plt_D1,GrpOPF_1Plt_D2,GrpOPF_1Plt_D3,GrpOPF_1Plt_Echo,GrpOP
F_1Plt_E1,GrpOPF_1Plt_E2,GrpOPF_1Plt_E3,GrpOPF_1Plt_Fox,GrpOPF_1Plt_F1,Grp
OPF_1Plt_F2,GrpOPF_1Plt_F3],30,2] execVM
"f\server\f_endOnCasualtiesCap.sqf";
```

CASUALTIES CAP (ADVANCED)

The Casualties Cap (Advanced) component automatically senses the percentage of casualties taken by a group (or several groups), and then executes custom code (written by the mission designer) when that threshold is reached.

To create this functionality segments of code are placed in the following files:

- **init.sqf**
- **f\server\f_casualtiesCapAdv.sqf**

To activate this component:

1. Open the file **init.sqf** and look for the code segment entitled:

   ```
   // BAS f - Casualties Cap (Advanced)
   ```

2. Edit the following line, removing the **//** at the start:

   ```
   // [[f_GrpBLU11A],100] execVM "f\server\f_casualtiesCapAdv.sqf";
   ```

3. On the same line, replace **f_GrpBLU11A** with the name of the group you want to monitor, for example (group is called **MyGrp1**):

   ```
   // [[MyGrp1],100] execVM "f\server\f_casualtiesCapAdv.sqf";
   ```

   If you want to monitor more than one group, ensure they are separated by a comma, for example (groups are called **MyGrp1** and **MyGrp2**):

   ```
   // [[MyGrp1,MyGrp2],100] execVM "f\server\f_casualtiesCapAdv.sqf";
   ```

4. On the same line, replace **100** with the percentage of casualties the group (or groups) will take in order to trigger the custom code. For example (percentage is 30%):

   ```
   // [[MyGrp1],30] execVM "f\server\f_casualtiesCapAdv.sqf";
   ```

5. Open the file **f\server\f_casualtiesCapAdv.sqf** and look for the code segment entitled:

   ```
   // CUSTOM CODE
   ```

6. Replace the following line with your custom code, removing the **//** at the start:

   ```
   // Replace me with your custom code (remember to delete the "//"
   characters).;
   ```

   Your custom code can be 1 or more lines; example, if you wanted to set some variables and trigger another script, your custom code might be:

   ```
   _myVariableA = 1;
   _myVariableB = 0;
   [] execVM "anotherScript.sqf";
   ```

Rather than forcing a specific ending (like the original Casualties Cap component), this component allows the mission designer to accomplish other tasks when the casualties threshold is met – such as setting variables, or triggering other scripts.

NOTE: This component *only* runs on the server, not on the clients, which means the mission designer's custom code is only executed on the server.

AUTOMATIC BODY REMOVAL (FIFO VERSION)

The Automatic Body Removal component described in section A of this manual uses a technique which adds an event handler to each unit; when the unit is killed, a script is run that will pause for a certain amount of time, then delete the body. One drawback to this approach is that bodies can start to disappear in front of players, which is not very sophisticated.

Another method for removing bodies is called FIFO – which stands for "First In, First Out". This approach also adds an event handler to each unit; however, when the unit is killed, it is added to an array containing all the dead bodies. As the array fills up, the first unit to enter it is removed from the mission (imagine a conveyor belt). The FIFO approach means that, hopefully, when a body disappears, it is less likely to be visible to players.

To further ensure that dead bodies do not disappear in front of players, the FIFO approach also lets the mission maker define how far a body must be from the player before it is removed.

To create this functionality segments of code are placed in the following files:

- **init.sqf**
- **f\server\f_abrFIFO.sqf**
- **f\common\f_setLocalVars.sqf**
- **f\common\f_addRemoveBodyEH.sqf**
- **f\common\f_abrAddToFIFO.sqf**

To activate this component:

1. Open the file **init.sqf** and look for the code segment entitled:

   ```
   // BAS f - Automatic Body Remover
   ```

2. Delete everything from the above line (including the line itself), to the next instance of:

   ```
   // ======================================================================
   ```

3. Look for the code segment entitled:

   ```
   // BAS f - Automatic Body Remover (FIFO Version)
   ```

4. Edit the following lines, removing the // at the start of each:

   ```
   // f_abrFIFOlength = 30;
   // f_abrDistance = 150;
   // f_abrFIFOmaxLength = 50;
   // f_doNotRemoveBodies = [];
   // ["fifo"] execVM "f\common\f_addRemoveBodyEH.sqf";
   // [] execVM "f\server\f_abrFIFO.sqf";
   ```

By default, this component is configured to starting removing dead bodies after 30 units have been been killed (30 is the size of the FIFO array). Bodies will also not be removed until they are 150m from the nearest player, unless the FIFO array contains 50 or more dead bodies – in which case the removal will commence regardless. Additionally, all groups (including player groups) will have their bodies removed - there are no exceptions by default.

To change these default settings:

1. Edit the following line, changing the value of f_**abrFIFOlength = 30;** to the desired number of bodies which much be in the FIFO array before a body is deleted:

   ```
   f_abrFIFOlength = 30;
   ```

2. Edit the following line, changing the value of **`f_abrDistance = 150;`** to the desired distance (in metres) a body must be from the nearest player before it is deleted:

   ```
   f_abrDistance = 150;
   ```

3. Edit the following line, changing the value of **`f_abrFIFOmaxLength = 50;`** to the desired number of bodies which much be in the FIFO array for bodies to start being removed regardless of distance to the nearest player:

   ```
   f_abrFIFOmaxLength = 50;
   ```

Because the gear on a dead body is also deleted, you may not want to apply this feature to some groups of soldiers (such as the players' group). To make a group exempt from this feature, and never delete its units' bodies:

1. Edit the following line, changing the value of **`f_doNotRemoveBodies`** from **`[]`** to include the name of the group(s) you want to exempt.

   ```
   f_doNotRemoveBodies = [];
   ```

   For example, to make the default players' group (which is named **`f_GrpBLU11A`** by default in BAS f) exempt from this feature, change the line to:

   ```
   f_doNotRemoveBodies = [f_GrpBLU11A];
   ```

   To make more than one group exempt, use commas to separate the group names:

   ```
   f_doNotRemoveBodies = [f_GrpBLU11A,GroupTwo,GroupThree];
   ```

A key limitation of this component is that it cannot automatically add the event handler to units which are created dynamically *during* the mission (for example, if you use a script to generate enemies or civilians dynamically). However, you can add the event handler by ensuring that any dynamically-created units have the following code in their **Init:** line:

```
this addEventHandler ["killed",
	{
		if (local BAS_Server_Logic) then
		{
			f_abrFIFO = f_abrFIFO + [_this select 0];
		} else
		{
			_this execVM "f\common\f_abrAddToFIFO.sqf"
		};
	}];
```

CONFIGURABLE PLAYABLE SLOTS (ACE VERSION)

By default, a mission created with BAS f contains 1 BLUFOR player group composed of standard ArmA units (see Configurable Playable Slots component). The Configurable Playable Slots (ACE Version) component replaces all standard units with ACE versions (US Army ACU), but in all other respects is identical to the core component on which it is based.

