ZDROB CTF & DM Templates Guide

by eRazer

V1.0

Summary

1 Introduction

This guide is bundled with some CTF and DM missions in pbo format and Kegetys cpbo command line utility that allows extracting the mission folder from a pbo file or packing the mission folder into a pbo file.

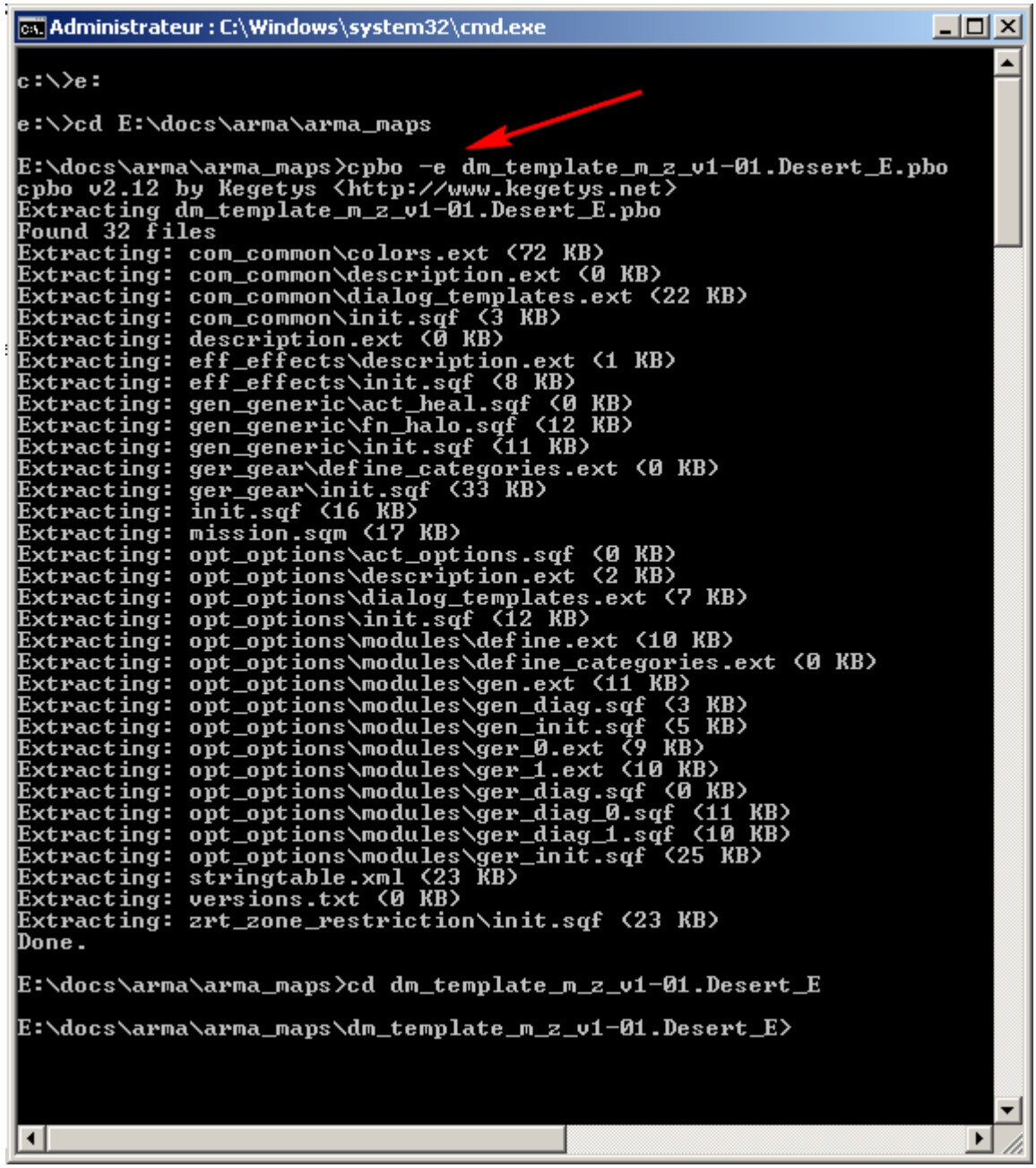
Note: these templates do not support AI units. All units should be playable units.

2 Quick Mission Making Tutorial

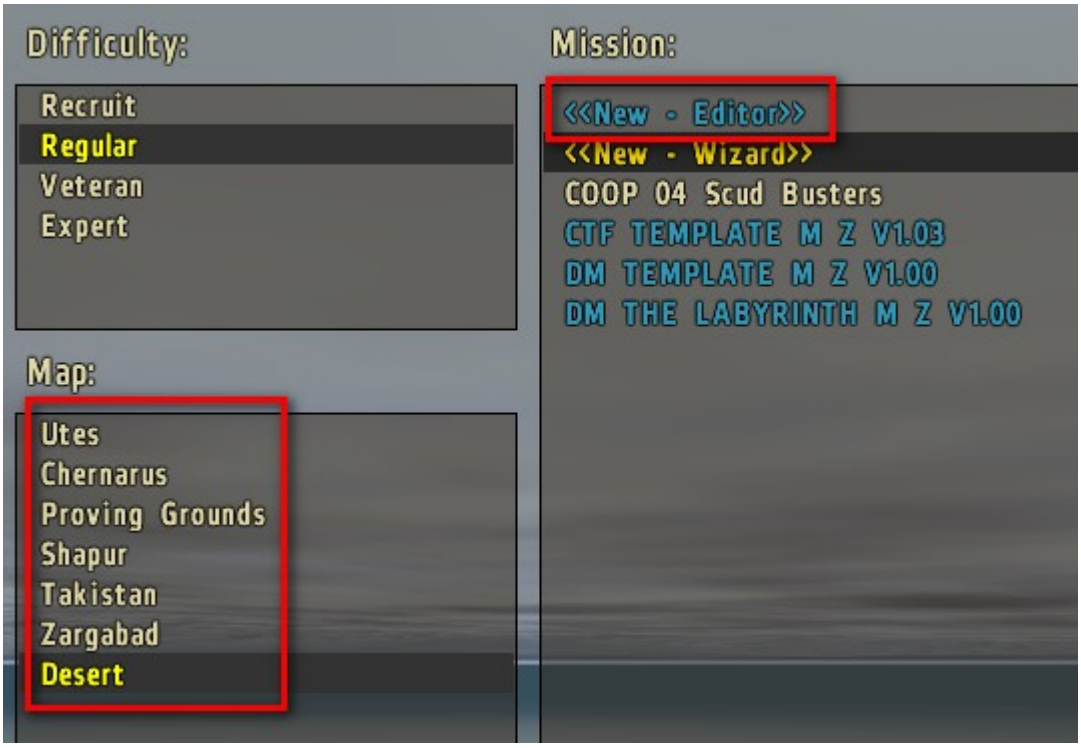
2.1 DM Mission

The following instructions guide you how to create your DM mission with the fewest possible steps.

1. Extract the DM template mission folder from the pbo file (ex: dm_template_m_z_v1-01.Desert_E.pbo)
   - Unpack the **ZDROB_CTF_&_DM_TEMPLATES.zip** archive which contains the DM template mission
   - Extract the mission folder from **dm_template_m_z_v1-01.Desert_E.pbo** file using the **cpbo** command line utility
   - To use the cpbo utility you must add it to the PATH system variable or specify the complete path to the cpbo.exe file (ex: `c:\_apps\cpbo -e mission.pbo`)

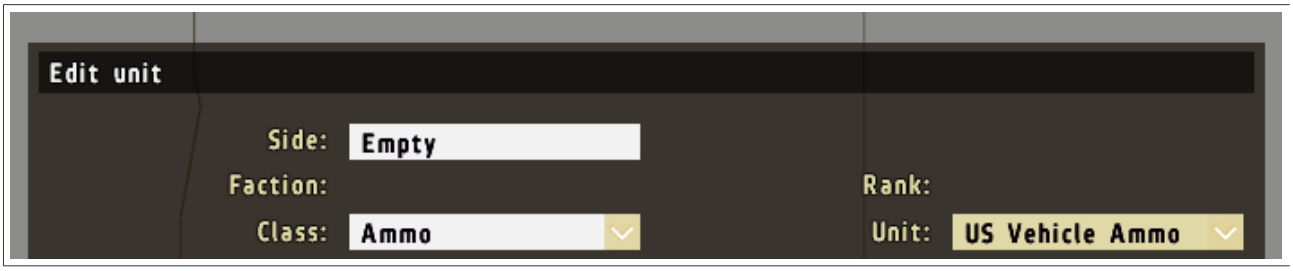


2. Create your mission using the ArmA Multiplayer Mission Editor
   - Open the ArmA Multiplayer Mission Editor

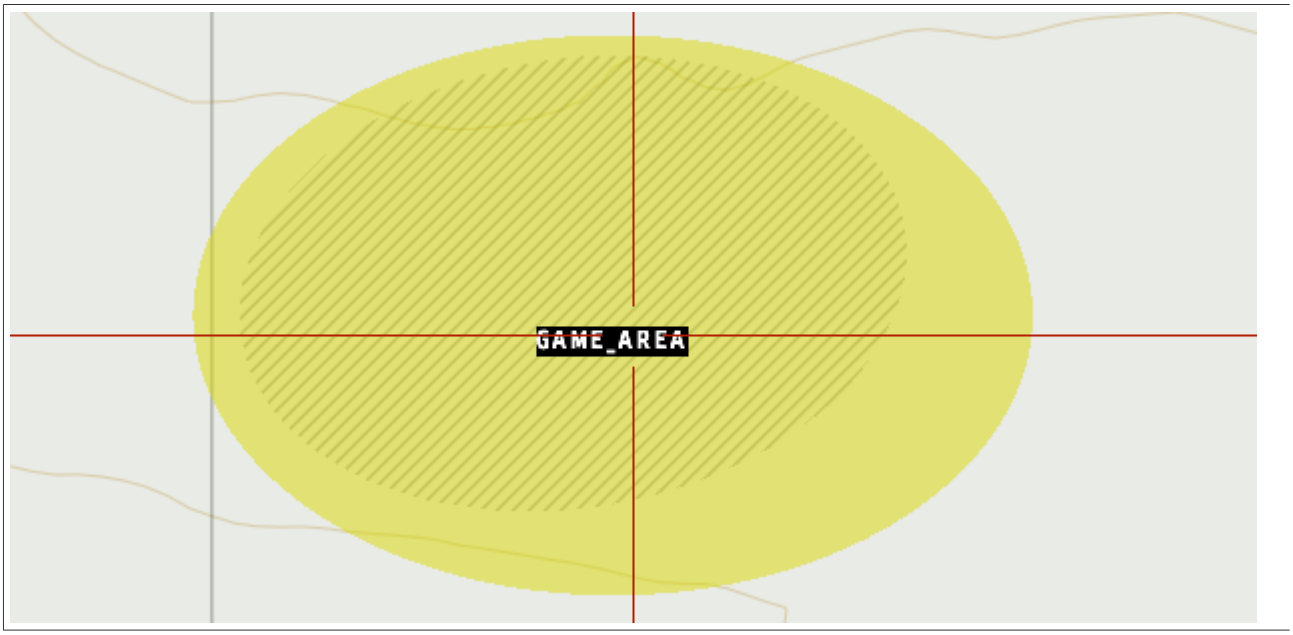


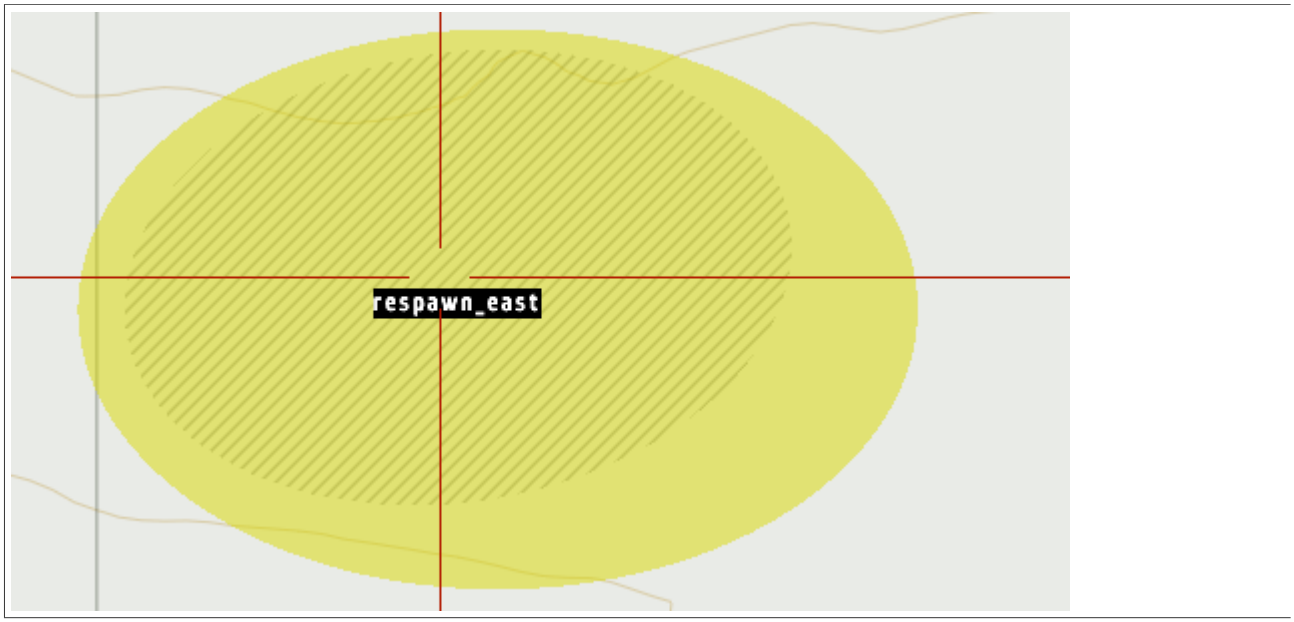
- Design your mission (by placing objects and units)
- Place ammoboxes of type **US Vehicle Ammo** (allows players to to rearm through the

