

How to create a database for the Set-Pos-In-Game script

1. Introduction

Set-Pos-In-Game doesn't have a database for the addon you're using? It's existence would improve your work? Create your own database file! It's not very complicated.

Let's face it. Right now I'm focusing right on the other projects and I don't plan to add any new bases. However, you can help expand the script!

The goal is to cover all object addons. If remaining OFP players would create at least one base then this objective would be easily achieved.

2. What is a database?

Database for the Set-Pos-In-Game script is a function containing variables (arrays) filled with lists of objects. When script is loading bases, those arrays are read into memory and later they're used to write list of objects for "Insert Object" menu.

3. The Plan

Choose for which addon you'll create a database. One pbo file – one base. Get familiar with the addon. What objects does it have? How you'll categorize them? I recommend writing all the information on a piece of paper: names, number of categories and objects assigned to them.

4. Database header

Once you have a plan you can start writing new database file. In the mission folder create a text file and give it the same name as the pbo file. Add *.sqf* extension at the end.

`<PBO name>.sqf`

For example:

PBO name	Database name
editorupdate102.pbo	editorupdate102.sqf
AGS_build.pbo	AGS_build.sqf

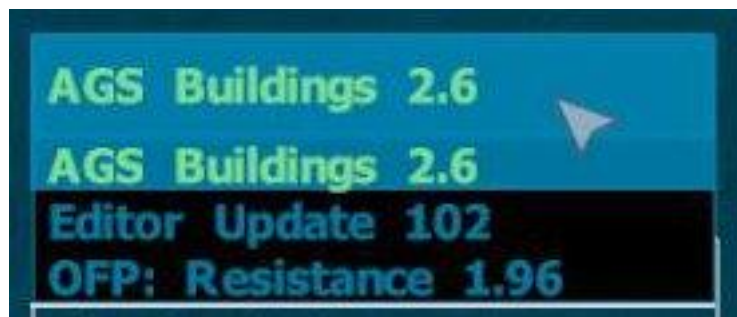
Open your file in the text editor. At the beginning define two local variables:

```
_addon = <string>;  
_name = <string>;
```

First one is the base identifier. It must not contain spaces. Second variable is a name displayed in the drop-down list. For example:

```
_addon = "agsbuildings";  
_name = "AGS Buildings 2.6";  
  
_addon = "editorupdate102";  
_name = "Editor Update 102";
```

That's how it looks in-game:



Important note about syntax:

In OFP functions (they are indicated by *.sqf* extension) every command line must end with a semicolon. Otherwise you'll receive an error message.

Next thing: copy this code into your file:

```
if (Format ["%1", SPIG_IO] == "scalar bool array string
0xfcffffef") then {SPIG_IO=[_addon]; SPIG_IOname=[_name]} else
{SPIG_IO=SPIG_IO+[_addon]; SPIG_IOname=SPIG_IOname+[_name]};
```

It's a single line. Do not modify it. It will add your database to the global array used by Set-Pos-In-Game script.

Last thing to do in a header is to write list of categories. Syntax:

```
SPIG_<identifier> = [ <category name>, ... ];
```

Second part of variable name should be an identification string you've wrote earlier (_addon). In the array write list of names for all planned categories (string values). Examples:

```
SPIG_agsbuildings = ["City Buildings", "City Objects"];

SPIG_editorupdate102 = ["Bush", "Castle", "Church", "Effect", "Fence",
"Forest", "Grass", "House", "Miscellaneous", "Road", "Rock", "Sign", "Tree"];
```

Move on to the next chapter when ready.

5. Writing List of Objects

For every category you have to create two variables. Their names are similar to the categories array name.

```
SPIG_<identifier><number> = [ ];  
SPIG_<identifier><number>name = [ ];
```

Where <identifier> is the `_addon` variable. <number> is the number of category (starting from zero). So 0 – first category, 1 – second, 2 – third, etc. Examples:

```
SPIG_agsbuildings0 = [ ];  
SPIG_agsbuildings0name = [ ];  
  
SPIG_agsbuildings1 = [ ];  
SPIG_agsbuildings1name = [ ];  
  
SPIG_editorupdate1020 = [ ];  
SPIG_editorupdate1020name = [ ];
```

Now the exhausting part. Fill variables with objects class names and their display names. Syntax:

```
SPIG_<..> = [ <class>, <class>, ... ];  
SPIG_<..>name = [ <name>, <name>, ... ];
```

All values must be string types. These arrays are related to each other. Object with class name with index X (position in the array) will have name as the value, from the other array, with the same index. Here's the graphical representation of this relation:

SPIG_agsbuildings0 = ["AGS_build_c",	"AGS_hospital",	"AGS_build5", ...
	↕	↕	↕
SPIG_agsbuildings0name = ["Cinema",	"Hospital",	"House (4flr)", ...