To use this optional component, **before** you begin to make your own mission:

1. Delete the file **`mission.sqm`** in the mission folder.

2. Change the name of the file **`mission_ACE.sqm`** to **`mission.sqm`**

3. Follow the steps in the section of this manual entitled "Naming Your Mission".

SECTION C

SHACKTACTICAL OPTIONAL COMPONENTS

ShackTactical is an international MP gaming community that focuses on ArmA (and, historically, OFP and the WGL mod). Find out more about ShackTactical at:

http://dslyecxi.com/

The founder of ShackTactical, Dslyecxi, is also the author of "Tactics, Techniques, and Procedures for Armed Assault", which is the benchmark guide for ArmA players and clans, and contains many concepts and ideas which are pertinent to good mission making. Read it at:

http://dslyecxi.com/armattp.html

The following mini-guides and optional components are intended to provide your mission with extra features which will help it conform to the ideas and standards used by ShackTactical:

- ShackTactical: Baseline Mission File Template
- ShackTactical: Group IDs
- ShackTactical: Markers
- ShackTactical: Markers (Addon Version)
- ShackTactical: Fireteam Markers
- ShackTactical: Fireteam Markers (Addon Version)
- ShackTactical: Briefing File Template (Coop Version)
- ShackTactical: Briefing File Template (Attack & Defend Version)
- ShackTactical: CoC CEX Support
- ShackTactical: kevb0's Wounding Script
- ShackTactical: kevb0's Outtro Script
- ShackTactical: kevb0's Assign Gear Script
- ShackTactical: ShackTac f

SHACKTACTICAL: BASELINE MISSION FILE TEMPLATE

ShackTactical missions use a regular platoon structure for organising player-controlled troops. As "Tactics, Techniques, and Procedures for Armed Assault" describes:

> "The ShackTac Platoon is based off of a standard USMC rifle platoon, with some minor differences. It works off of the principle that any person in the platoon should have to worry about three other people at most - the platoon commander deals with the three squad leaders, the squad leaders deal with their three fireteam leaders, and each fireteam leader deals with the three other players in his fireteam. Commanding three people is ideal, and since that occurs at all levels of leadership in the platoon, the structure is a very flexible and relatively easy one to work with. There is good reason why the US Marines use this method.
>
> Our Platoon ends up being 46 players strong, and can be fleshed out with AI any time that there are less than that many people on the server. Alternatively, the platoon can be cut down to a section (two squads) if there are not enough people to fill the whole thing and AI is not desired.
>
> The platoon consists of four main elements - the command element and three rifle squads, Alpha, Bravo, and Charlie. Each squad has three fireteams in it, labeled simply as 1st, 2nd, and 3rd fireteam, and each fireteam is comprised of three soldiers and one fireteam leader."

It is reasonably time-consuming for a mission designer to recreate this platoon structure – especially if the mission is attack & defend (with players on both sides). Additionally, playable slots in ShackTactical missions also have useful descriptions such as "1Plt Charlie Squad Leader" and "2Plt A3 Fireteam Leader", which can also be time-consuming to enter.

The ShackTactical: Baseline Mission File Template provides the mission designer with pre-placed BLUFOR, OPFOR and Resistance platoons (using ACE units and weapons) – all of which conform to the ShackTactical structure, naming and baseline equipment conventions. Please note that each pre-placed platoon is enlarged with the addition of Delta, Echo and Fox squads.

To use this optional component, **before** you begin to make your own mission:

1. Delete the file **`mission.sqm`** in the mission folder.

2. Decide what uniforms you would like BLUFOR troops to wear, as you have a choice between MARPAT and MARPAT D. For each uniform variant there is a separate mission.sqm file, as follows:

   - **`mission_ShackTac_ACE_MARPAT_D.sqm`**
   - **`mission_ShackTac_ACE_Woodland.sqm`**

   For the remainder of these instructions, let us assume you choose to use MARPAT D.

3. Change the name of the file **`mission_ShackTac_ACE_MARPAT_D.sqm`** to **`mission.sqm`**

4. Delete the file **`mission_ShacTac_ACE_MARPAT.sqm`**

5. Follow the steps in the section of this manual entitled "Naming Your Mission".

6. Delete the platoons and / or squads which you do not need for your mission.

Each platoon is arranged as follows:

Group	GroupName *	Units	Notes
Platoon HQ	GrpXXX_1Plt_PltHQ	1Plt PltHQ Platoon Commander	Has binoculars and radio
		1Plt PltHQ Rifleman	
		1Plt PltHQ Rifleman	
		1Plt PltHQ Medic	
Alpha Squad	GrpXXX_1Plt_Alpha	1Plt Alpha Squad Leader	Grenadier, has binoculars, smoke and radio
		1Plt Alpha Squad Medic	Has smoke
Alpha Fireteam 1	GrpXXX_1Plt_A1	1Plt A1 Fireteam Leader	Grenadier, has smoke and radio
		1Plt A1 Automatic Rifleman	
		1Plt A1 Assistant Automatic Rifleman	Has MG ammunition
		1Plt A1 Rifleman	
Alpha Fireteam 2	GrpXXX_1Plt_A2	1Plt A2 Fireteam Leader	Grenadier, has smoke and radio
		1Plt A2 Automatic Rifleman	
		1Plt A2 Assistant Automatic Rifleman	Has MG ammunition
		1Plt A2 Rifleman	
Alpha Fireteam 3	GrpXXX_1Plt_A3	1Plt A3 Fireteam Leader	Grenadier, has smoke and radio
		1Plt A3 Automatic Rifleman	
		1Plt A3 Assistant Automatic Rifleman	Has MG ammunition
		1Plt A3 Rifleman	
Bravo Squad	GrpXXX_1Plt_Bravo	1Plt Bravo Squad Leader	Grenadier, has binoculars, smoke and radio
		1Plt Bravo Squad Medic	Has smoke
Bravo Fireteam 1	GrpXXX_1Plt_B1	1Plt B1 Fireteam Leader	Grenadier, has smoke and radio
		1Plt B1 Automatic Rifleman	
		1Plt B1 Assistant Automatic Rifleman	Has MG ammunition
		1Plt B1 Rifleman	
Bravo Fireteam 2	GrpXXX_1Plt_B2	1Plt B2 Fireteam Leader	Grenadier, has smoke and radio
		1Plt B2 Automatic Rifleman	
		1Plt B2 Assistant Automatic Rifleman	Has MG ammunition
		1Plt B2 Rifleman	
Bravo Fireteam 3	GrpXXX_1Plt_B3	1Plt B3 Fireteam Leader	Grenadier, has smoke and radio
		1Plt B3 Automatic Rifleman	
		1Plt B3 Assistant Automatic Rifleman	Has MG ammunition
		1Plt B3 Rifleman	
Charlie Squad	GrpXXX_1Plt_Charlie	1Plt Charlie Squad Leader	Grenadier, has binoculars, smoke and radio
		1Plt Charlie Squad Medic	Has smoke
Charlie Fireteam 1	GrpXXX_1Plt_C1	1Plt C1 Fireteam Leader	Grenadier, has smoke and radio
		1Plt C1 Automatic Rifleman	
		1Plt C1 Assistant Automatic Rifleman	Has MG ammunition
		1Plt C1 Rifleman	

Charlie Fireteam 2	GrpXXX_1Plt_C2	1Plt C2 Fireteam Leader	Grenadier, has smoke and radio
		1Plt C2 Automatic Rifleman	
		1Plt C2 Assistant Automatic Rifleman	Has MG ammunition
		1Plt C2 Rifleman	
Charlie Fireteam 3	GrpXXX_1Plt_C3	1Plt C3 Fireteam Leader	Grenadier, has smoke and radio
		1Plt C3 Automatic Rifleman	
		1Plt C3 Assistant Automatic Rifleman	Has MG ammunition
		1Plt C3 Rifleman	
Delta Squad	GrpXXX_1Plt_Delta	1Plt Delta Squad Leader	Grenadier, has binoculars, smoke and radio
		1Plt Delta Squad Medic	Has smoke
Delta Fireteam 1	GrpXXX_1Plt_D1	1Plt D1 Fireteam Leader	Grenadier, has smoke and radio
		1Plt D1 Automatic Rifleman	
		1Plt D1 Assistant Automatic Rifleman	Has MG ammunition
		1Plt D1 Rifleman	
Delta Fireteam 2	GrpXXX_1Plt_D2	1Plt D2 Fireteam Leader	Grenadier, has smoke and radio
		1Plt D2 Automatic Rifleman	
		1Plt D2 Assistant Automatic Rifleman	Has MG ammunition
		1Plt D2 Rifleman	
Delta Fireteam 3	GrpXXX_1Plt_D3	1Plt D3 Fireteam Leader	Grenadier, has smoke and radio
		1Plt D3 Automatic Rifleman	
		1Plt D3 Assistant Automatic Rifleman	Has MG ammunition
		1Plt D3 Rifleman	
Echo Squad	GrpXXX_1Plt_Echo	1Plt Echo Squad Leader	Grenadier, has binoculars, smoke and radio
		1Plt Echo Squad Medic	Has smoke
Echo Fireteam 1	GrpXXX_1Plt_E1	1Plt E1 Fireteam Leader	Grenadier, has smoke and radio
		1Plt E1 Automatic Rifleman	
		1Plt E1 Assistant Automatic Rifleman	Has MG ammunition
		1Plt E1 Rifleman	
Echo Fireteam 2	GrpXXX_1Plt_E2	1Plt E2 Fireteam Leader	Grenadier, has smoke and radio
		1Plt E2 Automatic Rifleman	
		1Plt E2 Assistant Automatic Rifleman	Has MG ammunition
		1Plt E2 Rifleman	
Echo Fireteam 3	GrpXXX_1Plt_E3	1Plt E3 Fireteam Leader	Grenadier, has smoke and radio
		1Plt E3 Automatic Rifleman	
		1Plt E3 Assistant Automatic Rifleman	Has MG ammunition
		1Plt E3 Rifleman	
Fox Squad	GrpXXX_1Plt_Fox	1Plt Fox Squad Leader	Grenadier, has binoculars, smoke and radio
		1Plt Fox Squad Medic	Has smoke
Fox Fireteam 1	GrpXXX_1Plt_F1	1Plt F1 Fireteam Leader	Grenadier, has smoke and radio

		1Plt F1 Automatic Rifleman	
		1Plt F1 Assistant Automatic Rifleman	Has MG ammunition
		1Plt F1 Rifleman	
Fox Fireteam 2	GrpXXX_1Plt_F2	1Plt F2 Fireteam Leader	Grenadier, has smoke and radio
		1Plt F2 Automatic Rifleman	
		1Plt F2 Assistant Automatic Rifleman	Has MG ammunition
		1Plt F2 Rifleman	
Fox Fireteam 3	GrpXXX_1Plt_F3	1Plt F3 Fireteam Leader	Grenadier, has smoke and radio
		1Plt F3 Automatic Rifleman	
		1Plt F3 Assistant Automatic Rifleman	Has MG ammunition
		1Plt F3 Rifleman	

* For the GroupName, substitute BLU, OPF, or RES depending on side.