Options menu when near these ammoboxes)



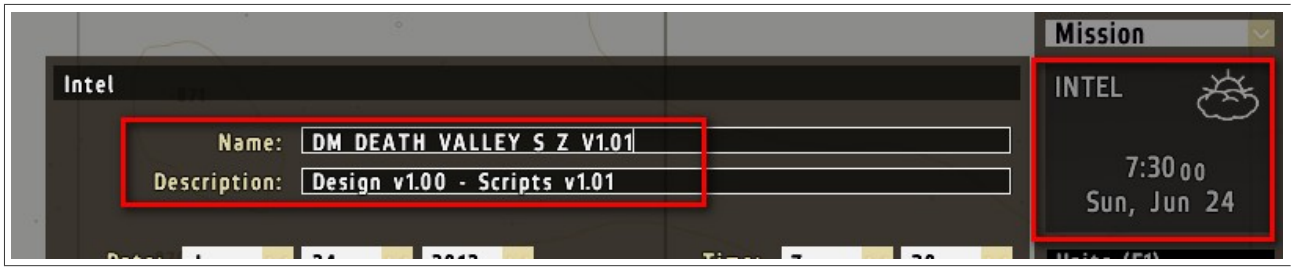
- Optional: Create markers named **GAME_AREAXXX** where **XXX** is anything (usually **_1**, **_2**, etc), these markers force players to play in a restricted area
- Create respawn markers named **respawn_westxxx** and **respawn_eastxxx** where **xxx** is anything and **west** and **east** are the sides of the units that will respawn inside the markers' areas
- All respawn markers must be placed inside one or more 'game area' markers otherwise players may die when respawning





- Save your mission (a folder will be created for the mission and placed inside C:\Users\user1\Documents\ArmA 2 Other Profiles\eRazer\MPMissions on Windows 7)
- When naming your mission folder (the **Mission name** field in the screenshot) follow this convention :
  - use lower-case letters
  - use 'underscore' character (_) instead of 'space' character
  - use hyphen character (-) instead of dot character (.)
  - start with **dm_** which stands for deathmatch
  - continue with the mission name: **death_valley_**
  - continue with the size of the map: **s_** for small, **m_** for medium, **l_** for large
  - continue with your personal signature or clan name acronym (I used **z_** for zdrob)
  - end with the mission version which in this case is the same as the **scripts version** of the template mission you will be copying scripts from (highlighted in the screenshot below)

- Edit the mission displayed name and description (these are displayed when selecting your mission on a server) by double-clicking on the **Intel** panel :
  - The mission displayed name is the same as the mission folder name but with some differences (use capital letters, 'space' character instead of 'underscore' character, 'dot' character instead of 'hyphen' character)
  - The mission description displays 2 versions:
    - The **design version** which indicates whether any modifications were made to the map to correct some problems (ex: better protected respawn areas, more obstacles to hide behind, different locations for ammoboxes, etc)
    - in this case, the design version is **1.00**
    - the **scripts version** which indicates whether any modifications were made to the scripts to correct some problems (ex: player respawns without weapon, etc)
    - in this case, the scripts version is the same as the scripts version of the DM template mission (1.01 in the example)
- Re-save the mission



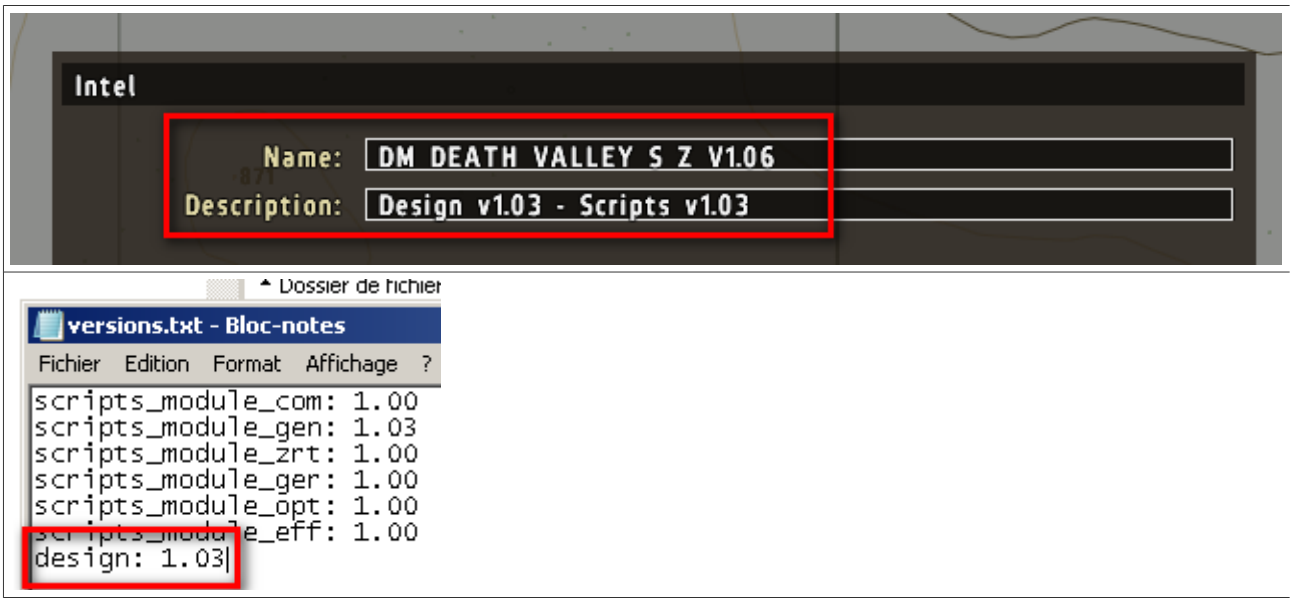
3. Copy the template scripts in your mission folder
   - Copy everything except **mission.sqm** file from **dm_template_m_z_v1-01.Desert_E** folder to your mission folder (ex: C:\Users\user1\Documents\ArmA 2 Other Profiles\eRazer\MPMissions\dm_death_valley_s_z_v1-01.chernarus)
4. Configure the scripts
   - Modify the number of roles (playable units) by opening **com_common\description.ext** file from your mission folder and modifying **maxPlayers** parameter to match the number of playable units in the mission

```
46
47  class Header
48  {
49      gameType = CTF;
50      minPlayers = 2;
51      maxPlayers = 80;
52  };
53  //////////////////////////////,
```

   - Modify the credits by opening **init.sqf** file from your mission folder and modify the following parameters by replacing the value with your name:
     - EFF_CREDIT_MAINDESIGNER = "eRazer [Z]";
     - EFF_CREDIT_DESIGNERS = ["eRazer [Z]"];
   - EFF_CREDIT_MAINDESIGNER is displayed when game starts and EFF_CREDIT_DESIGNERS when game ends

```
467 */
468 EFF_CREDIT_MAINDESIGNER = "eRazer [Z]";
469 EFF_CREDIT_DESIGNERS = ["eRazer [Z]"];
```

5. Pack the mission folder into a pbo file with the command `cpbo -p dm_death_valley_s_z_v1-01.chernarus`
6. Updating your mission
   - Extract the mission folder from the pbo file
   - Place it in C:\Users\user1\Documents\ArmA 2 Other Profiles\eRazer\MPMissions\ folder
   - Edit it with the ArmA Multiplayer Mission Editor
   - Do your design updates
   - Update the design version and the mission name
     - The mission version is computed from the design version and the scripts version using this generic formula: `(design_version * 100 + scripts_version * 100 - 2 * 100 + 100) / 100`
     - If the design version is 1.03 and the scripts version is 1.03 then the mission version is 1.06
     - Update the **versions.txt** file from your mission folder with the new design version

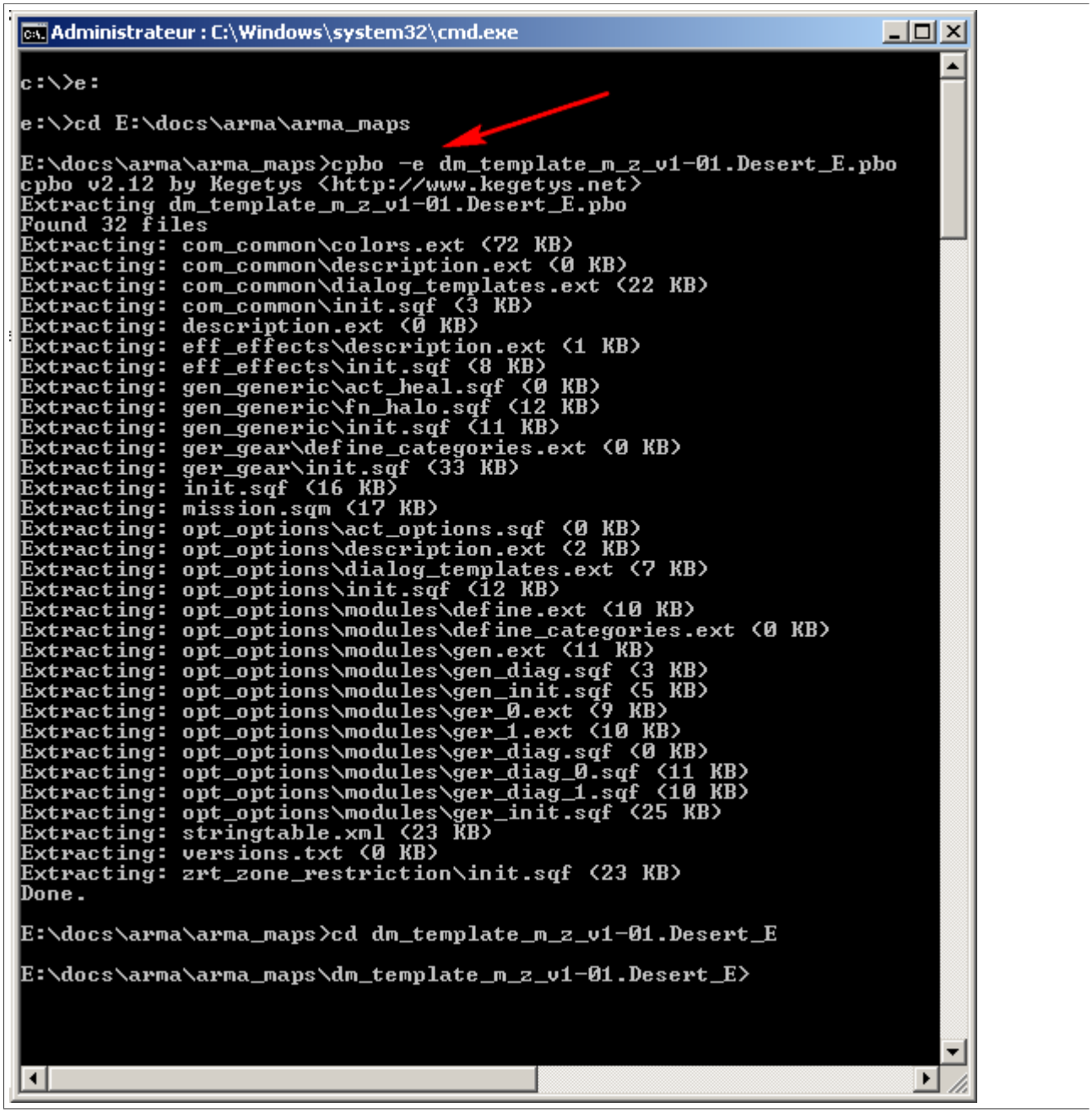


   - Update the number of roles in the **com_common\description.ext** file (as described in step 4) if you modified the number of playable units in the mission