What if addons have a stringtable:

Instead of standard name write string name from the stringtable. For example:

```
SPIG_mapheaps0 = [ "MAP_Coalheap_s1", ... ];  
SPIG_mapheaps0name = [ "STR_Coalheap_s1", ... ];
```

It's more complicated with categories. First, translate their names using dictionary.
Second – write condition(s) to change array if user have a localized game. Syntax:

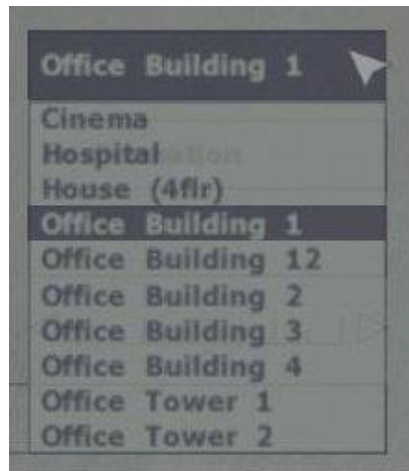
```
If (localize "STR_WEST" == <string>) then  
{  
    SPIG_<categories> = [ <translated>, <trans>, ...];  
};
```

STR_WEST is the name of a string taken from the global stringtable
(\res\bin\stringtable.csv). <string> should be it's value in desired language. For
example a German translation will include:

```
SPIG_mapheaps = [ "Coal", "Earth, ... ];  
If (localize "STR_WEST" == "Westen") then  
{  
    SPIG_mapheaps=["Kohlehaufen","Erdhaufen",...];  
};
```

My method to fill the arrays:

First thing to do is to make a screenshot of a mission editor showing list of objects:



They are sorted alphabetically here. Second step is to extract addon and open the "config.cpp" file. Then look for needed object (e.g.: find "Cinema"), copy class name and paste it to array. Here's how the process looks:

```
Class AGS_build_c : ...  
{  
    ...  
    displayName = "Cinema";  
    ...  
};
```

} Config



```
SPIG_agsbuildings0 = ["AGS_build_c", ...  
SPIG_agsbuildings0name = ["Cinema", ...
```

} Database

If addon has a “Stringtable.csv” then the first step is to find a string name:

```
STR_Coalheap_s1,Coal Heap 1
```

Then look for it in “Config.cpp”:

```
Class AGS_build_c : ...
{
    ...
    displayName = STR_Coalheap_s1;
    ...
};
```

At last – add names to the array:

```
SPIG_mapheaps0 = [ "MAP_Coalheap_s1", ...
SPIG_mapheaps0name = [ "STR_Coalheap_s1", ...
```

Seems to be tiring? Yes but it's also rewarding. You've got a clean database, friendly for the user.

I also would like to mention that sorting names alphabetically is **not always** helpful. For example: if you've got objects like:

```
Barrel
...
Small Barrel
```

Then it will be tiring to move from one end of the list to the other. The better way is to alter the name and write:

```
Barrel
Barrel Small
...
```

6. Testing

Don't try to write everything at once. When you finish one group of arrays - test them in the game.

First you need to open *Start.sqs* in the Set-Pos-In-Game script directory. Modify line 29:

```
SPIG_IOCreateBase = true
```

It will extend the "Insert Object" menu.

To load your database write in mission's *Init.sqs*:

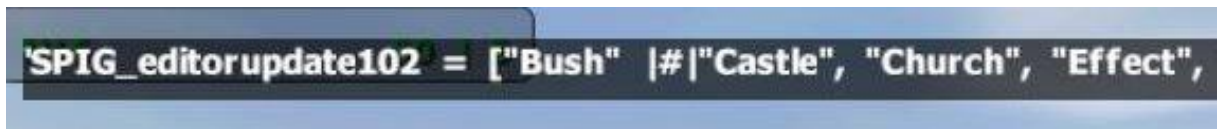
```
Call loadFile <name>
```

For example:

```
Call loadFile "editorupdate102.sqf"
```

Then open "Insert Object" menu. Your base should be on the top of the list.