Note: All units are equipped with NVGs, regardless of rank or role.

SHACKTACTICAL: GROUP IDS

When players use side chat, the name of their squad / group appears at the start of each message. The default ArmA group names (or IDs) follow the format "1-1-A", "1-1-B" etc. This is not very helpful to players in a mission which uses the standard ShackTactical platoon structure, in which formations have names like "Alpha SL" or "Charlie FireTeam 3".

The ShackTactical: Group IDs component automatically applies meaningful names to all standard groups in a mission which uses the ShackTactical: Baseline Mission File Template. This means that when players use side chat, their messages will start with "1st Plt Alpha SL" or "1st Plt Charlie FT3" etc. This makes in-game text communications much clearer.

To create this functionality segments of code are placed in the following files:

- **`init.sqf`**
- **`f\common\ShackTac_setGroupIDs.sqf`**

To use this optional component:

1. Open the file **`init.sqf`** and look for the code segment entitled:

   ```
   // BAS f - ShackTactical - Group IDs
   ```

2. Edit the following line, removing the `//` at the start:

   ```
   // [] execVM "f\common\ShackTac_setGroupIDs.sqf";
   ```

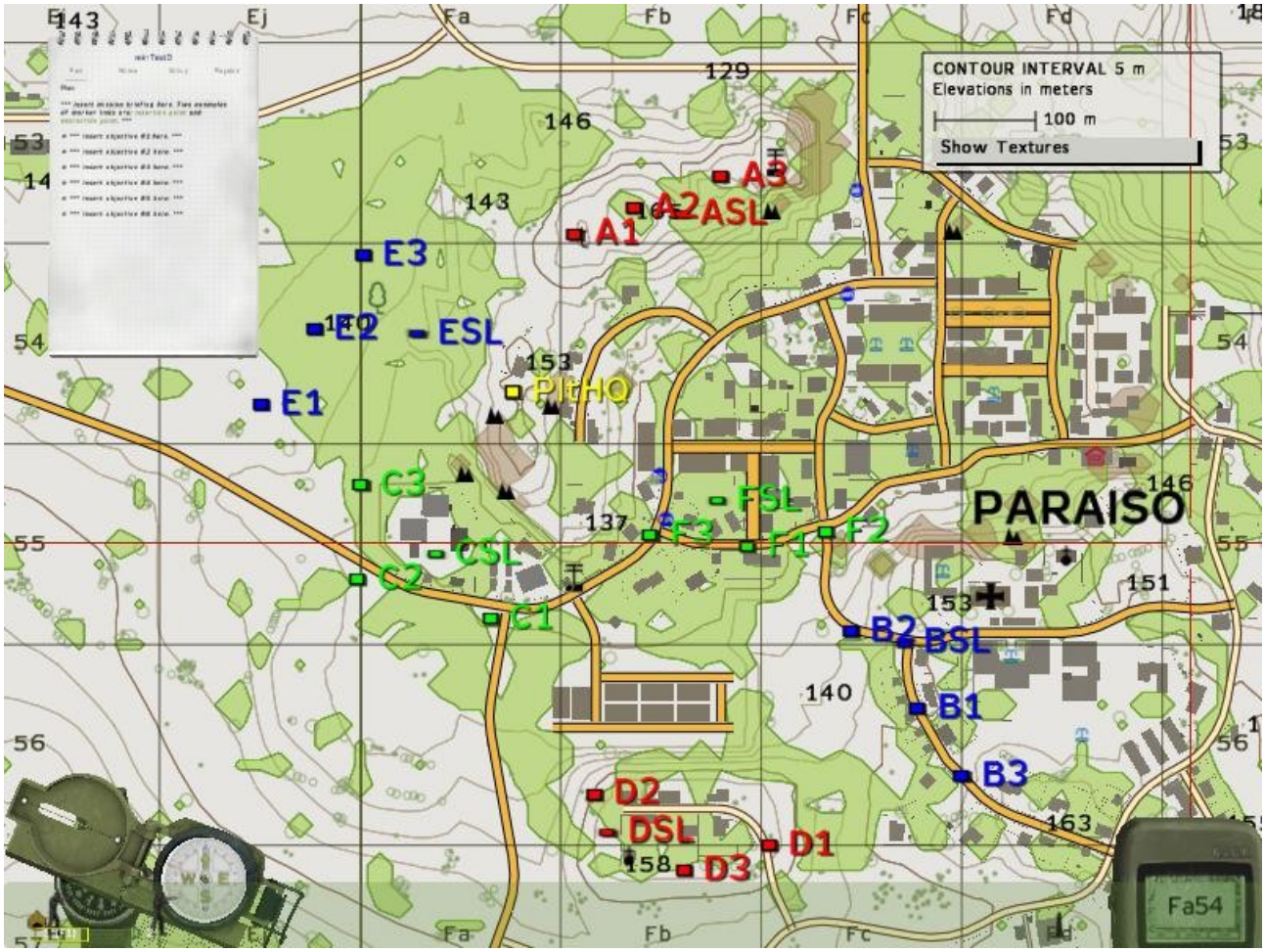
What players see in side chat

When players use side chat, their comments will be prefixed by the following strings

Group	In side chat
Platoon HQ	1st Plt CO
Alpha Squad	1st Plt Alpha SL
Alpha Fireteam 1	1st Plt Alpha FT1
Alpha Fireteam 2	1st Plt Alpha FT2
Alpha Fireteam 3	1st Plt Alpha FT3
Bravo Squad	1st Plt Bravo SL
Bravo Fireteam 1	1st Plt Bravo FT1
Bravo Fireteam 2	1st Plt Bravo FT2
Bravo Fireteam 3	1st Plt Bravo FT3
Charlie Squad	1st Plt Charlie SL
Charlie Fireteam 1	1st Plt Charlie FT1
Charlie Fireteam 2	1st Plt Charlie FT2
Charlie Fireteam 3	1st Plt Charlie FT3
Delta Squad	1st Plt Delta SL
Delta Fireteam 1	1st Plt Delta FT1
Delta Fireteam 2	1st Plt Delta FT2
Delta Fireteam 3	1st Plt Delta FT3
Echo Squad	1st Plt Echo SL
Echo Fireteam 1	1st Plt Echo FT1
Echo Fireteam 2	1st Plt Echo FT2

Echo Fireteam 3	1st Plt Echo FT3
Fox Squad	1st Plt Fox SL
Fox Fireteam 1	1st Plt Fox FT1
Fox Fireteam 2	1st Plt Fox FT2
Fox Fireteam 3	1st Plt Fox FT3

SHACKTACTICAL: MARKERS



If you use the ShackTactical: Baseline Mission File Template you may also want to use the same marker system employed by ShackTactical. The ShackTactical: Markers component automatically creates markers which show a player the position of the leaders of all elements in his/her platoon.