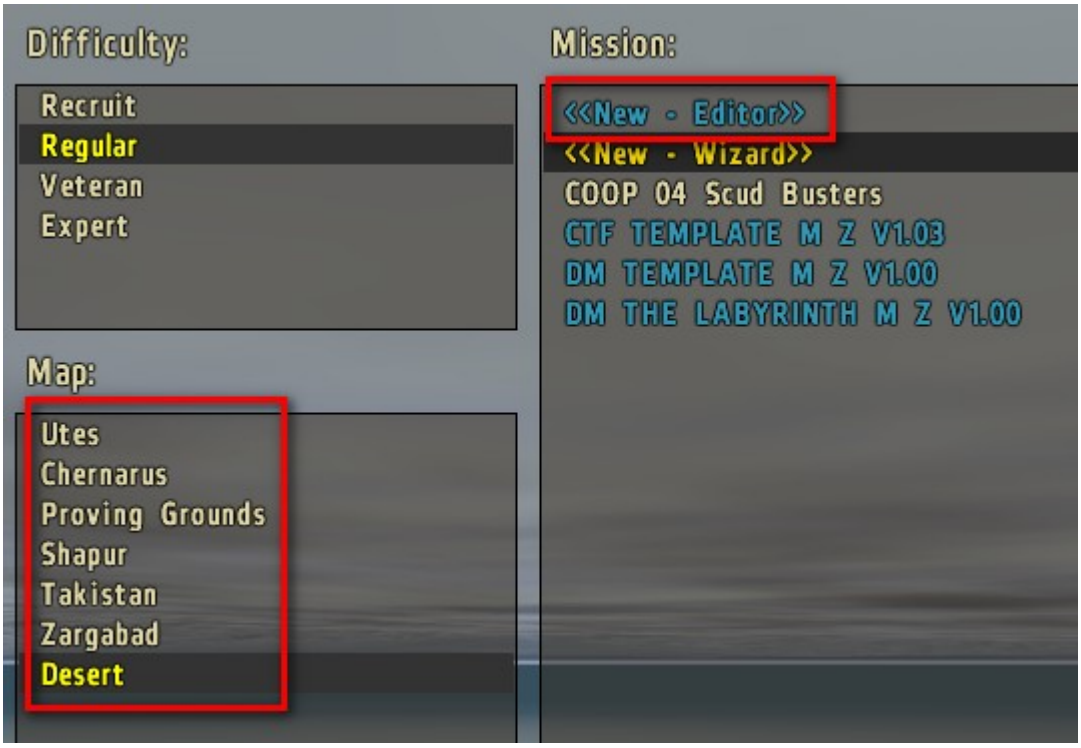
2.2 CTF Mission

The following instructions guide you how to create your CTF mission with the fewest possible steps.

1. Extract the CTF template mission folder from the pbo file (ex: ctf_template_m_z_v1-03.Desert_E.pbo)
   - Unpack the **ZDROB_CTF_&_DM_TEMPLATES.zip** archive which contains the CTF template mission
   - Extract the mission folder from **ctf_template_m_z_v1-03.Desert_E.pbo** file using the **cpbo** command line utility
   - To use the cpbo utility you must add it to the PATH system variable or specify the complete path to the cpbo.exe file (ex: `c:\_apps\cpbo -e mission.pbo`)



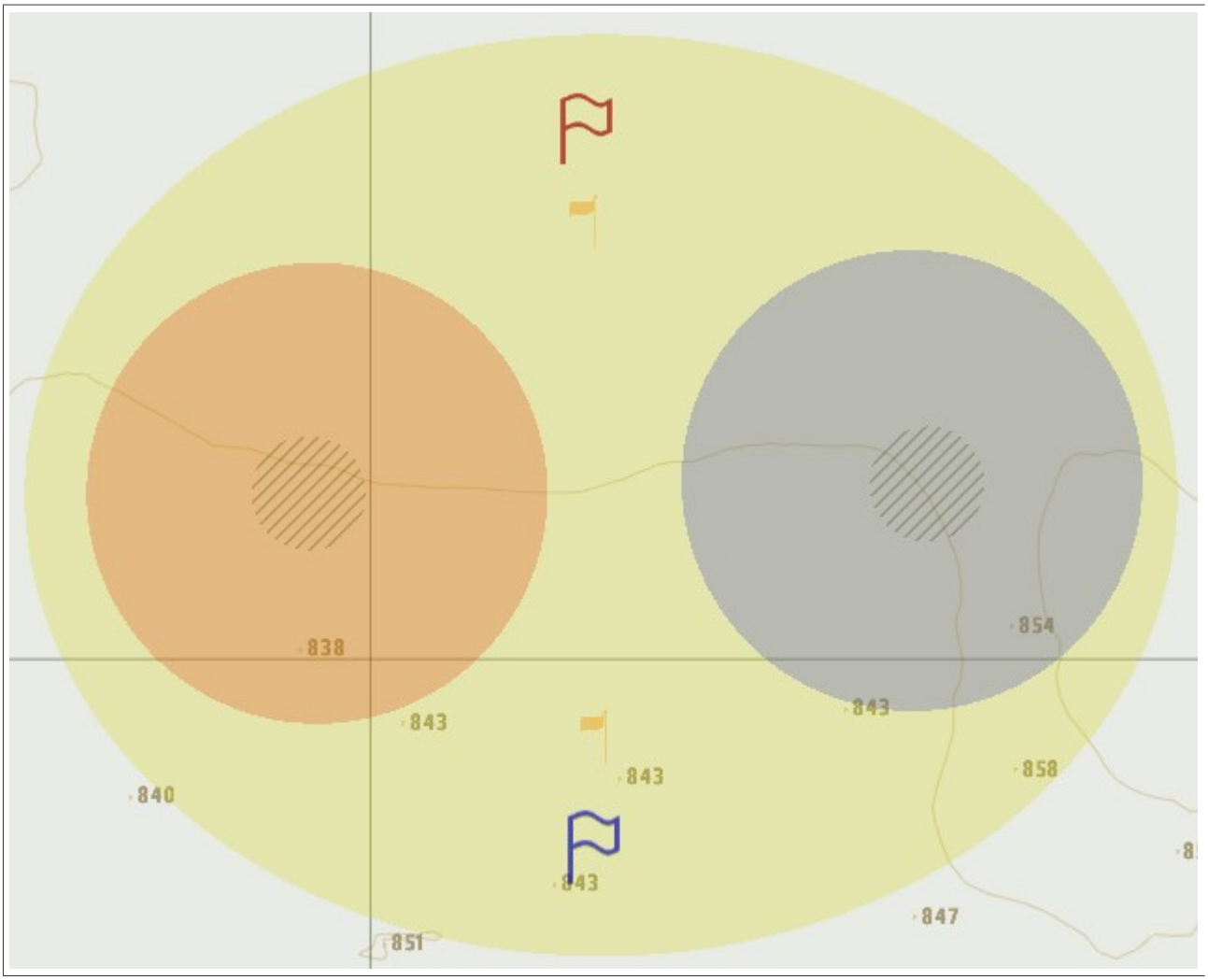
2. Create your mission using the ArmA Multiplayer Mission Editor
   - Open the ArmA Multiplayer Mission Editor



- Design your mission (by placing objects and units)
- Create markers named **FLAG_WEST** and **FLAG_EAST** near the west and east flag

poles

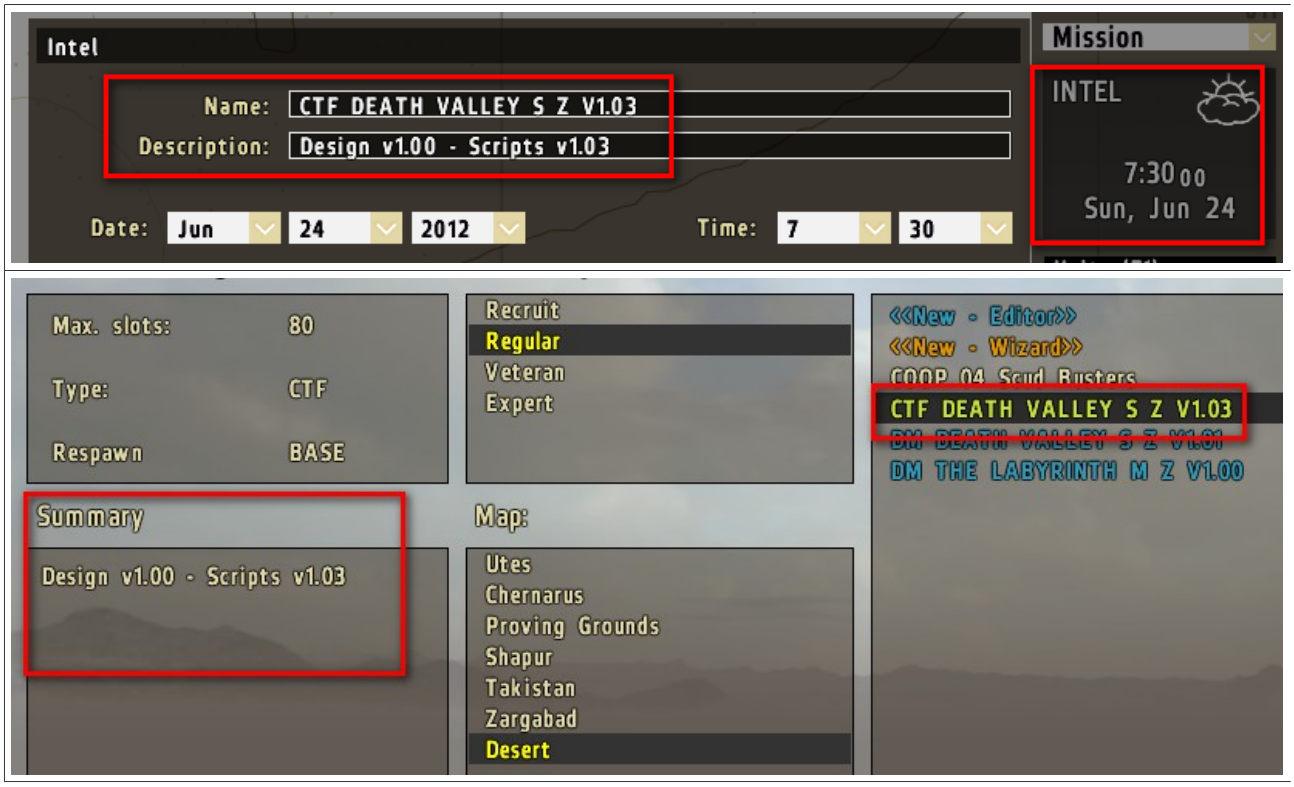
- Create markers named **BASE_WESTXXX** and **BASE_EASTXXX**, these markers prevent players from the enemy side from entering the area and enables access to the Options menu
- Optional: Create markers named **GAME_AREAXXX** where **XXX** is anything (usually **_1**, **_2**, etc), these markers force players to play in a restricted area
- Create respawn markers named **respawn_westxxx** and **respawn_eastxxx** where **xxx** is anything and **west** and **east** are the sides of the units that will respawn inside the markers' areas
- All respawn markers must be placed inside one or more 'game area' markers otherwise players may die when respawning



- Save your mission (a folder will be created for the mission and placed inside C:\Users\user1\Documents\ArmA 2 Other Profiles\eRazer\MPMissions on Windows 7)
- When naming your mission folder (the **Mission name** field in the screenshot) follow this convention :

- use lower-case letters
- use 'underscore' character (_) instead of 'space' character
- use hyphen character (-) instead of dot character (.)
- start with **ctf_** which stands for capture the flag
- continue with the mission name: **death_valley_**
- continue with the size of the map: **s_** for small, **m_** for medium, **l_** for large
- continue with your personal signature or clan name acronym (I used **z_** for zdrob)
- end with the mission version which in this case is the same as the **scripts version** of the template mission you will be copying scripts from (highlighted in the screenshot below)