If you'll encounter error message. For example:



Then it's most likely a syntax error. Go back to file and recheck your code. In this image you can notice that there is a missing colon in the array.

On the upper right corner you'll notice additional information:



ID – index (object's position in array).

CamDist – distance from the object to camera.

Class – object's class.

Count classes – number of elements in the first array.

Count names – number of elements in the second array.

Both arrays should have the same number of elements so if the last two values are not the same then you should go back to the file and check arrays.

7. Setting a camera

There are smaller and bigger objects. It's good if the camera can keep up with them. Database should also store information about camera position. Although it's optional.

Write an extra array:

```
SPIG_<identification><number>cam = [];
```

For example:

```
SPIG_mapheaps0 = ["MAP_Coalheap_s1", ... ];  
SPIG_mapheaps0name = ["STR_Coalheap_s1", ... ];  
SPIG_mapheaps0cam = [ [3,50], [4,50] ];
```

This array contains other arrays. Here's a syntax for interior matrix:

```
[<id>, <zoom>]
```

Where <id> is index (element's position in the array) and <zoom> is the distance from camera to the object.

When you browse objects press + and – keys to find proper camera distance. After then press *F1* and the value will be saved to a text file. Repeat that process with every other object (if necessary). When you're done go to \fwatch\mdb directory and find text file with your identifier. Inside you should see something similar to this:

```
category0 = [ [-1,-1], [0,10], [1,50], [2,50] ]
```

Ignore the first array ([-1,-1]). It's only there to make fwatch command work properly. Copy all the other values and paste them into your database cam array. Restart mission and check if it works.

It's desirable to have adjacent values similar to each other (if possible). For example if you have:

```
[0,48], [1,50]
```

It's much better to write:

```
[0,50], [1,50]
```

If most of the objects in the same category have similar camera settings then instead of defining cam zoom for every single one you can use "all" string value. Here's how it works:

```
SPIG_milobj0cam = [ ["all",7] ];
```

Camera distance to an object will be equal to 7 meters for all elements in this group.

If there are exceptions write normal numeric values and put "all" at the end of the array:

```
SPIG_milobj0cam = [ [0,20], ["all",7] ];
```

Camera zoom will be equal to 7 except for the first object – it will be 20 meters.

8. Enhancing

If the addon uses "Stringtable.csv" and you have wrote string names in the arrays then copy below code at the end of the database file:

```
_a = 0;
while "_a < count ( call ({SPIG_}+_addon) )" do
{
    _name = Format ["SPIG_%1%2name", _addon, _a];
    _b = 0;
    _string = "";

    while "_b < count (call _name)" do
    {
        Format ["_string = localize (%1 select _b)", _name] forEach [0];
        Format ["if (_string == {}) then {_string=(%1 select _b)}", _name] forEach [0];
        Format ["%1 set [_b, _string]", _name] forEach [0];
        _b = _b + 1;
    };

    _a=_a+1;
};
```

Don't change anything in it. It will localize all the strings.

How to get rid of prefixes?

Localized names may contain prefixes. For example:



To remove them you have add extra code in the second while loop. Exactly between second and third Format command:

```
while "_b < count (call _name)" do
{
    Format ["_string = localize (%1 select _b)", _name] forEach [0];
    Format ["if (_string == {}) then {_string=(%1 select _b)}", _name] forEach [0];

    <Write you code here>

    Format ["%1 set [_b, _string]", _name] forEach [0];
    _b = _b + 1;
};
```

Line to add:

```
if (_a == 0) then {_string = call loadFile format[{:string range "%1" 4 100}, _string];}
```

Change the condition inside the first bracket. `_a` is the number of category. So if `_a==0` then you'll remove all prefixes from the first category. `_b` is the object's ID so you are able to remove prefix from a single object name.

Then modify the number after expression `"%1"` (in above example it's four). It sets how many letters will be cut from the name.

9. Afterword

If you need more examples go to `\Set-Pos-In-Game\db\`. There are a lot of files there so you won't have problem with finding a template.

If you have any problems or questions send me an e-mail for an adress: faguss@o2.pl.

If your database is 100% done then submit it to the same adress and I will include the file in the next Set-Pos-In-Game script update.

Have fun

Faguss