To create this functionality segments of code are placed in the following files:

- **`init.sqf`**
- **`f\common\ShackTac_setLocalMarkers.sqf`**
- **`f\common\ShackTac_localMarker.sqf`**

To use this optional component:

1. Open the file **`init.sqf`** and look for the code segment entitled:

   ```
   // BAS f - ShackTactical - Markers
   ```

2. Edit the following lines, removing the `//` at the start:

   ```
   // [] execVM "f\common\ShackTac_setLocalMarkers.sqf"
   // ShackTac_requireRadio = 0;
   ```

3. If you want markers to only update if the squad / fireteam leader is carrying a radio, edit the following line, changing the value of **`ShackTac_requireRadio`** to `1`:

   ```
   ShackTac_requireRadio = 0;
   ```

Note: If you do not use *all* the pre-placed groups in the ShackTactical: Baseline Mission File Template, markers for unused groups will not appear.

What BLUFOR, OPFOR and Resistance players see

Markers are used to denote the position of the leaders of all elements in the player's platoon:

Group	Marker Shape	Marker Colour	Marker Text
Platoon HQ	Dot (0.7 x 0.6)	Yellow	PLtHQ
Alpha Squad	Dot (0.7 x 0.3)	Red	ASL
Alpha Fireteam 1	Dot (0.7 x 0.5)	Red	A1
Alpha Fireteam 2	Dot (0.7 x 0.5)	Red	A2
Alpha Fireteam 3	Dot (0.7 x 0.5)	Red	A3
Bravo Squad	Dot (0.7 x 0.3)	Blue	BSL
Bravo Fireteam 1	Dot (0.7 x 0.5)	Blue	B1
Bravo Fireteam 2	Dot (0.7 x 0.5)	Blue	B2
Bravo Fireteam 3	Dot (0.7 x 0.5)	Blue	B3
Charlie Squad	Dot (0.7 x 0.3)	Green	CSL
Charlie Fireteam 1	Dot (0.7 x 0.5)	Green	C1
Charlie Fireteam 2	Dot (0.7 x 0.5)	Green	C2
Charlie Fireteam 3	Dot (0.7 x 0.5)	Green	C3
Delta Squad	Dot (0.7 x 0.3)	Red	DSL
Delta Fireteam 1	Dot (0.7 x 0.5)	Red	D1
Delta Fireteam 2	Dot (0.7 x 0.5)	Red	D2
Delta Fireteam 3	Dot (0.7 x 0.5)	Red	D3
Echo Squad	Dot (0.7 x 0.3)	Blue	ESL
Echo Fireteam 1	Dot (0.7 x 0.5)	Blue	E1
Echo Fireteam 2	Dot (0.7 x 0.5)	Blue	E2
Echo Fireteam 3	Dot (0.7 x 0.5)	Blue	E3
Fox Squad	Dot (0.7 x 0.3)	Green	FSL
Fox Fireteam 1	Dot (0.7 x 0.5)	Green	F1
Fox Fireteam 2	Dot (0.7 x 0.5)	Green	F2
Fox Fireteam 3	Dot (0.7 x 0.5)	Green	F3

What Civilian players see

By default, players in civilian slots (which are usually used for observers) will see markers denoting the position of the leaders of all elements in all active platoons (BLUFOR, OPFOR and Resistance). All markers are colour-coded to denote side:

- BLUFOR: Blue
- OPFOR: Red
- Resistance: Green

How the markers update if ShackTac_requireRadio is set to 0

The position of each marker is updated every 6 seconds, if the leader of the element is alive. If all units in the element die, the marker will remain visible, but its position will not be updated again.

How the markers update if ShackTac_requireRadio is set to 1

If the ShackTactical: Baseline Mission File Template has been used, by default, the leaders of each element in a platoon are equipped with a radio (in this build of BAS f the radio is represented using a 'Laserbatteries' magazine object).

The position of each marker is updated every 6 seconds, if the following conditions are met:

1. The leader of the element is alive.
2. The leader of the element is carrying a radio.

If a leader dies, the marker will remain visible, but its position will not begin to update again unless the *new* leader of the element picks up a radio. Note that the new leader does not have to pick up the radio found on the dead leader's body – *any* radio will work, since markers are linked to the group, not specific radio objects.

Note: This component is intended for use on servers which have disabled the default ArmA map markers for units and vehicles (e.g. the green triangles for each infantryman).

SHACKTACTICAL: MARKERS (ADDON VERSION)

The ShackTactical: Markers (Addon Version) component is identical to the ShackTactical: Markers component, but is designed to work with the ShackTactical Markers addon. This automatically creates markers which show a player the position of the leaders of all elements in his/her platoon.

To create this functionality segments of code are placed in the following files:

- **init.sqf**
- **f\common\@ShackTac_setLocalMarkers.sqf**
- **f\common\@ShackTac_localMarker.sqf**

To use this optional component:

1. Open the file **init.sqf** and look for the code segment entitled:

   ```
   // BAS f - ShackTactical - Markers (Addon Version)
   ```

2. Edit the following lines, removing the // at the start:

   ```
   // [] execVM "f\common\@ShackTac_setLocalMarkers.sqf"
   // ShackTac_requireRadio = 0;
   ```

3. If you want markers to only update if the squad / fireteam leader is carrying a radio, edit the following line, changing the value of **ShackTac_requireRadio** to **1**:

   ```
   ShackTac_requireRadio = 0;
   ```

Note: If you do not use *all* the pre-placed groups in the ShackTactical: Baseline Mission File Template, markers for unused groups will not appear.

What BLUFOR, OPFOR and Resistance players see

Markers are used to denote the position of the leaders of all elements in the player's platoon:

Group	Marker Shape	Marker Colour	Marker Text
Platoon HQ	Custom	Yellow	HQ Plt
Alpha Squad	Custom	Red	ASL
Alpha Fireteam 1	Custom	Red	A1
Alpha Fireteam 2	Custom	Red	A2
Alpha Fireteam 3	Custom	Red	A3
Bravo Squad	Custom	Blue	BSL
Bravo Fireteam 1	Custom	Blue	B1
Bravo Fireteam 2	Custom	Blue	B2
Bravo Fireteam 3	Custom	Blue	B3
Charlie Squad	Custom	Green	CSL
Charlie Fireteam 1	Custom	Green	C1
Charlie Fireteam 2	Custom	Green	C2
Charlie Fireteam 3	Custom	Green	C3
Delta Squad	Custom	Red	DSL
Delta Fireteam 1	Custom	Red	D1
Delta Fireteam 2	Custom	Red	D2
Delta Fireteam 3	Custom	Red	D3
Echo Squad	Custom	Blue	ESL
Echo Fireteam 1	Custom	Blue	E1
Echo Fireteam 2	Custom	Blue	E2
Echo Fireteam 3	Custom	Blue	E3
Fox Squad	Custom	Green	FSL
Fox Fireteam 1	Custom	Green	F1
Fox Fireteam 2	Custom	Green	F2
Fox Fireteam 3	Custom	Green	F3

What Civilian players see

By default, players in civilian slots (which are usually used for observers) will see markers denoting the position of the leaders of all elements in all active platoons (BLUFOR, OPFOR and Resistance). All markers are colour-coded to denote side:

- BLUFOR: Blue
- OPFOR: Red
- Resistance: Green

How the markers update if ShackTac_requireRadio is set to 0

The position of each marker is updated every 6 seconds, if the leader of the element is alive. If all units in the element die, the marker will remain visible, but its position will not be updated again.

How the markers update if ShackTac_requireRadio is set to 1

If the ShackTactical: Baseline Mission File Template has been used, by default, the leaders of each element in a platoon are equipped with a radio (in this build of BAS f the radio is represented using a 'Laserbatteries' magazine object).