- Edit the mission displayed name and description (these are displayed when selecting your mission on a server) by double-clicking on the **Intel** panel :
  - The mission displayed name is the same as the mission folder name but with some differences (use capital letters, 'space' character instead of 'underscore' character, 'dot' character instead of 'hyphen' character)
  - The mission description displays 2 versions:
    - The **design version** which indicates whether any modifications were made to the map to correct some problems (ex: better protected respawn areas, more obstacles to hide behind, different locations for ammoboxes, etc)
    - in this case, the design version is **1.00**
    - the **scripts version** which indicates whether any modifications were made to the scripts to correct some problems (ex: player respawns without weapon, etc)
    - in this case, the scripts version is the same as the scripts version of the CTF template mission (1.03 in the example)
- Re-save the mission



3. Copy the template scripts in your mission folder
   - Copy everything except **mission.sqm** file from **ctf_template_m_z_v1-03.Desert_E** folder to your mission folder (ex: C:\Users\user1\Documents\ArmA 2 Other Profiles\eRazer\MPMissions\ctf_death_valley_s_z_v1-03.Desert_E)
4. Configure the scripts
   - Modify the number of roles (playable units) by opening **com_common\description.ext**

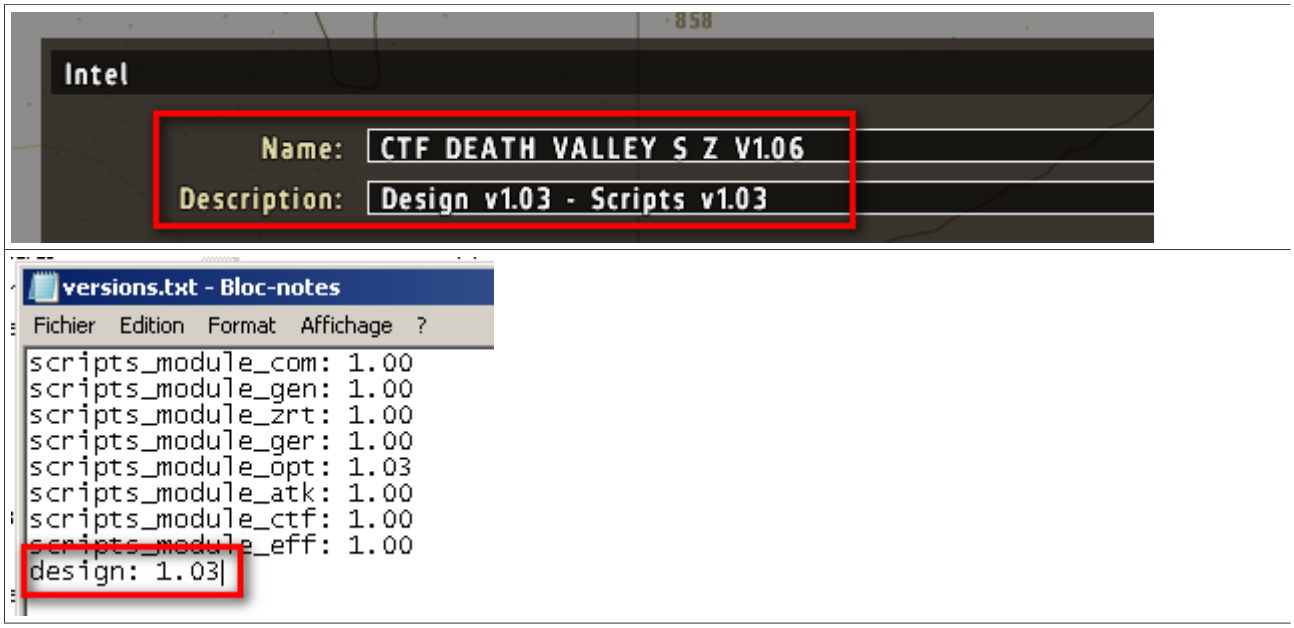
file from your mission folder and modifying **maxPlayers** parameter to match the number of playable units in the mission

```
46
47  class Header
48  {
49      gameType = CTF;
50      minPlayers = 2;
51      maxPlayers = 80;
52  };
53  ////////////////////////////////,
```

- Modify the credits by opening **init.sqf** file from your mission folder and modify the following parameters by replacing the value with your name:
  - EFF_CREDIT_MAINDESIGNER = "eRazer [Z]";
  - EFF_CREDIT_DESIGNERS = ["eRazer [Z]"];
- EFF_CREDIT_MAINDESIGNER is displayed when game starts and EFF_CREDIT_DESIGNERS when game ends

```
467 */
468 EFF_CREDIT_MAINDESIGNER = "eRazer [Z]";
469 EFF_CREDIT_DESIGNERS = ["eRazer [Z]"];
```

5. Pack the mission folder into a pbo file with the command `cpbo -p ctf_death_valley_s_z_v1-03.Desert_E`
6. Updating your mission
   - Extract the mission folder from the pbo file
   - Place it in C:\Users\user1\Documents\ArmA 2 Other Profiles\eRazer\MPMissions\ folder
   - Edit it with the ArmA Multiplayer Mission Editor
   - Do your design updates
   - Update the design version and the mission name
     - The mission version is computed from the design version and the scripts version using this generic formula: `(design_version * 100 + scripts_version * 100 - 2 * 100 + 100) / 100`
     - If the design version is 1.03 and the scripts version is 1.03 then the mission version is 1.06
     - Update the **versions.txt** file from your mission folder with the new design version



- Update the number of roles in the **com_common\description.ext** file (as described in step 4) if you modified the number of playable units in the mission

3 Advanced Tutorial

3.1 Gear Customization

This applies for both CTF and DM templates.

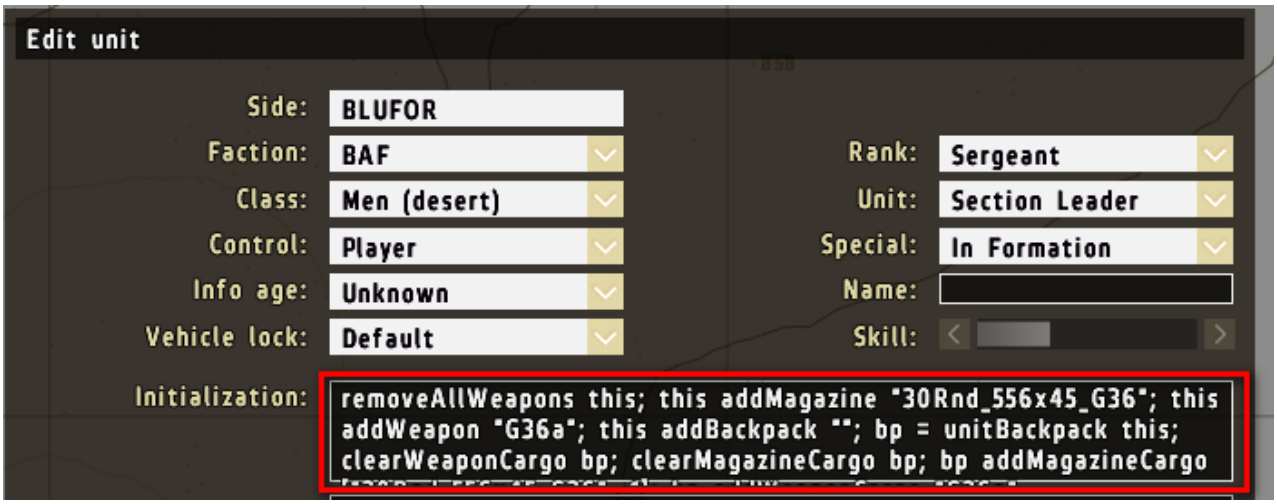
The mission maker has 2 possibilities for setting the available gear during the game.

The first possibility is to use the **init.sqf** file and set the following parameters :

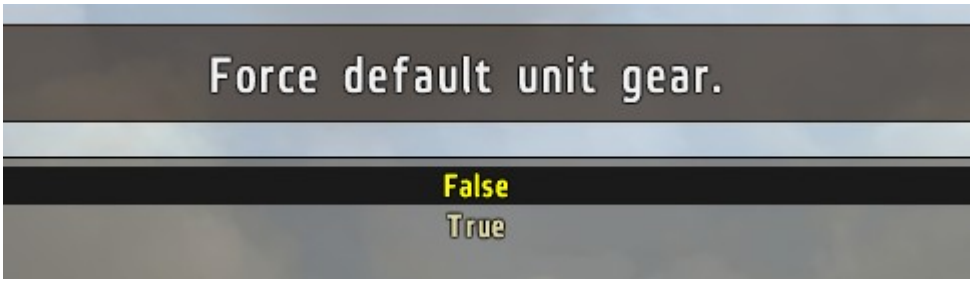
- GER_DEFAULT_ALLOWED_WEAPONS
- GER_DEFAULT_ALLOWED_MAGAZINES
- GER_DEFAULT_ALLOWED_BACKPACKS
- GER_DEFAULT_PLAYER_WEAPONS
- GER_DEFAULT_PLAYER_MAGAZINES
- GER_DEFAULT_PLAYER_BACKPACK

as explained in the file itself. This will set the "allowed gear" and the "starting gear". Players will start the game with the "starting gear" and will be able to choose different gear from the "allowed gear" list during the game. The "allowed gear" list can be modified by the admin during the game.

The second possibility is to use the init field of each unit to set its gear. This must be used in combination with the "Force default gear" game lobby parameter. This enforces each player to use different gear corresponding to what was set in the init field of his unit. For instance, this allows the mission maker to set different functions for each unit (team leader, grenadier, machinegunner, etc). During the game players will not be able to change gear but only rearm by getting the "starting gear" defined in the init field of their respective units.




```
removeAllWeapons this; this addMagazine "30Rnd_556x45_G36"; this
addWeapon "G36a"; this addBackpack ""; bp = unitBackpack this;
clearWeaponCargo bp; clearMagazineCargo bp; bp addMagazineCargo
["30Rnd_556x45_G36", 1]; bp addWeaponCargo "G36a"
```



3.2 Scripting Framework

3.2.1 How The Mission Folder Works

When you save a mission edited with the ArmA Multiplayer Mission Editor (http://community.bistudio.com/wiki/Mission_Editor) a folder is generated for this mission in a folder like **C:\Users\toto\Documents\ArmA 2 Other Profiles\eRazer\MPMissions**.

In the mission folder there is only one file, **mission.sqm**, which holds in text format all the editing operations results made in the editor (units, triggers, weather, mission name, etc).

There are 3 other files, **init.sqf**, **description.ext** and **stringtable.xml**, that are used in a special way when the mission is played in ArmA.

init.sqf (http://community.bistudio.com/wiki/6thSense.eu:EG#Order_of_Initialization) is used to execute commands when mission starts and more precisely when briefing screen is displayed. This doesn't mean that commands execute exclusively at this moment (ex : a loop can wait for a specific condition that is true in the middle of the game and at that moment some commands are executed).

description.ext (http://community.bistudio.com/wiki/Description.ext) is used to set various game parameters and GUI elements definitions .

stringtable.xml (http://community.bistudio.com/wiki/Stringtable.csv) is used to avoid writing raw message texts in scripts. Instead of that a key is used that references several translations in the stringtable.xml file. So when the script executes it displays the message using the translation corresponding to the language with which each player configured its ArmA.

stringtable.csv uses an older format than the stringtable.xml file.

