# Editor-Scripte (EM4)

## Editor-Scripte

Analog zu Command- und Mission-Scripte verfügt auch der EMERGENCY 4 Editor über eine Schnittstelle zum Laden von Scripten. Hierbei ist es möglich, im Editor verschiedene Funktionen auszuführen.

Editor-Scripte müssen, um von EMERGENCY erkannt zu werden, im Data-Ordner gespeichert werden. Daher sollte man darauf aufpassen, dass alle Änderungen nur im Editor-Script-Ordner stattfinden! Diese Dokumentation ist möglicherweise unvollständig und / oder fehlerhaft, da sie nicht auf dem offiziellen Script-SDK basiert, sondern durch Reverse Engineering erarbeitet wurde!

## Inhaltsverzeichnis



### Grundstruktur

Die Grundstruktur eines Editorscripts ist ähnlich zu der von Missions- und Commandscripts:

### Quellcode

```
1. object Example : EditorScript
2. {
3. Example() //Konstruktor
4. {
5. }
6. ~Example() //Destruktor
7. {
8. }
9. void OnThink() //Wird wiederholt aufgerufen (einmal pro Frame?)
10. {
11. }
12. ScriptResult OnPlaceObject(float x, float y, float z) //Wird aufgerufen, wenn der Benutzer ein Objekt platziert
13. {
14. }
15. ScriptResult OnObjectPlaced() //Wird aufgerufen, wenn der Benutzer ein Objekt platziert hat (die Maustaste nach dem Platzieren loslässt)
16. {
17. }
18. OnFloorClicked(float x, float y, float z) //Wird aufgerufen, wenn der Benutzer auf den Boden klickt
19. {
20. }
21. };
```

Alles anzeigen

Ein Objekt, welches von der Klasse EditorScript abgeleitet wird und in diesem Fall „Example“ heißt, wobei der Name natürlich frei wählbar ist. Er muss aber nicht eindeutig sein, das Script wird anhand seines Dateipfades identifiziert. Das Beispiel zeigt alle definierbaren Funktionen, die aber natürlich nicht alle definiert werden müssen.

Editor-Scripts müssen als „Data/Scripts/Editor/Kategorie/Name.script“ gespeichert werden, wobei Kategorie und Name frei wählbar sind. Sie werden im Editor unter Scripts?Kategorie?Name aufgerufen.

### Spezifische Funktionen, Klassen und Enums

Grundsätzlich verwenden Editor-Scripts natürlich das selbe Scriptsystem, wie Command- und Missionsscripts und können daher alle Elemente des bekannten SDKs verwenden. Zu beachten ist hierbei, dass jedoch nicht alles im Editor möglich ist, da einige Funktionen (z.B. Die „PushAction“-Funktionen der „GameObject“-Klasse) auf das laufende Spiel angewiesen sind.

#### Editor-Script beenden

Editorscripts werden nicht automatisch beendet, sondern über die Funktion „Process::Kill()“.

Ein Beispiel:

## Quellcode

```
1. object DeleteVehicles : EditorScript
2. {
3. void OnThink()
4. {
5. GameObjectList l(TYPE_VEHICLE);
6. for(int i = 0; i < l.GetNumObjects(); i++)
7. Game::RemoveGameObject(l.GetObject(i));
8. Process::Kill();
9. }
10. };
```

„**Process::Kill()**“ verhindert hier, dass das Script im Hintergrund weiterläuft und beim Setzen eines neuen Fahrzeuges dieses sofort wieder löscht.

**Objekte platzieren**
Es ist ebenfalls möglich, Objekte mit „Game::CreateObject()“, „Game::CreatePerson()“ oder „Game::CreateVehicle()“ zu erstellen!

Um den aktuell ausgewählten (an der Maus „hängenden“) Prototypen abzufragen, kann die Funktion „GetMousePrototype()“ verwendet werden, die einen Pointer unbekannten Typs zurückgibt. Dieser kann in einer Variable vom Typ „void*“ gespeichert werden:

```
void *mousePrototype = GetMousePrototype();
```

Es ist auch möglich, das noch nicht endgültig platzierte Objekt mit „GetMouseObject()“ abzufragen. Die Funktion gibt einen „Actor“ zurück, der auch in ein „GameObject“ konvertiert werden kann:

```
Actor mouseActor = GetMouseObject();
GameObject mouseObject(&GetMouseObject());
```

Um nun ein Objekt zu platzieren wird die Funktion „PlaceObject(void *prototype_, float x, float.y, float.z, float yaw, float pitch, float roll)“ verwendet. So lässt sich zum Beispiel ein Objekt genau in der Kartenmitte platzieren:

```
PlaceObject(mousePrototype, 0, 0, 0, 0, 0, 0); //Variable stammt aus vorherigen
Codebeispielen
PlaceObject(GetMousePrototype(), 0, 0, 0, 0, 0, 0); //Kurzform ohne Variable
```