The position of each marker is updated every 6 seconds, if the following conditions are met:

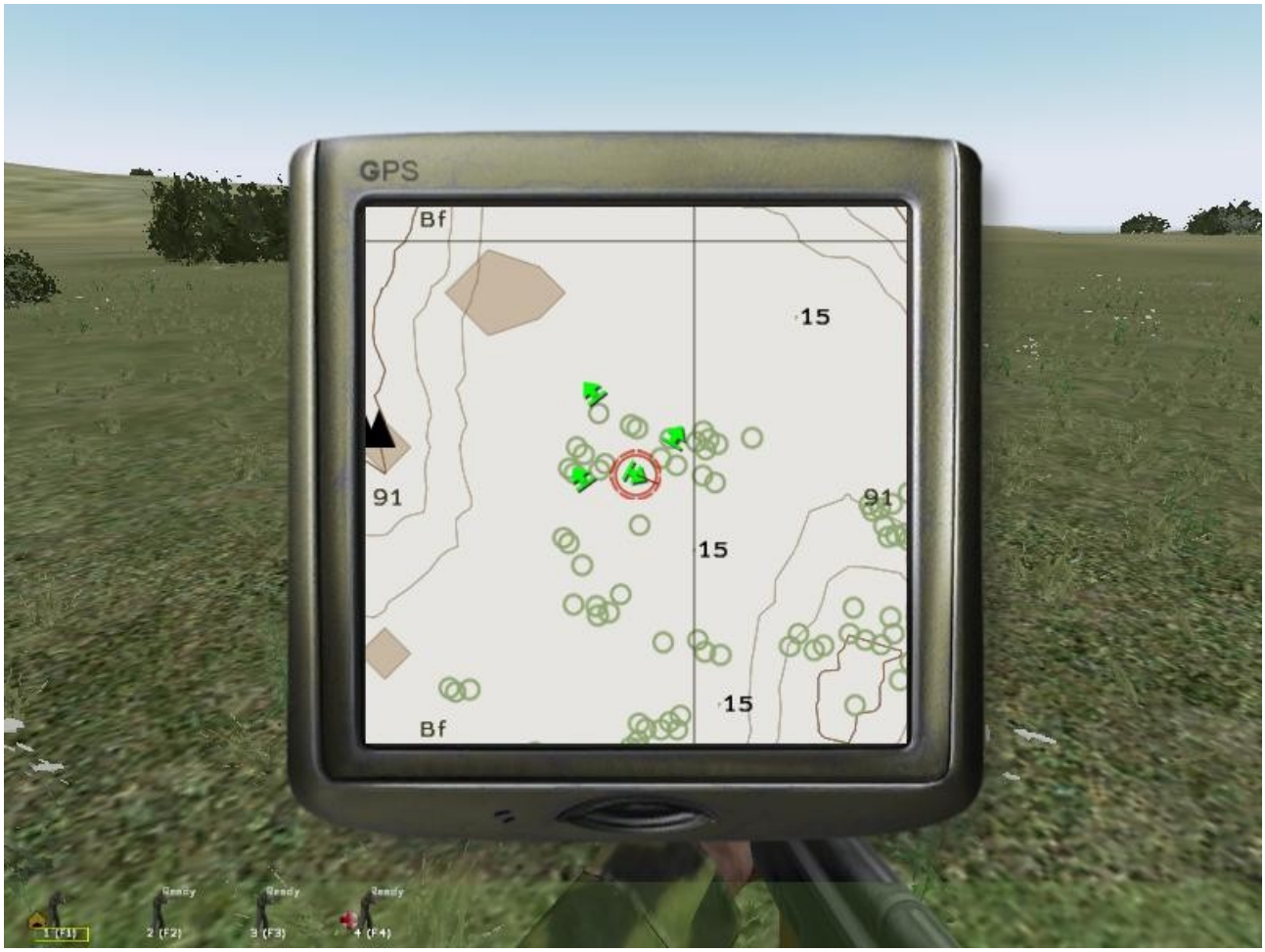
1. The leader of the element is alive.
2. The leader of the element is carrying a radio.

If a leader dies, the marker will remain visible, but its position will not begin to update again unless the *new* leader of the element picks up a radio. Note that the new leader does not have to pick up the radio found on the dead leader's body – *any* radio will work, since markers are linked to the group, not specific radio objects.

Note 1: This component is intended for use on servers which have disabled the default ArmA map markers for units and vehicles (e.g. the green triangles for each infantryman).

Note 2: This component should NOT be used if you are also using the ShackTactical: CoC CEX Support component (which has its own markers built in).

SHACKTACTICAL: FIRETEAM MARKERS



If you use the ShackTactical: Baseline Mission File Template and ShackTactical: Markers component, you may also want to use the ShackTactical: Fireteam Markers component. This automatically creates markers which show a player the position and orientation of other soldiers in his/her fireteam (group).

To create this functionality segments of code are placed in the following files:

- **`init.sqf`**
- **`f\common\ShackTac_setLocalFTMarkers.sqf`**
- **`f\common\ShackTac_localFTMarker.sqf`**

To use this optional component:

1. Open the file **`init.sqf`** and look for the code segment entitled:

   ```
   // BAS f - ShackTactical - Fireteam Markers
   ```

2. Edit the following line, removing the `//` at the start:

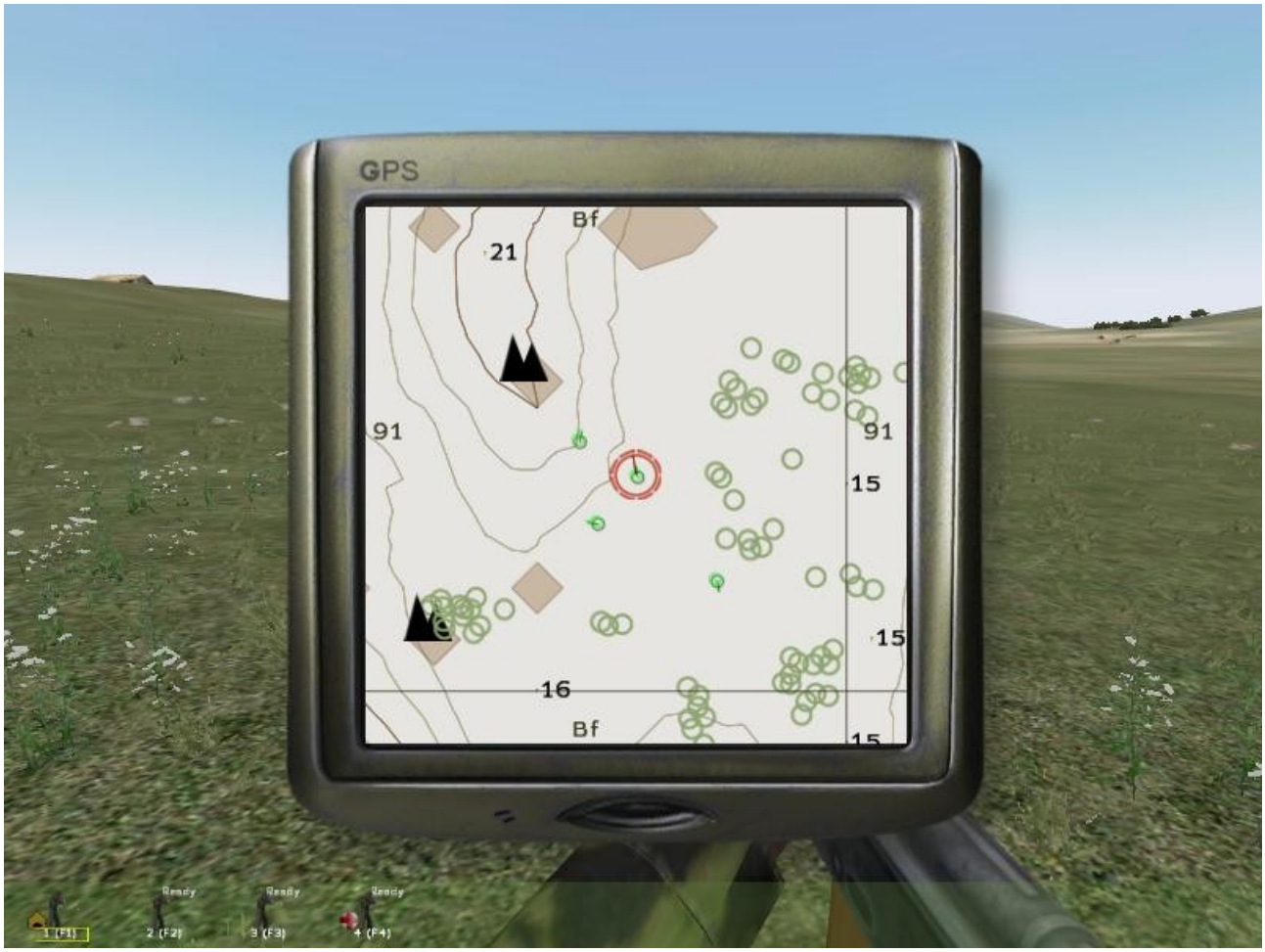
   ```
   // [] execVM "f\common\ShackTac_setLocalFTMarkers.sqf"
   ```

How the markers update

The position of each marker is updated every 3 seconds, if the unit is still alive. If a unit dies, its marker will be moved to co-ordinates [0,0] on the map, effectively making it invisible.