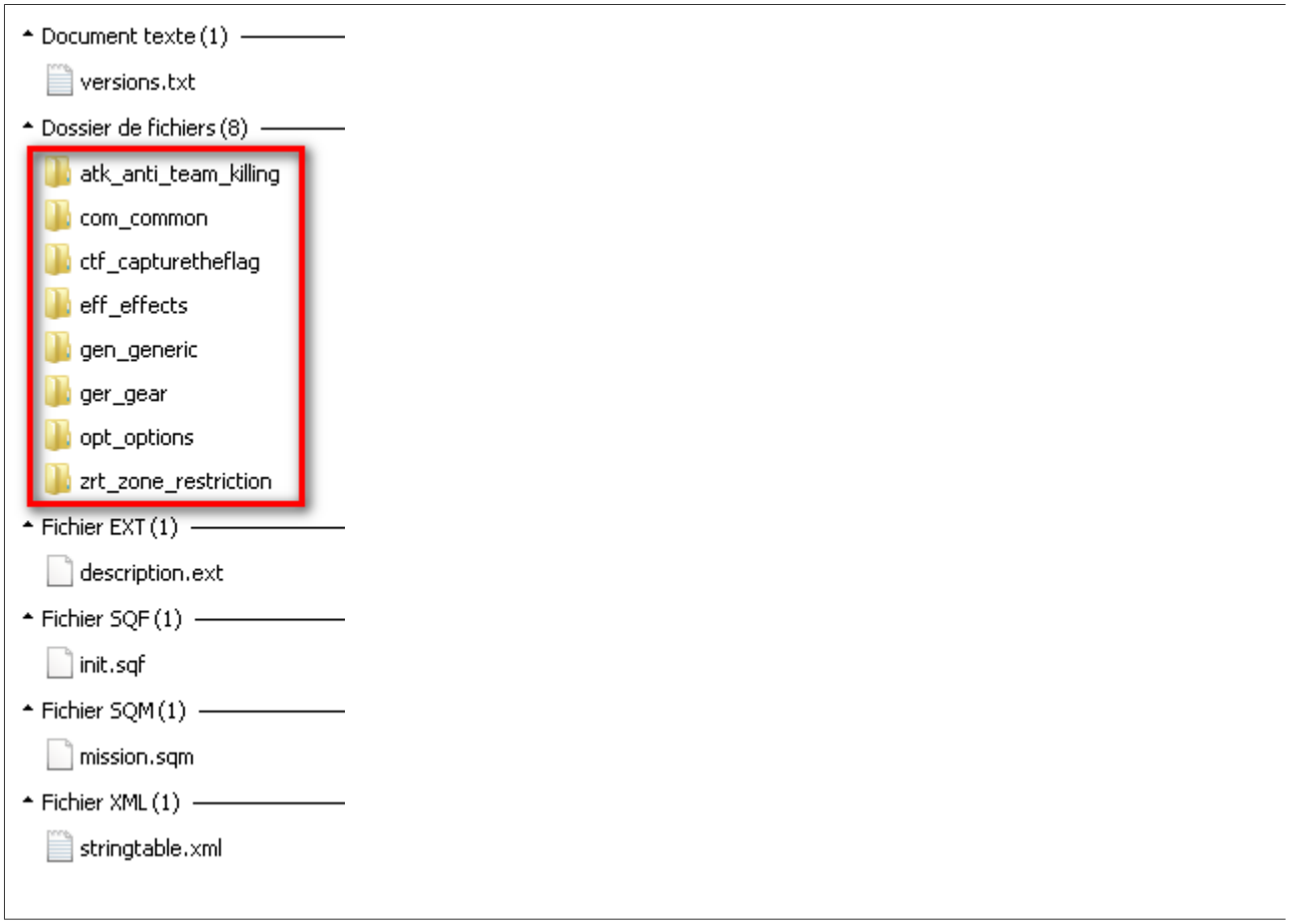
3.2.2 How The Scripting Framework Works

2.2.1 Introduction

The purpose of the scripting framework is flexibility and extensibility.

It divides the mission into modules. A module is dedicated to a specific feature of the mission (CTF, anti-teamkilling, gear selection, etc) and can use any mission files (init.sqf, description.ext, stringtable.csv, mission.sqm).

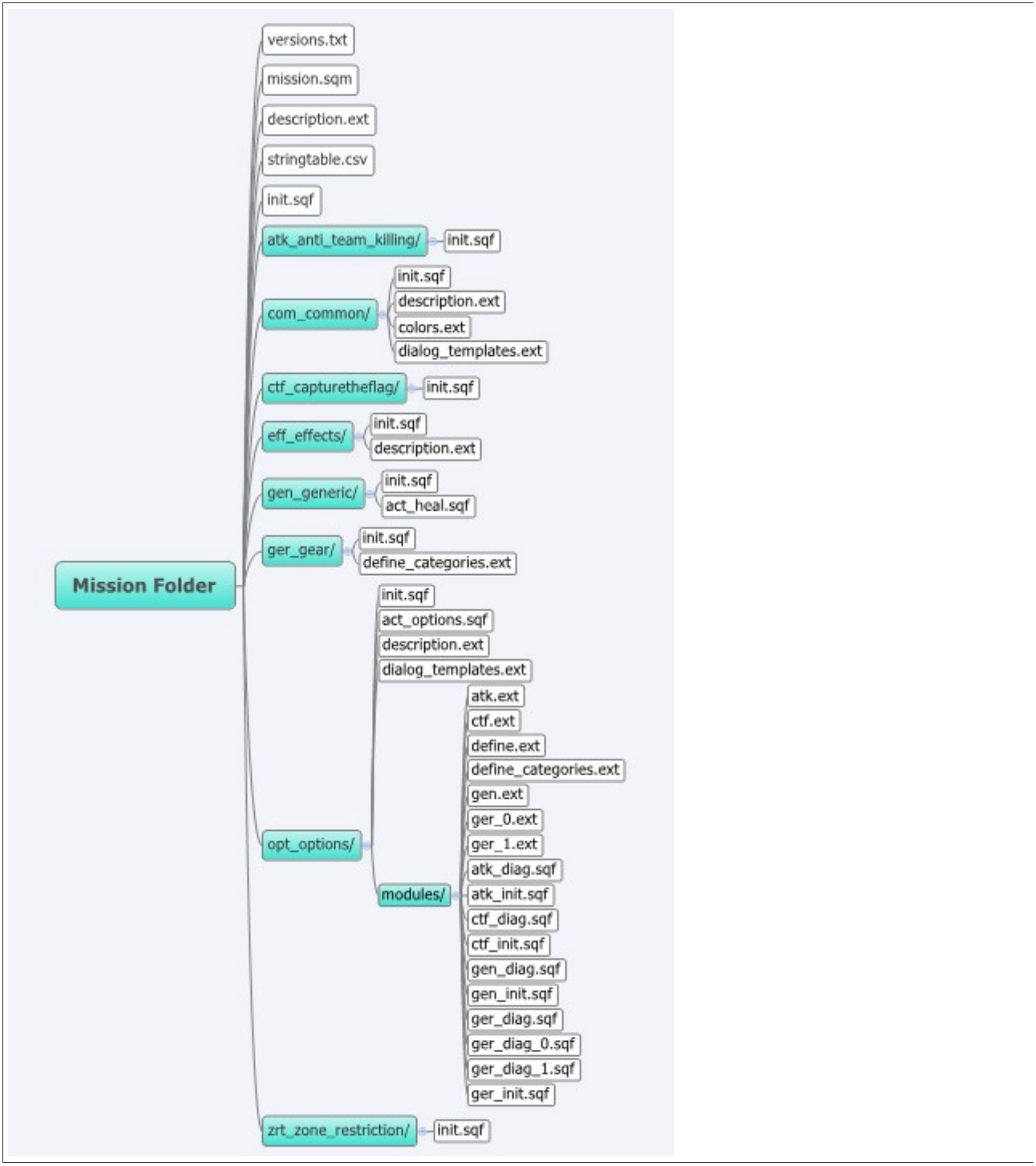
For each module there is a subfolder in the mission folder.



Each module has its own prefix (ARK, COM, CTF, EFF, etc). All global variables are named according to their module's prefix. (COM_MODULE_SCRIPTS, GEN_VIEW_DISTANCE, ZRT_GAME_AREA_LOCAL, etc).

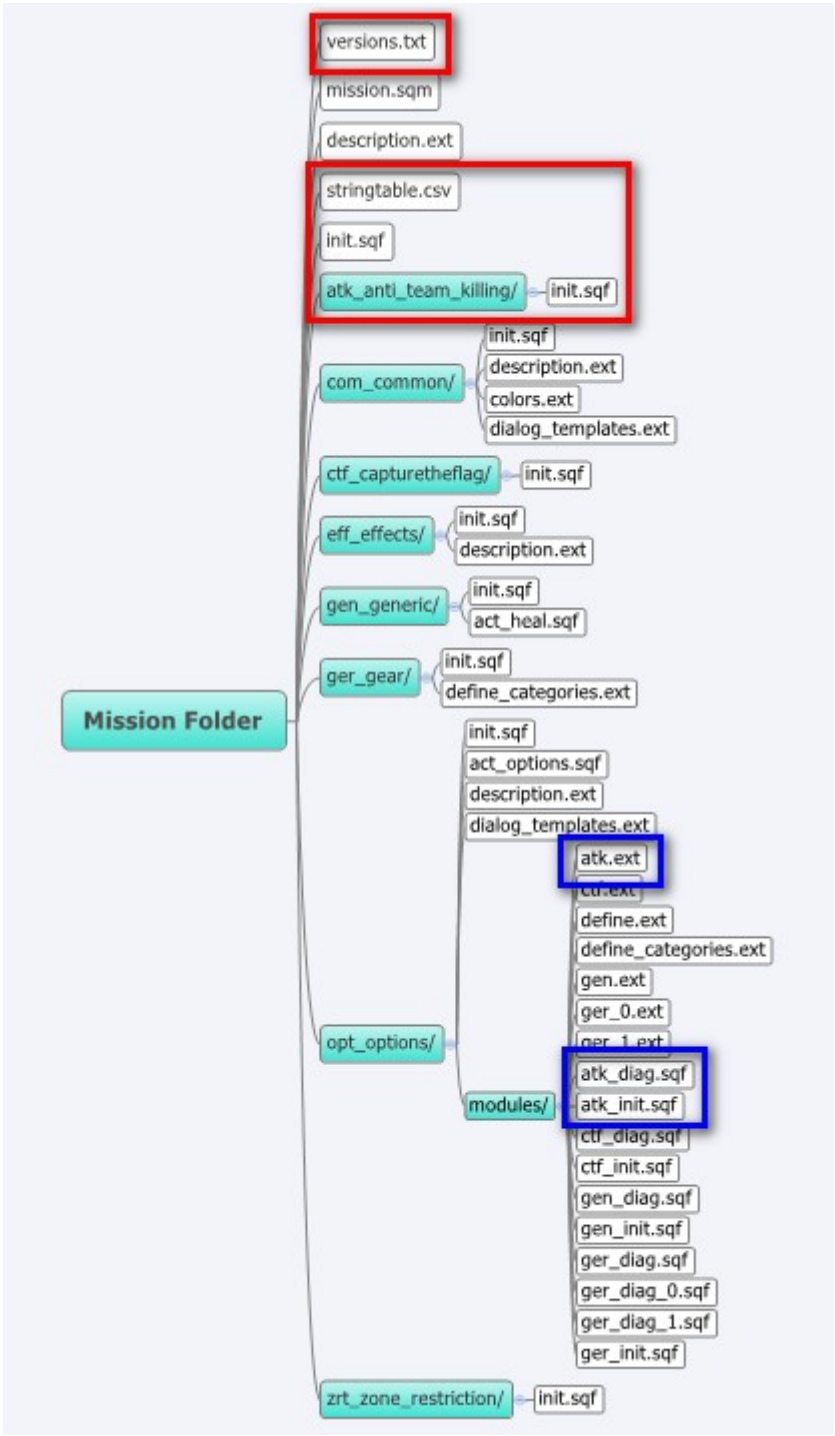
The **versions.txt** file is used to keep track of each module's version. This file is also read by a special Perl script that updates multiple missions at once and computes the mission global version from the versions indicated in the versions.txt file.

Below is the complete filesystem structure.

