Einstellungen für Mods

Table Of Contents

[Blocked Image: https://ftp.train-fever.net/flaggen/gb.png] The English version can be found here: Settings for Mods

1 Einleitung

Mit hat es schon öfter gestört, dass man, wegen kleiner Anpassungen an die Vorlieben der Nutzer, zwei nur leicht verschiedene Mods oder einen Hauptmod mit Zusatzmods anbieten musste. Meine Lösung war ein einfaches Lua Script (settings.lua), das im Ordner des Mods liegt und alle Optionen in der Form `Option = Wert` enthält:
Code: settings.lua

```
1. return {
2. option1 = true,
3. option2 = 4,
4. }
```

Diese Datei kann dann entweder direkt ausgelesen werden oder man nutzt die dafür vorgesehenen Funktionen meines ModUtil Scripts (einfach den Inhalt des angehängten Archivs in den Mod Ordner kopieren). Erstgenannte Variante ist etwas komplizierter und nur zu empfehlen, wenn man sich mit Lua auskennt. Daher werde ich hier auch nur die Variante mit dem ModUtil Script vorstellen.

2 Optionen definieren

Um den Mod Nutzern die Mühe zu sparen, die settings.lua selbst mit einem Texteditor zu bearbeiten, war @Xanos so freundlich dem TPFMM ein weiteres Fenster zu spendieren (eines großes Danke dafür👍), in dem die Optionen grafisch dargestellt und bearbeitet werden können. Natürlich kann der TPFMM das aber nur leisten, wenn er die möglichen Optionen und deren erlaubte Werte kennt.
Wir haben uns dazu entschlossen, die dafür nötigen Informationen aus der mod.lua auszulesen, dazu müssen sie dort aber erst definiert werden. Das geht über den Eintrag `settings`, zusätzlich zum Eintrag `info`, wo man beliebig viele Optionen definieren kann:
Code: mod.lua

```
1. function data()
2. return {
3. info = {
4. -- ...
```

```
5. },
6. settings = { -- optionen },
7. }
8. end
```

Jede Option wird dann durch verschiedene Parameter bestimmt:

Code

```
1. option1 = { -- interne ID der Option, mit der sie später ausgelesen werden kann
2. type = "number", -- "boolean", "number", "string", "table" (Seit TPFMM Version 1.0.32)
3. name = _("Option1"), -- Name der im TPFMM für diese Option angezeigt wird (kann mit der strings.lua übersetzt werden)
4. -- optionale Parameter
5. description = _(""), -- genauere Beschreibung was diese Option beeeinflusst, wird im TPFMM als Tooltip angezeigt
6. default = 0, -- Standardwert, wenn nicht angegeben, wird für "boolean" false, für "number" 0, für "string" "" und für "table" {} genutzt
7. min = -2147483647,
8. max = 2147483648, -- minimal und maximal möglicher Wert (nur bei "number")
9. values = { -- mögliche Werte für die Mehrfachauswahl (nur bei "table")
10. {
11. text = _("this is a 100"), -- Text der im TPFMM für diese Option angezeigt wird
12. value = 100, -- Wert, der in die Tabelle geschrieben wird, wenn diese Option ausgewählt wurde
13. },
14. },
15. },
```

Display More

Die ID der Option sollte innerhalb des Mods einzigartig sein. Anders gesagt, wird immer nur die zuletzt definierte Option, mit dieser ID, verwendet.
Sofern nicht anders angegeben, stellen bei den optionalen Parametern, die in diesem Bespiel genutzten Werte, auch die Standarwerte dar, falls dieser Parameter fehlt.

Wird das ModUtil Script verwendet, ist es sinnvoll die Optionen erst in einer Variable zu speichern damit diese dann sowohl für den TPFMM als auch für das Script verwendet werden kann:

Code: mod.lua

```
1. local modUtil = require "merk_modutil_1"
2. local settings_def = {
3. option1 = {
4. type = "boolean",
5. name = _("Option1"),
6. },
7. option2 = {
8. type = "number",
9. name = _("Option2"),
10. },
11. }
12. function data()
```

```
13. return {
14. info = {
15. -- ...
16. },
17. settings = settings_def,
18. runFn = function(settings)
19. modUtil.userSettings.create("mod_id", settings_def)
20. end
21. }
22. end
```

Display More

Mit der Zeile `modUtil.userSettings.create("mod_id", settings_def)` wird dem Script mitgeteilt, welche Optionen erwartet werden. Außerdem werden automatisch die settings.lua des Mods ausgelesen (sofern vorhanden) und die dort angegeben Werte für die spätere Verwendung gespeichert. Die "mod_id" wird intern genutzt, um zu unterscheiden, von welchem Mod die Einstellungen stammen. Zwei unterschiedliche Mods sollten also nicht die gleiche ID haben. Meiner Meinung nach bietet sich daher der Ordnername des Mods an.

3 Optionen auslesen

Nachdem das Script die settings.lua ausgelesen hat, kann jederzeit auf die Werte der Optionen zugegriffen werden und zwar in jeder beliebigen Scriptdatei (runFn der mod.lua, .con, .mdl, .msh, etc.). Wenn man direkt nach der Definition darauf zugreifen möchte (also noch in der runFn der mod.lua), empfehle ich folgenden Code:
Code

```
1. local options = modUtil.userSettings.get("mod_id")
```

In allen anderen Dateien ist folgender Code sinnvoller (damit man nicht jedes mal das ModUtil Script einbinden muss):

Code

```
1. if merk_modutil and merk_modutil[1] then
2. local options = merk_modutil[1].userSettings.get("mod_id")
3. -- Einstellungen verarbeiten
4. end
```

"mod_id" bestimmt in beiden Fällen von welchem Mod man die Einstellungen haben möchte. Im Normalfall wird es also die gleiche ID wie bei "create" sein, man kann aber auch die Einstellungen eines anderen Mods auslesen (z.B. wenn der eigene diesen nur ergänzt und daher auch zum Teil die gleichen Optionen nutzt). Der Zugriff auf die einzelnen Optionen funktioniert ebenfalls in beiden Fällen identisch, z.B. mit `options.option1`.