Note: This component is intended for use on servers which have disabled the default ArmA map markers for units and vehicles (e.g. the green triangles for each infantryman).

SHACKTACTICAL: FIRETEAM MARKERS (ADDON VERSION)



The ShackTactical: Fireteam Markers (Addon Version) component is identical to the ShackTactical: Fireteam Markers component, but is designed to work with the ShackTactical Markers addon. This automatically creates markers which show a player the position and orientation of other soldiers in his/her fireteam (group).

To create this functionality segments of code are placed in the following files:

- **`init.sqf`**
- **`f\common\@ShackTac_setLocalFTMarkers.sqf`**
- **`f\common\@ShackTac_localFTMarker.sqf`**

To use this optional component:

1. Open the file **`init.sqf`** and look for the code segment entitled:

   ```
   // BAS f - ShackTactical - Fireteam Markers (Addon Version)
   ```

2. Edit the following line, removing the `//` at the start:

   ```
   // [] execVM "f\common\@ShackTac_setLocalFTMarkers.sqf"
   ```

How the markers update

The position of each marker is updated every 3 seconds, if the unit is still alive. If a unit dies, its marker will be moved to co-ordinates [0,0] on the map, effectively making it invisible.

Note: This component is intended for use on servers which have disabled the default ArmA map markers for units and vehicles (e.g. the green triangles for each infantryman).

SHACKTACTICAL: BRIEFING FILE TEMPLATE (COOP VERSION)

For ShackTactical coop missions, a more structured approach to the briefing is required, one which follows a convention based loosely on the 'five-paragraph order' format in use with military organisations such as the USMC and British Army. This is easy to accomplish, but requires the use of a slightly different version of the **briefing.html** template.

The BAS f framework comes with a template **briefing.html** file designed for ShackTactical coop missions. To use this template:

1. Delete the file **briefing.html** in the mission folder.

2. Change the name of the file **briefing_ShackTac.html** to **briefing.html**

To create your briefing, open up the file **briefing.html** and complete the following sections:

- Notes
- Plan
- Debriefings
- Situation
- Mission
- Execution
- Administration
- Mission Credits

Throughout the new version of the **briefing.html** file the sections which you should edit have been labelled:

***** Insert [specific information] here. *****

Replace the text starting and ending with ******* using your own content (delete the ******* as well).

In the last section, *Mission Credits*, the suggested format for mission version is *n-n-n (DD MMM CCYY)*. An example of a mission that has reached version 1.7 on the 30th of April, 2007, would be: 1-7-0 (30 APR 2007).

The format of the **briefing.html** file is similar to HTML, although it is not exactly the same. Only a few HTML tags will work, but here are the key ones:

**
** - Will give a carriage return (new line).

**

** - Will give a blank line between paragraphs.

Text - Will create a link that, when clicked, will automatically centre the map over the marker named **mkrName** (be sure to name the marker in the ArmA editor).

If you would like to make the briefing available in more than one language, follow these steps:

1. Complete the original version of the **briefing.html** file in English, and save it.

2. Make a copy of the file, and rename it:

   briefing.german.html

   This will create a version of the file that is opened automatically by German language versions of ArmA.

3. Open the file **briefing.german.html** and translate your inserted texts into German.

4. Repeat steps 2 and 3 for the languages: **Czech**, **Polish**, **French**, **Spanish**, **Italian**, **French** and **Russian**. Note that if a non-English version of ArmA cannot find a copy of the **briefing.YourLanguage.html** file in its language, it will use the file **briefing.html** (which should be in English).

Please note that if you are creating a version in Russian, you must ensure that the file is saved in Unicode format.

SHACKTACTICAL: BRIEFING FILE TEMPLATE (ATTACK & DEFEND VERSION)

For ShackTactical attack and defend missions, a more structured approach to the briefing is required, one which follows a convention based loosely on the 'five-paragraph order' format in use with military organisations such as the USMC and British Army. This is easy to accomplish, but requires the use of a slightly different version of the **`briefing.html`** template.

The BAS f framework comes with a template **`briefing.html`** file designed for briefing different sides in a ShackTactical mission. To use this template:

1. Delete the file **`briefing.html`** in the mission folder.

2. Change the name of the file **`briefing_bySide_ShackTac.html`** to **`briefing.html`**

To create your briefing, open up the file **`briefing.html`** and complete the following sections:

- Notes - BLUFOR (if you have playable BLUFOR slots)
- Notes - OPFOR (if you have playable OPFOR slots)
- Notes - Resistance (if you have playable Resistance slots)
- Notes – Civilian (if you have playable Civilian slots)
- Plan - BLUFOR (if you have playable BLUFOR slots)
- Plan - OPFOR (if you have playable OPFOR slots)
- Plan - Resistance (if you have playable Resistance slots)
- Plan – Civilian (if you have playable Civilian slots)
- Situation (BLUFOR) (if you have playable BLUFOR slots)
- Mission (BLUFOR) (if you have playable BLUFOR slots)
- Execution (BLUFOR) (if you have playable BLUFOR slots)
- Administration (BLUFOR) (if you have playable BLUFOR slots)
- Situation (OPFOR) (if you have playable OPFOR slots)
- Mission (OPFOR) (if you have playable OPFOR slots)
- Execution (OPFOR) (if you have playable OPFOR slots)
- Administration (OPFOR) (if you have playable OPFOR slots)
- Situation (Resistance) (if you have playable Resistance slots)
- Mission (Resistance) (if you have playable Resistance slots)
- Execution (Resistance) (if you have playable Resistance slots)
- Administration (Resistance) (if you have playable Resistance slots)
- Situation (Civilian) (if you have playable Civilian slots)
- Mission (Civilian) (if you have playable Civilian slots)
- Execution (Civilian) (if you have playable Civilian slots)
- Administration (Civilian) (if you have playable Civilian slots)
- Debriefings
- Mission Credits

Throughout the new version of the **`briefing.html`** file the sections which you should edit have been labelled:

```
*** Insert [specific information] here. ***
```

Replace the text starting and ending with **`***`** using your own content (delete the **`***`** as well).

In the last section, *Mission Credits*, the suggested format for mission version is *n-n-n (DD MMM CCYY)*. An example of a mission that has reached version 1.7 on the 30th of April, 2007, would be: 1-7-0 (30 APR 2007).

The format of the **`briefing.html`** file is similar to HTML, although it is not exactly the same. Only a few HTML tags will work, but here are the key ones:

**
** - Will give a carriage return (new line).

**

** - Will give a blank line between paragraphs.

Text - Will create a link that, when clicked, will automatically centre the map over the marker named **mkrName** (be sure to name the marker in the ArmA editor).

If you would like to make the briefing available in more than one language, follow these steps:

1. Complete the original version of the **briefing.html** file in English, and save it.

2. Make a copy of the file, and rename it:

   briefing.german.html

   This will create a version of the file that is opened automatically by German language versions of ArmA.