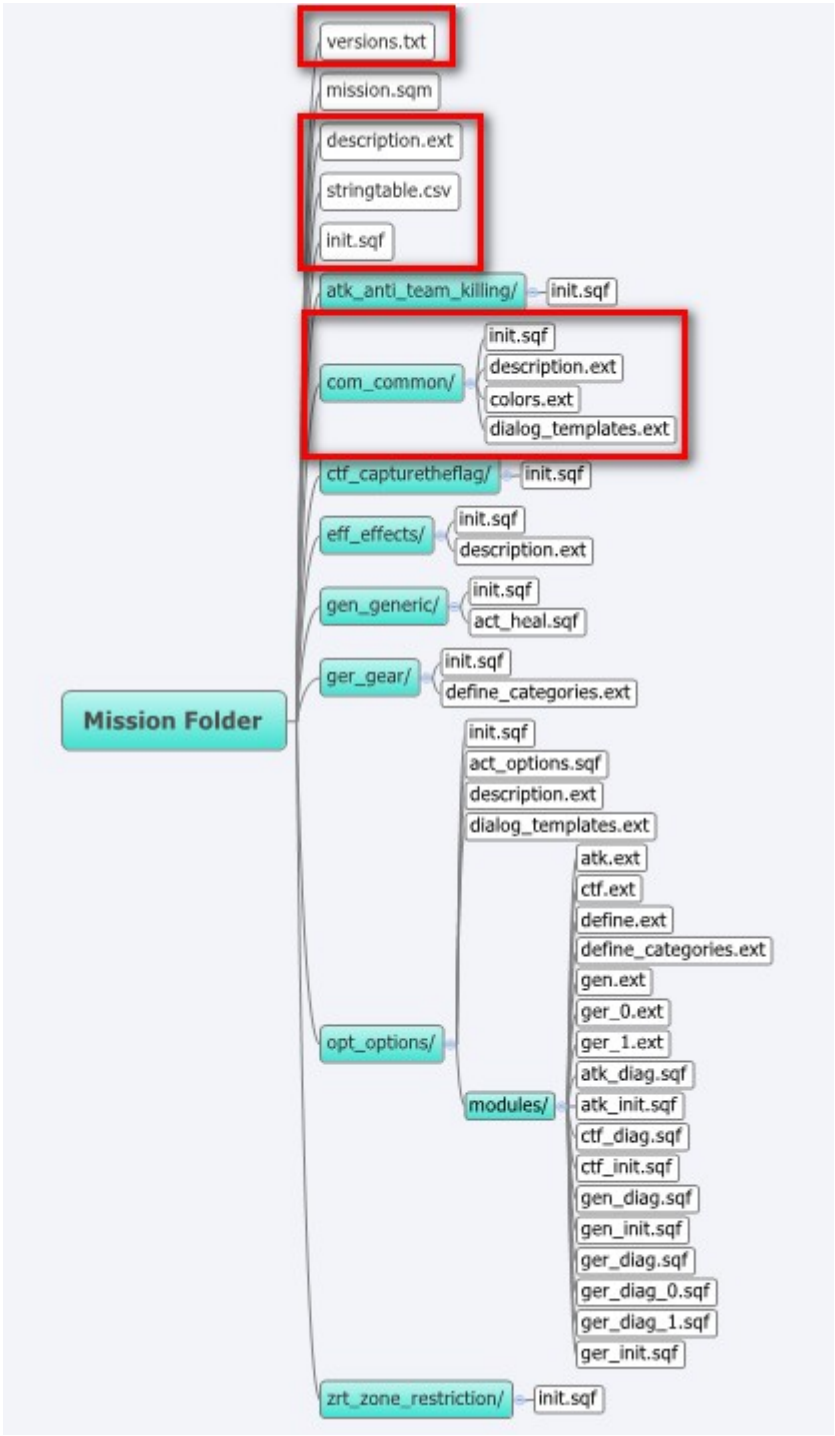


Below are the lists of files and folders used by each module (highlighted by a red rectangle). Highlighted with a blue rectangle are the elements from the Options module (which is a special module) that are also used.