3. Open the file **briefing.german.html** and translate your inserted texts into German.

4. Repeat steps 2 and 3 for the languages: **Czech**, **Polish**, **French**, **Spanish**, **Italian**, **French** and **Russian**. Note that if a non-English version of ArmA cannot find a copy of the **briefing.YourLanguage.html** file in its language, it will use the file **briefing.html** (which should be in English).

Please note that if you are creating a version in Russian, you must ensure that the file is saved in Unicode format.

SHACKTACTICAL: COC CEX SUPPORT

CEX is a sophisticated addon created by Chain of Command (CoC), which allows players to command complex formations such as platoons, companies and battalions. The troops within these formations can be both AI and human-playable. It is even possible for humans to play junior commanding roles – taking command of squads or platoons, whilst another player has overall command of the complete formation (such as the company or battalion).

For missions created using the ShackTactical: Baseline Mission File Template, you may want to enable CEX so that humans can command entire platoons, even if not enough human players are available to fill every slot. The BAS f framework contains template configuration files for quickly enabling CEX functionality for the standard ShackTactical BLUFOR, OPFOR and Resistance (RACS) platoons.

To create this functionality segments of code are placed in the following files:

- **description.ext**
- **f\common\@ShackTac_CEX_BLU_Platoon.hpp**
- **f\common\@ShackTac_CEX_OPF_Platoon.hpp**
- **f\common\@ShackTac_CEX_RES_Platoon.hpp**

To use this optional component:

1. In the ArmA MP Mission editor, make sure you place a game logic of the type **CEX SERVER** (which is available from **Units >> Game Logic >> CoC**).

2. Open the file **description.ext** and look for the code segment entitled:

   ```
   // BAS f - ShackTactical - CoC CEX Support
   ```

3. Edit the following lines, removing the // at the start of each:

   ```
   // class CEX
   // {
   ```

4. To enable CEX for the standard ShackTactical BLUFOR platoon, edit the following line and remove the // at the start:

   ```
   // #include "f\common\@ShackTac_CEX_BLU_Platoon.hpp"
   ```

   To enable CEX for the standard ShackTactical OPFOR platoon, edit the following line and remove the // at the start:

   ```
   // #include "f\common\@ShackTac_CEX_OPF_Platoon.hpp"
   ```

   To enable CEX for the standard ShackTactical Resistance (RACS) platoon, edit the following line and remove the // at the start:

   ```
   // #include "f\common\@ShackTac_CEX_RES_Platoon.hpp"
   ```

   Note: You can enable CEX for all platoons at the same time if you choose (if all three platoons are to be used in the same mission).

5. Edit the following line, removing the // at the start:

   ```
   // };
   ```

Each of the **.hpp** files contains the configuration settings and definitions required by CEX, and has been written to work with the relevant standard ShackTactical platoon as supplied in the ShackTactical: Baseline Mission Template component. If you decide to make changes to the platoon's structure or composition you may need to open the relevant **.hpp** file and make the appropriate changes (please see the CEX manual for more detail).

Free Look Camera Feature

By default the **Free Look** camera feature of CEX is enabled for each side (this is very useful for players commanding a platoon that is 100% AI, or mostly AI). If you wish to change this setting for a particular platoon (particularly one which is going to be composed of only human players), open the relevant **.hpp** file and find the following line:

```
CamEnabled      = 1;
```

To disable the **Free Look** camera feature, change this line to:

```
CamEnabled      = 0;
```

SHACKTACTICAL: KEVB0'S WOUNDING SCRIPT

When a player is wounded, standard ArmA functionality does not simulate effects such as being stunned or bleeding. kevb0's Wounding Script adds these effects and more:

- When a player is shot, if s/he is not killed instantly s/he will be stunned (during this time the player cannot move and will suffer effects such as black-outs)
- When stunned there is a chance that the player will drop his/her weapon: 75% if standing, 50% if crouched, or 25% if prone
- The amount of damage sustained determines how long the player remains stunned
- A medic can revive a stunned player, although the effects (black-outs) will continue for some time
- Once the player recovers from being stunned, s/he will start to bleed
- Depending on the amount of damage sustained, the bleeding may continue and the effects become worse over time (eventually the player may die if not treated)
- Bandages (represented by Laser Batteries) can be used to stop the bleeding

To create this functionality segments of code are placed in the following files:

- **description.ext**
- **init.sqf**
- **f\common\f_woundingScript\bandage.sqf**
- **f\common\f_woundingScript\bleeding.sqf**
- **f\common\f_woundingScript\f_woundingScriptSounds.hpp**
- **f\common\f_woundingScript\mando_getpos.sqf**
- **f\common\f_woundingScript\wakeup.sqf**
- **f\common\f_woundingScript\wounded.sqf**
- **f\common\f_woundingScript\sounds*.ogg**

To use this optional component:

1. Open the file **init.sqf** and look for the code segment entitled:

   ```
   // BAS f - kevb0's Wounding Script
   ```

2. Edit the following lines, removing the // at the start:

   ```
   // mando_getpos = compile (preprocessFileLineNumbers...
   // [player] execVM "f\common\f_woundingScript\wounded.sqf";
   ```

3. Open the file **description.ext** and look for the code segment entitled:

   ```
   // BAS f - kevb0's Wounding Script
   ```

4. Edit the following line, removing the // at the start:

   ```
   // #include "f\common\f_woundingScript\f_woundingScriptSounds.hpp"
   ```

Note: Do not use this component in conjunction with ACE (which has its own system for wounding and medical care).

SHACKTACTICAL: KEVB0'S OUTTRO SCRIPT

For adversarial missions, determining which side has won may not be as simple as detecting which is the first to be destroyed. For example: if an attacking force takes an outpost from a force one-third its size, but loses over half of its troops in the process, the mission designer may regard that as a defeat (or at least only a very minor victory).

kevb0's Outtro Script provides an easy way to track each side's performance using a points system for casualties and major objectives. The points earned by each side will determine the eventual winner, and the degree of victory.

To create this functionality segments of code are placed in the following files:

- **f\common\f_outtro.sqf**

To use this optional component:

1. Create a trigger which will fire at the point when you want the mission to end; for example, create a **Seized By** trigger over a town.

2. In the **On Activation** field you will need to insert a line of code that looks like this (please note there is no line-break – all the code should be on one line):

```
anEnding =
["Side1",pointsperkillside1,Side1Objectivepoints,"Side2",pointsperkillside
2,Side2Objectivepoints,"Whymissionisover",objecttopointcameraat] execVM
"f\common\f_outtro.sqf";
```

3. The precise composition of the line of code will be up to the mission designer. The following table provides a guide to each variable:

Variable	Type	Possible Values
"Side1"	String	Can be "**EAST**", "**WEST**", "**RESISTANCE**", or "**CIVILIAN**". It must be in capital letters and be enclosed by quotation marks.
pointsperkillside1	Number	The number of points lost by side #1 for every casualty it suffers. Must be a number.
Side1Objectivepoints	Number	The number of points won/lost by side #1 when the mission ends due to this trigger. Must be a number (it can be positive or negative).
"Side2"	String	Can be "**EAST**", "**WEST**", "**RESISTANCE**", or "**CIVILIAN**". It must be in capital letters and be enclosed by quotation marks.
pointsperkillside2	Number	The number of points lost by side #2 for every casualty it suffers. Must be a number.
Side2Objectivepoints	Number	The number of points won/lost by side #1 when the mission ends due to this trigger. Must be a number (it can be positive or negative).
"Whymissionisover"	String	The reason why the mission has ended (due to this trigger).
objecttopointcameraat	Object	The object which will be targeted by the camera used in the outtro screen at the end of the mission, which displays each side's scores and the overall mission result.

Please note that the mission ending called by the script is always #1 (so you may want to ensure that your text for debriefing #1 in the **briefing.html** file is appropriate).

Example

A BLUFOR SF Team must clear an town held by OPFOR insurgents. The script is activated using a "Seized By BLU" trigger placed over the town. The following line of code is placed in the **On**

Activation field of the trigger (please note there is no line-break):

```
anEnding = ["WEST",20,50,"EAST",5,-25,"The SF Team Cleared the
town",player] execVM "f\common\f_outtro.sqf";
```

The SF Team will lose 20 points for each person killed, and gain 50 for the objective (seizing the town) being completed. The Insurgents will lose 5 for each unit killed, and lose 25 for the town being lost.

SHACKTACTICAL: KEVB0'S ASSIGN GEAR SCRIPT

Changing the gear for different roles within a ShackTac platoon can become very time-consuming if using commands entered directly into the **INIT** field of each unit.
kevb0's Assign Gear Script provides an easy way to quickly and easily assign the correct gear for all the standard roles in a ShackTac platoon, from commander to rifleman.

To create this functionality segments of code are placed in the following files:

- **f\common\@ShackTac_assignGear.sqf**

To use this optional component:

1. Enter one of the following lines of code directly into the **INIT** field of each unit (this can be at the end of any other commands in the field):

Role	Code (no line breaks)
Platoon Commander	`nul = ["pltco",this] execVM "f\common\@ShackTac_assignGear.sqf";`
Platoon Medic	`nul = ["pltmedic",this] execVM "f\common\@ShackTac_assignGear.sqf";`
Squad Leader	`nul = ["sl",this] execVM "f\common\@ShackTac_assignGear.sqf";`
Squad Medic	`nul = ["squadmedic",this] execVM "f\common\@ShackTac_assignGear.sqf";`
Fireteam Leader	`nul = ["ftl",this] execVM "f\common\@ShackTac_assignGear.sqf";`
Automatic Rifleman	`nul = ["ar",this] execVM "f\common\@ShackTac_assignGear.sqf";`
Assistant Automatic Rifleman	`nul = ["aar",this] execVM "f\common\@ShackTac_assignGear.sqf";`
Antiarmor Specialist	`nul = ["at",this] execVM "f\common\@ShackTac_assignGear.sqf";`
Rifleman	`nul = ["rifleman",this] execVM "f\common\@ShackTac_assignGear.sqf";`

SHACKTACTICAL: SHACKTAC F

To speed up the process of creating a mission for use by ShackTactical, a special build of the BAS f mission template folder has been created for all supported islands; these folders use the following naming structure:

- **`ShackTac_f_v1-3-1.Island`**

This mission template folder differs from the standard BAS f template folder in the following ways:

- **ShackTactical: Mission File**
  The appropriate ShackTactical mission file (**`mission_ShackTac_ACE_MARPAT.sqm`**) has been renamed to **`mission.sqm`**. The original BAS f **`mission.sqm`** file has been deleted from the folder, although the remaining ShacTac mission file(s) remain so you can choose a different uniforms for BLUFOR troops. You will want to delete all unused **`mission.sqm`** files to reduce overall mission download size.

- **ShackTactical: Briefing File Templates**
  Only ShackTactical briefing file templates (**`briefing_ShackTac.html`** and **`briefing_bySide_ShackTac.html`**) are present in folder.

- **ShackTactical: Group IDs**
  This optional feature has been pre-enabled; all groups within the standard ShackTactical platoons will have the appropriate names in-game.

- **ShackTactical: Markers (addons version)**
  This optional feature has been pre-enabled; all groups within the standard ShackTactical platoons will have the appropriate markers in-game (note that fireteam leaders do NOT have to be carrying a 'radio' for the markers to update).

- **ShackTactical: Fireteam Markers (addons version)**
  This optional feature has been pre-enabled; all individuals within the standard ShackTactical groups will have the appropriate markers in-game (for any human players in the same group).

- **Kegetys' Spectator Script**
  This optional feature has been pre-enabled; when a player dies s/he will be able to view units from all sides in spectator mode (but remember: dead men don't talk!).

- **BAS f Gear Snippets**
  This core component has been removed.

SECTION D

LDD KYLLIKKI OPTIONAL COMPONENTS

LDD Kyllikki is an MP gaming community that focuses on ArmA (and, historically, the FDF Mod). Find out more about LDD Kyllikki at:

http://www.kyllikki.fi/

The following mini-guides and optional components are intended to provide your mission with extra features which will help it conform to the ideas and standards used by LDD Kyllikki:

- LDD Kyllikki: Baseline Mission File Template (FDF Version)

LDD KYLLIKKI : BASELINE MISSION FILE TEMPLATE (FDF VERSION)

LDD Kyllikki missions use a regular company structure for organising player-controlled FDF troops. It is reasonably time-consuming for a mission designer to recreate this company structure, especially if playable slots also have useful descriptions such as "JgrPlt 1 JJ (Officer)" and "MJgrPlt 1 Rk4 (Soldier)", which can also be time-consuming to enter.

The LDD Kyllikki: Baseline Mission File Template (FDF Version) provides the mission designer with a pre-placed RDF (Resistance) company (using FDF units and weapons) which conforms to the LDD Kyllikki structure, naming and baseline equipment conventions.

To use this optional component, **before** you begin to make your own mission:

1. Delete the file **`mission.sqm`** in the mission folder.
2. Change the name of the file **`mission_LDDK_FDF.sqm`** to **`mission.sqm`**
3. Follow the steps in the section of this manual entitled "Naming Your Mission".
4. Delete the platoons and / or squads which you do not need for your mission.

The company is arranged as follows:

HQ Platoon

Group	GroupName	Units	Notes
HQ Squad	GrpRES_HQPlt_HQ	HQPlt HQ Kpääl (Officer)	
		HQPlt HQ Lääkm (Medic)	
		HQPlt HQ Lääkm (Medic)	
		HQPlt HQ TkAmp (Sniper)	
		HQPlt HQ TkAmp (Sniper)	
		HQPlt HQ Lent (Pilot)	
		HQPlt HQ Lent (Pilot)	
		HQPlt HQ Lent (Pilot)	
Sissi Squad	GrpRES_HQPlt_Sissi	HQPlt Sissi RJ (Officer)	
		HQPlt Sissi RvaraJ (Soldier)	
		HQPlt Sissi Sissi1 (Soldier)	
		HQPlt Sissi Sissi2 (Soldier)	
		HQPlt Sissi Sissi2 (Soldier)	

Jaeger Platoon

Group	GroupName	Units	Notes
Jaeger Squad 1	GrpRES_JgrPlt_1	JgrPlt 1 JJ (Officer)	
		JgrPlt 1 RvaraJ (Asst. Squad Leader)	
		JgrPlt 1 Rk1 (Machine Gunner Asst.)	
		JgrPlt 1 Pst1 (AT Soldier)	
		JgrPlt 1 Kk (Machine Gunner)	
		JgrPlt 1 Rk2 (Soldier)	
		JgrPlt 1 Pst2 (Heavy AT Soldier)	
		JgrPlt 1 Rk3 (Grenadier)	

Jaeger Squad 2	GrpRES_JgrPlt_2	JgrPlt 2 RJ (Squad Leader)	
		JgrPlt 2 RvaraJ (Asst. Squad Leader)	
		JgrPlt 2 Rk1 (Machine Gunner Asst.)	
		JgrPlt 2 Pst1 (AT Soldier)	
		JgrPlt 2 Kk (Machine Gunner)	
		JgrPlt 2 Rk2 (Soldier)	
		JgrPlt 2 Pst2 (Heavy AT Soldier)	
		JgrPlt 2 Rk3 (Grenadier)	

Mechanised Jaeger Platoon

Group	GroupName	Units	Notes
Mechanised Jaeger Squad 1	GrpRES_MJgrPlt_1	MJgrPlt 1 JJ (Officer)	
		MJgrPlt 1 RvaraJ (Asst. Squad Leader)	
		MJgrPlt 1 Rk1 (Machine Gunner Asst.)	
		MJgrPlt 1 Pst1 (AT Soldier)	
		MJgrPlt 1 Kk (Machine Gunner)	
		MJgrPlt 1 Rk2 (Soldier)	
		MJgrPlt 1 Pst2 (Heavy AT Soldier)	
		MJgrPlt 1 Rk3 (Grenadier)	
		MJgrPlt 1 Rk4 (Soldier)	
		MJgrPlt 1 ItOhjm (AT Soldier)	Has Strela
BMP Crew	GrpRES_MJgrPlt_BMP	MJgrPlt BMP RJ (Squad Leader)	
		MJgrPlt BMP RvaraJ (Asst. Squad Leader)	

Note: This component requires FDF Mod.