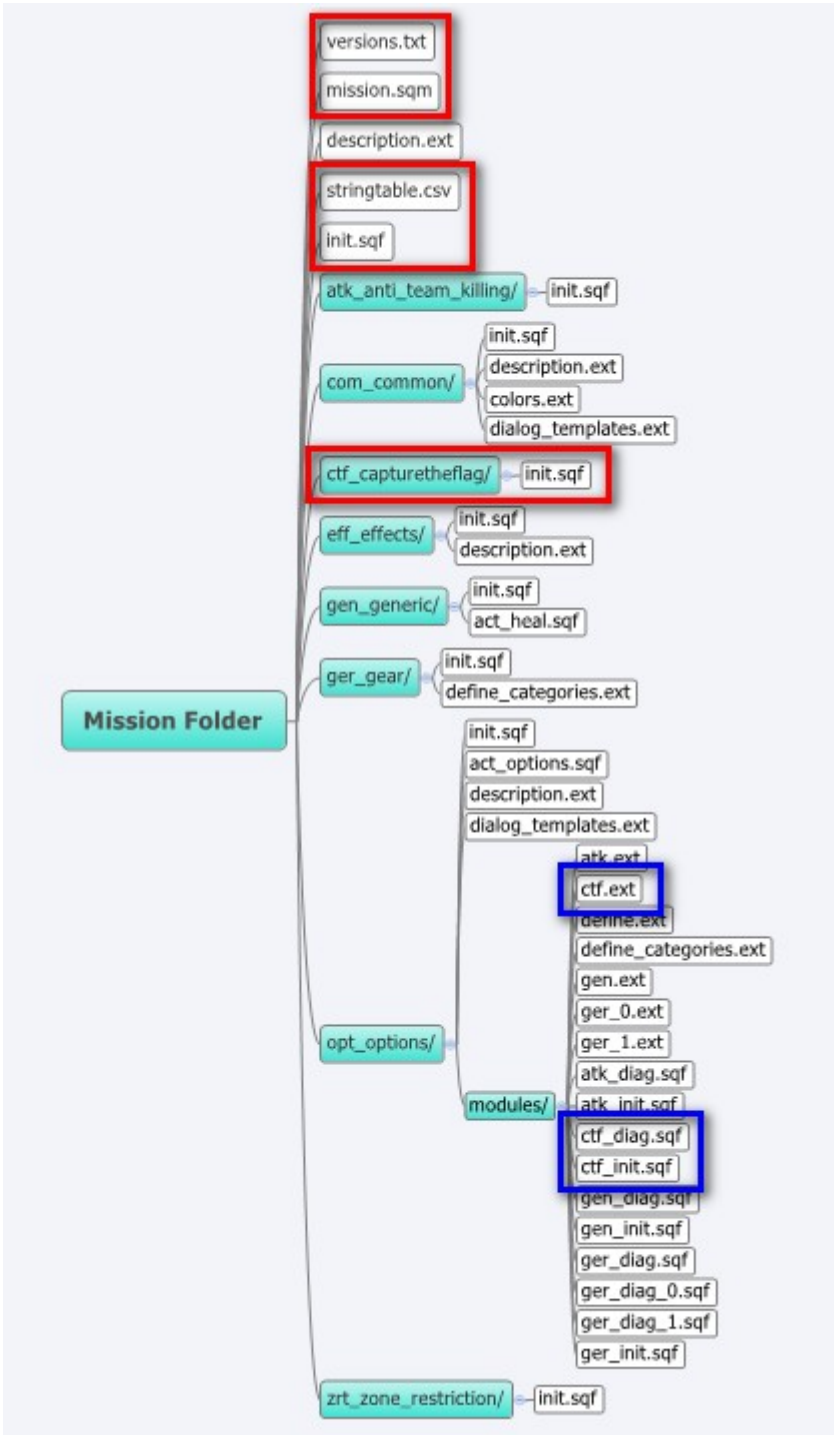
Anti-Teamkilling Module



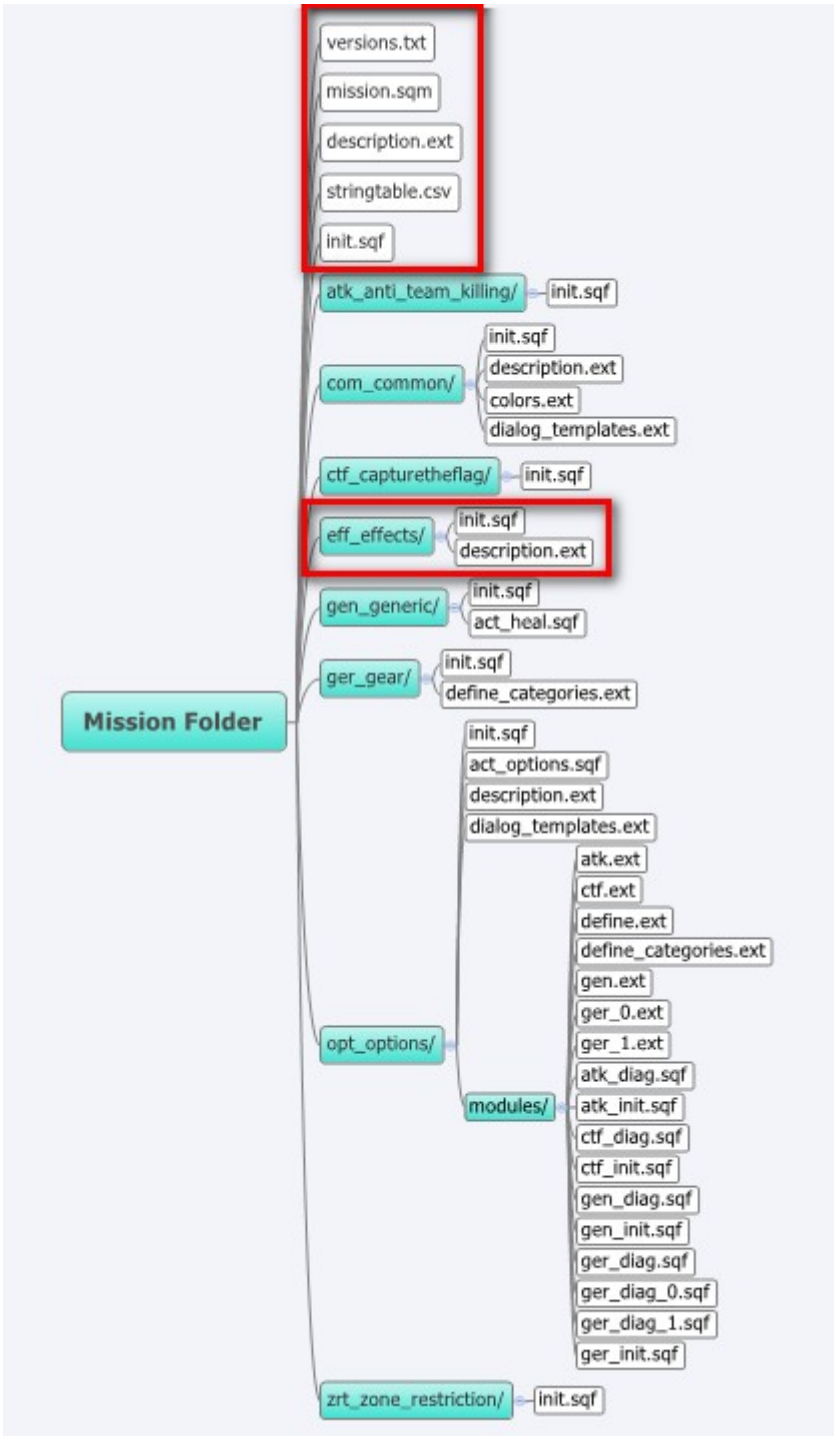
Common Module



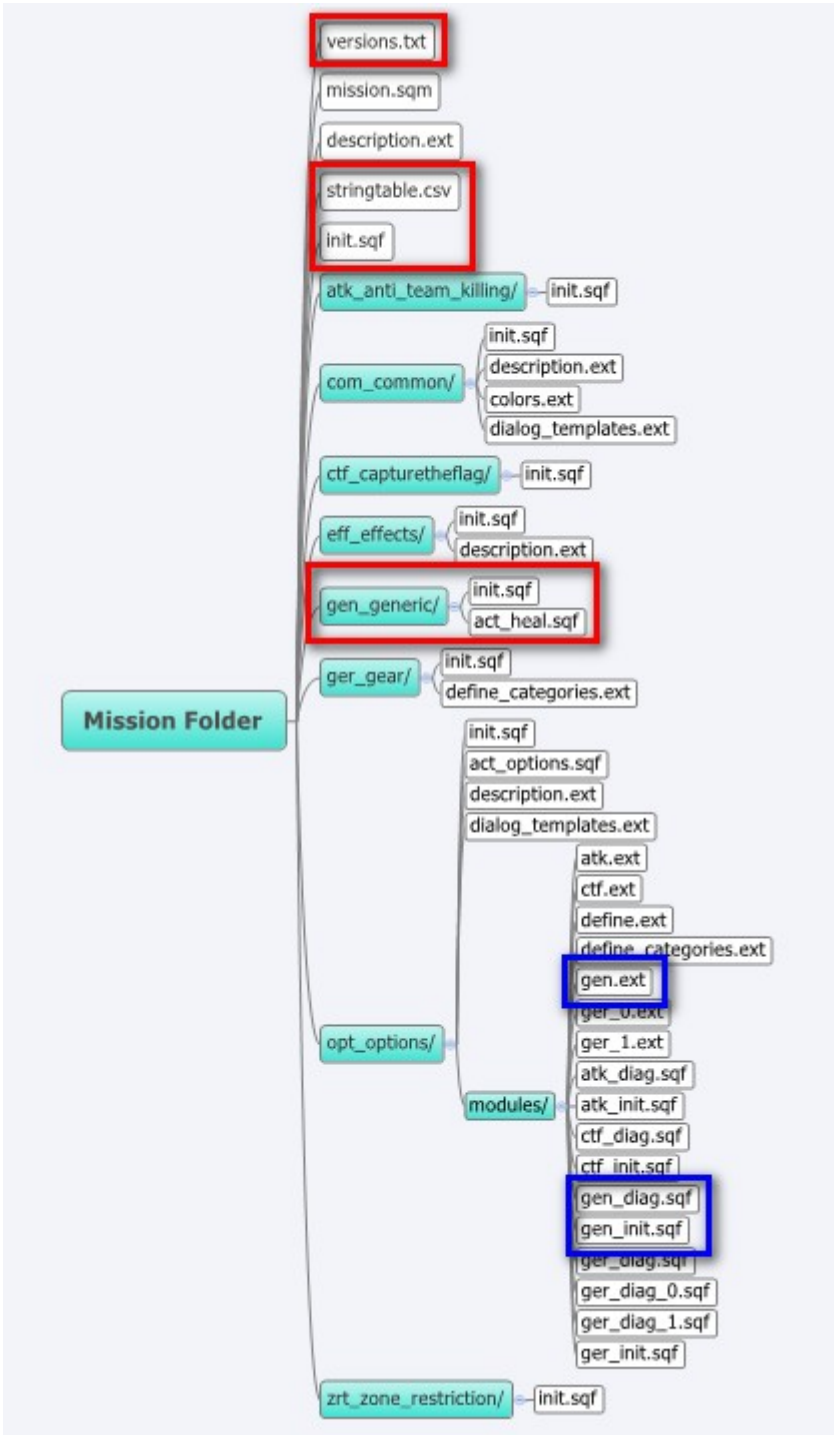
Capture The Flag Module



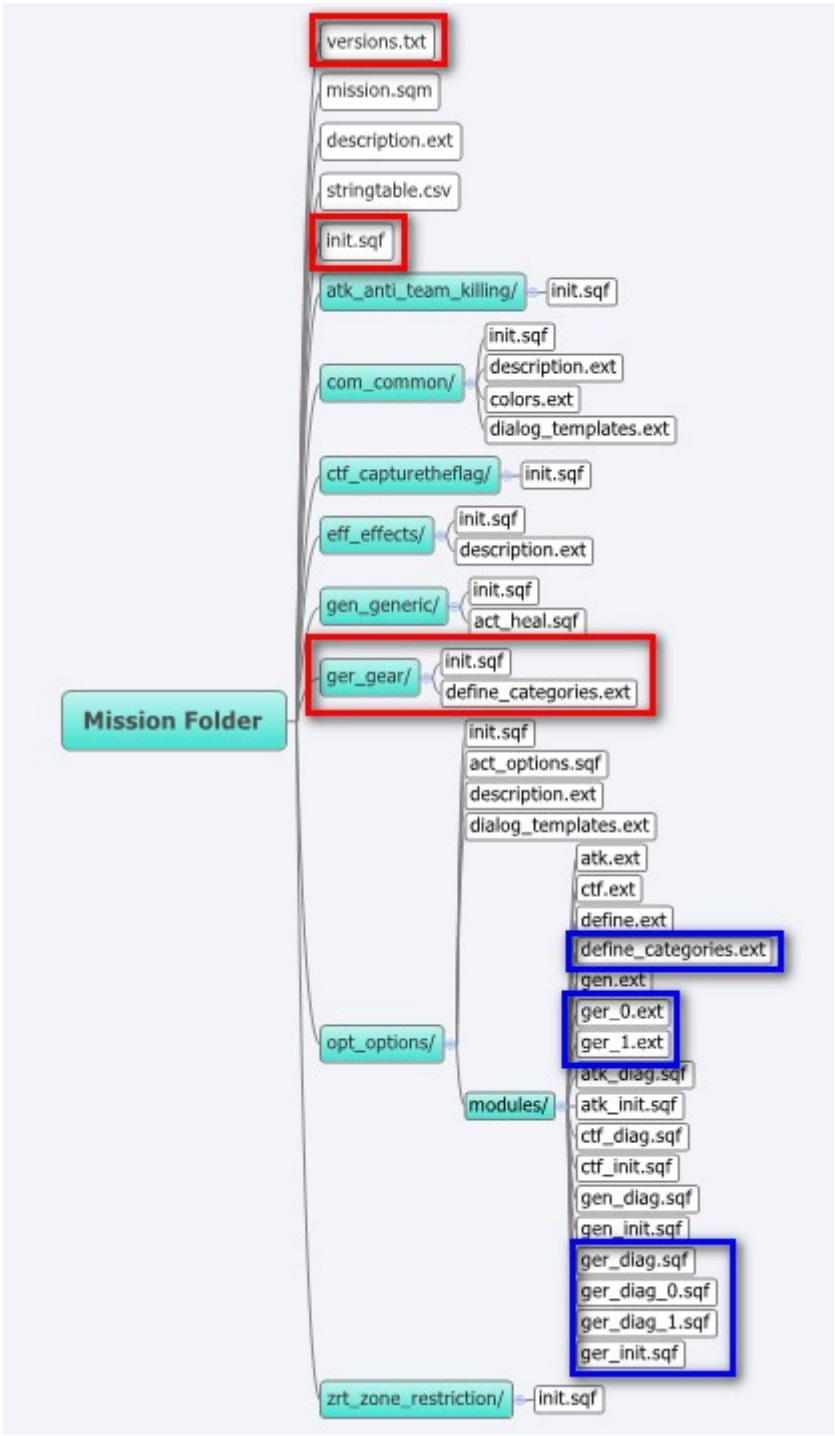
Effects Module



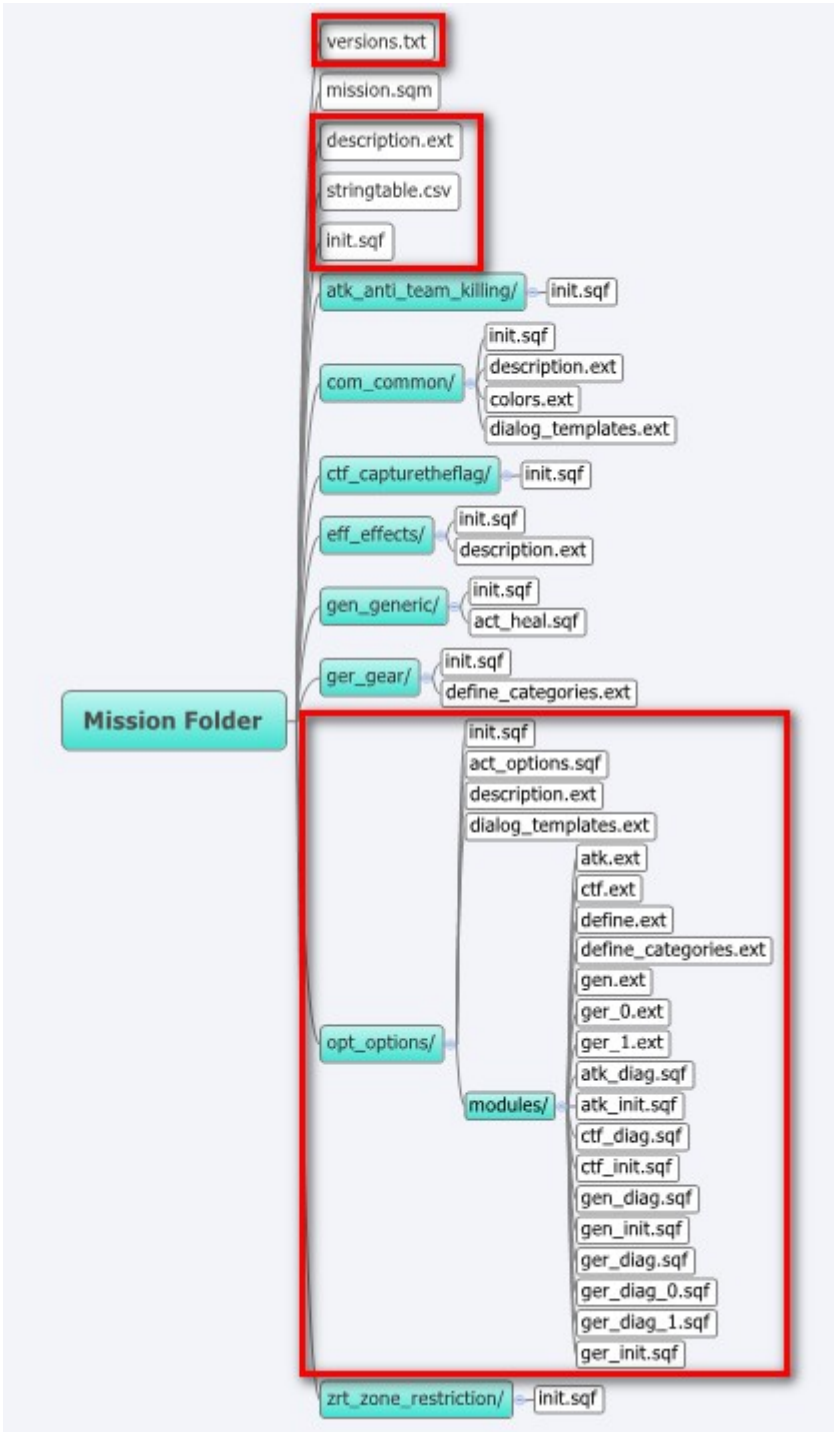
Generic Module



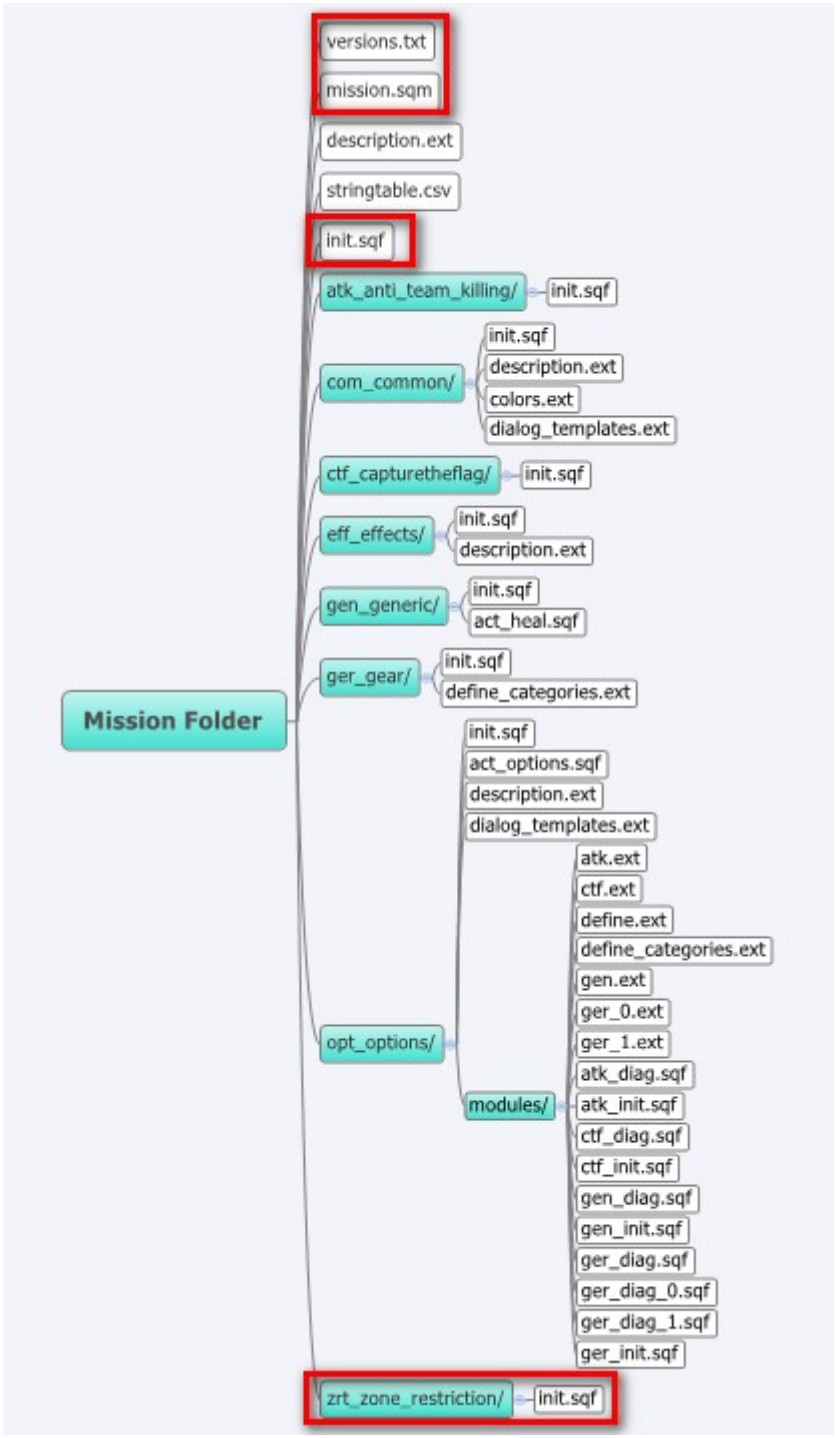
Gear Module


Mission Folder
versions.txt
mission.sqm
description.ext
stringtable.csv
init.sqf
atk_anti_team_killing/
init.sqf
com_common/
init.sqf
description.ext
colors.ext
dialog_templates.ext
ctf_capturetheflag/
init.sqf
eff_effects/
init.sqf
description.ext
gen_generic/
init.sqf
act_heal.sqf
ger_gear/
init.sqf
define_categories.ext
opt_options/
init.sqf
act_options.sqf
description.ext
dialog_templates.ext
atk.ext
ctf.ext
define.ext
define_categories.ext
gen.ext
ger_0.ext
ger_1.ext
modules/
atk_diag.sqf
atk_init.sqf
ctf_diag.sqf
ctf_init.sqf
gen_diag.sqf
gen_init.sqf
ger_diag.sqf
ger_diag_0.sqf
ger_diag_1.sqf
ger_init.sqf
zrt_zone_restriction/
init.sqf

Options Module



Zone Restriction Module



2.2.2 Adding A New Module

Depending on the new feature you want to add to your mission you can choose adding a new module with its own prefix. Or, if it is a small feature then the best choice is to add a section in the Generic Module and use the GEN prefix for all the global variables.

Here are some questions to ask yourself first :

1. Is your module using parameters that are set in the game lobby (before game starts) ?
2. Is your module using parameters that should be set in the init.sqf file (this is helpful if you want to adjust your module by hardcoding parameters in the file and not allowing players do it during game) ?
3. Are your module's parameters settable during game ? Then some GUI elements should be added to the Options dialog to allow admin or players change them during game
4. Does your module display messages during game or use the Options dialog ? If yes then entries will have to be added to the stringtable.xml file

Here are the steps for adding a new complete module :

1. **Main Module Elements**

Create Folder

Inside the mission folder create a sub-folder named <module_prefix>_my_module (ex: atk_anti_team_killing where **atk** is the module prefix).

Create Files

Does your module use his own dialog menus ? If yes then create a description.ext file where you will put these elements.

Check the Options module (opt_options/description.ext) for an example.

Your module must be using scripts. So add an init.sqf file that executes the commands you want (check the CTF module – ctf_capturetheflag/init.sqf – for an example).

Other

Does your module use graphical elements like colored text messages with special fonts ? If yes then you must update the Effects module to add these elements (check **RscTitles** class in the eff_effects/description.ext file). The best choice is to use the Effects Module to create all your

mission effects.

After you added RscTitle class elements in the Effects module, your module is now dependent on the Effects module. So you have to add this to your module's init.sqf file:

```
//////////WAIT FOR DEPENDENCIES
////////////////////////////////////////////////////////////////////////////
{while {! scriptDone _x} do {sleep 0.5}} forEach EFF_MODULE_SCRIPTS;
////////////////////////////////////////////////////////////////////////////
```

If your module depends on other modules as well then do the same as for the Effects module.

2. Lobby Parameters

To use lobby parameters accessible in your module's init.sqf file follow these steps :

Modify Common Module

Add the new lobby parameters and their possible values in the **Params** class of the com_common/description.ext file :

```
class Params
{
    class KeepGear
    {
        title = $STR_COM_PARAM_KEEPGEAR;
        values[] = {0, 1};
        default = 0;
        texts[] = {$STR_COM_FALSE, $STR_COM_TRUE};
    };

    class TimeLimit
    {
        title = $STR_COM_PARAM_TIMELIMIT;
        values[] = {900, 1800};
        default = 1800;
        texts[] = {$STR_COM_PARAM_TIMELIMIT_15MIN, $STR_COM_PARAM_TIMELIMIT_30MIN};
    };
};
```

In the com_common/init.sqf file add the global variables that will hold the lobby parameters values chosen by the admin:

```
/* Constant Description
    - lobby parameters
*/
LOBBYPARAM_KEEPGEAR = PARAMSARRAY select 0;
LOBBYPARAM_TIMELIMIT = PARAMSARRAY select 1;
```

Check For Dependency

Your module is now dependent on the Common Module. Add this in your module's init.sqf file before using the LOBBY_XXX variables:

```
//////////WAIT FOR DEPENDENCIES
////////////////////////////////////////////////////////////////////////////////////
{while {! scriptDone _x} do {sleep 0.5}} forEach COM_MODULE_SCRIPTS;
////////////////////////////////////////////////////////////////////////////////////
```

Check the Gear Module for an example (ger_gear/init.sqf).

3. Hardcoded Module Parameters

The main init.sqf (mission_folder/init.sqf) file will spawn a thread that will execute your module's init.sqf file. This way your module will execute in parallel to the other modules.

Here is the minimal entry you have to add to the main init.sqf file for your module:

```
//////////OPT - OPTIONS MODULE
////////////////////////////////////////////////////////////////////////////
/* Module Description
    - features:
        - framework that enables other modules to interact with players through a GUI menu system
        - can manage a large number of dialog controls (100 sections, each section can have 10 subsections)
        - controls and/or subsections can be allowed to admin only
    - dependencies:
        - requires all modules that use the options dialog
        - ZRT module (ZRT_BASE_EAST_TRIGS, ZRT_BASE_WEST_TRIGS)
    - performance considerations: none
    - dialogs:
        - uses IDD 1
*/




/* Module Start */
OPT_MODULE_SCRIPTS = [];
OPT_MODULE_SCRIPTS = OPT_MODULE_SCRIPTS + [execVM "opt_options\init.sqf"];
////////////////////////////////////////////////////////////////////////////
```

Check the main init.sqf file for examples on how to add hardcoded parameters.

Leave the Effects Module at the bottom of the init.sqf file because it uses a special command that blacks out the screen after briefing. This seems not to work when the command is put elsewhere:

```
// black screen during initialization
cutText ["", "black faded"];
```

4. Options Dialog

This is a complex step. First you must know exactly what kind of GUI elements you need. The best is to look at some examples: opt_options/modules/ger_0.ext, ger_1.ext, atk.ext.

The Options menu is divided first in sections (top left drop-down list) then in subsections (top right drop-down list). The ger_0.ext file holds the GUI elements of the first subsection of the Gear Module.

Before creating the .ext files in the opt_options/modules/ folder start by updating the opt_options/modules/define.ext file that holds the IDCs of all the GUI elements you will be adding (check the existing define.ext for examples).

If all the GUI elements you want to put in a subsection do not fit then you can use a control group as it is done in the Gear Module (ger_0.ext and ger_1.ext).

Once you updated the define.ext and created the .ext file (or files in case you have several subsections) you have to update the opt_options/description.ext file like this:

```
/* *******GER - GEAR MODULE */
/* ************************************************************************** */

#include "modules\ger_0.ext"
#include "modules\ger_1.ext"

/* ************************************************************************** */
```

At this point you can visualize the new GUI elements but they will be empty and overlap the existing GUI elements.

The next step is to write the commands that will fill the drop-down lists, set button actions, etc.

There are 2 files to create in the opt_options/modules/ folder:

- the <prefix>_init.sqf file which is mainly used to define the functions used for the buttons, and more generally everything that doesn't need to execute each time the options dialog is opened (check atk_init.sqf for an example)
- the <prefix>_diag.sqf file is executed each time the options dialog is opened; it is used to fill drop-down lists, set button actions, add event handlers for drop-down lists, write text, etc.

Also it is manadatory to update the stringtable.xml file as eplained in the following section, for naming the sections and possible subsections you may have added.

5. Localization with stringtable.xml

There are situations where you want to display text on a button or display a title text in the center of the screen. That's when you need to use the stringtable.xml file.

The stringtable.xml is divied in sections for each existing module that uses it:

```
219
220
221
222
223   <!-- //////////ATK - ANTI TEAM-KILLING MODULE -->
224   <!-- ////////////////////////////////////////////////////////////////////////////// -->
225   <Key ID="STR_ATK_MSG">
226       <English>You are punished for teamkilling %1 or more players in less than %2 seconds.</English>
227       <French>You are punished for teamkilling %1 or more players in less than %2 seconds.</French>
228       <Italian>You are punished for teamkilling %1 or more players in less than %2 seconds.</Italian>
229       <Spanish>You are punished for teamkilling %1 or more players in less than %2 seconds.</Spanish>
230       <German>You are punished for teamkilling %1 or more players in less than %2 seconds.</German>
231   </Key>
232   <!-- ////////////////////////////////////////////////////////////////////////////// -->
233
234
235
236
237
238
239
240
241
242
243   <!-- //////////CTF - CAPTURE THE FLAG MODULE -->
244   <!-- ////////////////////////////////////////////////////////////////////////////// -->
245   <Key ID="STR_CTF_SIDENAME_WEST">
246       <English>West</English>
247       <French>West</French>
248       <Italian>West</Italian>
249       <Spanish>West</Spanish>
250       <German>West</German>
251   </Key>
```

The section of the Options module is special because it is sub-divided in another level of sections corresponding to the different modules that integrate with the options module:

```
318
319   <!-- //////////OPT - OPTIONS MODULE -->
320   <!-- ////////////////////////////////////////////////////////////////////////////////////////// -->
321   <Key ID="STR_OPT_OPTIONS">
322       <English>Options</English>
323       <French>Options</French>
324       <Italian>Opzioni</Italian>
325       <Spanish>Opciones</Spanish>
326       <German>Optionen</German>
327   </Key>
328   <Key ID="STR_OPT_BTN_UPDATE">
329       <English>Update</English>
330       <French>Update</French>
331
332
333
334
335
336   <!-- **********ATK - ANTI TEAM-KILLING MODULE -->
337   <!-- ************************************************************************************ -->
338   <Key ID="STR_OPT_SECT_0">
339       <English>Anti Team-Killing</English>
340       <French>Anti Team-Killing</French>
341       <Italian>Anti Team-Killing</Italian>
342       <Spanish>Anti Team-Killing</Spanish>
343       <German>Anti Team-Killing</German>
344   </Key>
345   <Key ID="STR_OPT_SECT_0_SUBSECT_0">
346       <English>Anti Team-Killing</English>
347       <French>Anti Team-Killing</French>
348       <Italian>Anti Team-Killing</Italian>
349       <Spanish>Anti Team-Killing</Spanish>
350       <German>Anti Team-Killing</German>
351   </Key>
```

If your new module uses the Options menu then it has at least one section and one subsection where you place the GUI elements of your module. Besides defining text translations for GUI elements used by your module you need to add at least 2 translations for a section and a subsection:

```
373   <!-- **********ATK - ANTI TEAM-KILLING MODULE -->
374   <!-- ************************************************************************************ -->
375   <Key ID="STR_OPT_SECT_0">
376       <English>Anti Team-Killing</English>
377       <French>Anti Team-Killing</French>
378       <Italian>Anti Team-Killing</Italian>
379       <Spanish>Anti Team-Killing</Spanish>
380       <German>Anti Team-Killing</German>
381   </Key>
382   <Key ID="STR_OPT_SECT_0_SUBSECT_0">
383       <English>Anti Team-Killing</English>
384       <French>Anti Team-Killing</French>
385       <Italian>Anti Team-Killing</Italian>
386       <Spanish>Anti Team-Killing</Spanish>
387       <German>Anti Team-Killing</German>
388   </Key>
```

The **0** in **STR_OPT_SECT_0** and **STR_OPT_SECT_0_SUBSECT_0** corresponds to the number of the section of your module in the **opt_options\modules\define.ext** file:

```
108  /* *******ATK - ANTI TEAM-KILLING MODULE */
109  /* ************************************************************************** */
110  // SECTION NUMBER
111  #define _ATK_SECTNUM_ 0
112
```

In conclusion your module can have two sections. One of its own, independent of other module sections, where you can generally place text messages that don't use the Options dialog. And it can also have a subsection inside the Options module section where you place translations for all the GUI elements and messages displayed in the Options dialog.

2.2.3 Extending The Generic Module

If your module is a very simple one (add briefing notes, prevent players from camping, crawling, remove dead units from game area, etc) then you can extend the Generic module whose purpose is exactly this: a collector of all extensions that don't need a complete new module (like Gear module, Zone Restriction module or CTF module).

The Generic module works like any other module so you can use the steps from the section describing how to add a complete module.

2.2.4 Removing An Existing Module

To remove any module you just need to inverse the steps used to create it.

Here are the general steps for removing a module:

1. Check if other modules depend on it (dependencies should be mentioned in the main **init.sqf** file) and decide how to remove these dependencies
2. Remove entries from **versions.txt**, **init.sqf**, **description.ext** and **stringtable.xml** files
3. Remove references to this module used inside the mission itself (**mission.sqm** file)
4. Remove the module folder
5. Remove entries from the Options module