Symfonia – Anleitung

Das Nachschlagewerk für den neuen Anwender

by totalwarANGEL

Inhaltsverzeichnis

1 Vorwort

Symfonia, oder auch QSB-S, hat es sich zum Ziel gesetzt, das Leben des Mappers zu erleichtern. Viele bekannte und geschätzte Standardfunktionen befinden sich bereits in der QSB und müssen nicht mehr in die eigenen Skripte kopiert werden. Um in den Genuss der Vorzüge dieses Rahmenwerkes (englisch: Framework[1]) zu kommen, muss man sich an einige *wenige* Regeln halten, denn ohne Regeln ist das Leben ein Chaos. Das ist überhaupt nicht kompliziert und macht sogar Spaß! Sobald die Grundregeln beachtet werden, geht Dir das Mappen so leicht von der Hand, wie eine Scheibe Brot zu schmieren.

Es ist vollkommen unnötig seine Zeit mit Gedanken wie „jetzt ist alles anders“ oder „ich bin kein Genie“ zu verschwenden. Bei der Programmierung der QSB wurde Wert darauf gelegt, altbewährte Vorgehensweisen beizubehalten und zu erweitern. Zusätzlich sich einige Dinge hinzugekommen, die unkompliziert eingesetzt werden können um aus den eigenen Maps etwas besonderes zu machen. Dabei sind Kreativität und Individualität keine Grenzen gesetzt! Bald wirst Du in den Genuss der einzigartigen Möglichkeiten kommen, die Dir die QSB anbieten kann.

Doch zuvor stehen einige Erklärungen aus. Niemand erwartet von Dir sofort alles zu verstehen. Und genau das ist das großartige an Symfonia: es gibt nichts zu verstehen! Die gesamte Handhabe ist intuitiv designt. Und wer einmal den Dreh raus hat, wird nie wieder etwas anderes wollen. Aber genug der langen Vorrede: Jetzt werde ich die wichtigsten Dinge erklären, die Du benötigst um Symfonia zu nutzen. Ich werde auf die speziellen Funktionalitäten eingehen, angefangen bei der Installation und den ersten Schritten zur Verwendung bis hin zu einigen speziellen Features.

1.1 Rechtsbelehrung

DIE SIEDLER – Aufstieg eines Königreichs sind eingetragene Marke von Ubisoft. Alle Rechte bezüglich des Spiels liegen bei Ubisoft. Ubisoft kann jedoch keine Fragen zur QSB oder den Programmen beantworten, da dies ein Community Projekt ist.

Obwohl das Programm weder ein Virus noch ein Trojaner ist – und wenn Kasperski und Co das Gegenteil behauten – geschiet die Nutzung auf eigene Gefahr. Für eventuelle Schäden an Geräten oder den Verlust von Arbeit und/oder Freizeit wird keinerlei Haftung übernommen! Die gleichen Regeln gelten für die QSB-S und alle anderen mitgelieferten Programme.

1 Framework: ist ein Programmiergerüst, welches vereinfacht und dem Anwender Vorteile gegenüber der nativen Möglichkeiten einer Umgebung bietet, in der er sich bewegt.

2 Installation

2.1 Siedelwood Programme

Die QSB-S wird als Archiv mit Anleitungen, Beispielen und Programmen ausgeliefert. Um die QSB zu erhalten, musst Du QSBSBuilderInstaller.exe ausführen und das Programm auf deinem PC installieren. Das Programm selbst benötigt die Java Runtime Enviorment Version 8 (JRE) um zu funktionieren. Dies gilt übrigens für alle Siedelwood-Programme.

Beispiele für Runtime Enviorments:

- AdoptOpenJDK (Enthält JRE)
- Open JDK (Enthält JRE)
- Oracle JRE

Wer noch kein Java auf seinem PC installiert hat, kann sich z.B. die AdoptOpenJDK von dieser Seite herunterladen.

https://adoptopenjdk.net

Du kannst auswählen, welche Software herruntergeladen wird. Wähle OpenJDK8 aus. Als Platform Windows x64 JRE aus. Solltest Du noch immer ein 32-Bit-System haben, musst Du stattdessen Windows x86 JRE auswählen.

Folge anschließend den Anweisungen auf der Seite.

2.2 Installation des QSBSBuilder

Führe die `QSBSBuilderInstaller.exe` aus und installiere das Programm in ein beliebiges Verzeichnis. Im Startmenü findest du einen Eintrag „Siedelwood“. Dort befindet sich der QSB-S-Builder und kann gestartet und auch deinstalliert werden. Du kannst Dir auch eine Verknüpfung auf den Desktop anlegen, wenn Du das willst.

2.3 Erster Programmstart

Wird das Programm zum ersten mal gestartet, wird es versuchen die Quelldateien der QSB herunterzuladen. Du benötigst dafür eine Internetverbindung. Es wird automatisch immer die aktuelle Version der QSB heruntergeladen. Nachdem der Download abgeschlossen ist, kannst Du direkt und ohne Neustart loslegen.

Wenn die Struktur der lokal vorhandenen Quelldateien nicht mehr mit denen im Netz übereinstimmt, wird auch automatisch ein Update beim Programmstart ausgeführt und die aktuelle Version aus dem Internet heruntergeladen.

Dieses „Zwangsupdate“ geschieht nur zu dem Zweck die Funktionalität dieser Software und der mit ihr erzeugten QSB zu gewährleisten!

2.4 Updates manuell laden

Eines der Features des Programms ist es, nach Updates zu suchen. Durch ein manuelles Update werden nur die Quelldateien der QSB aktualisiert. Ein solches Versionsupdate wird Dir nicht aufgezwungen. Du musst selbstständig über die Option „Aktualisieren“ im Block „Nach Updates suchen“ ein Update ausführen.

Ist eine Aktualisierung verfügbar, wird Dir eine Seite mit allen Änderungen angezeigt. Hängst Du mehr als eine Version hinterher, bekommst du immer automatisch alle Updates angezeigt. Du kannst dann das Update ausführen oder abbrechen.

Sollte es keine neue Version geben, wirst Du darüber informiert und kehrst zur Hauptseite zurück. Schaue regelmäßig mal nach!

2.5 QSB erstellen und verwenden

Über „Editor öffnen“ im Block „QSB zusammenstellen“ wird die Komponentenauswahl angezeigt. Auf dieser Seite können die einzelnen Bestandteile der QSB ausgewählt werden. Bundles können immer ausgewählt werden. AddOns haben Abhängigkeiten ohne die sie nicht funktionieren. Die Abhängigkeiten werden automatisch mit ausgewählt. Solltest du aber eine Abhängigkeit entfernen, wird auch das AddOn wieder entfernt.

Standardmäßig sind Core, BundleClassicBehavior, BundleQuestGeneration und AddOnQuestDebug ausgewählt und können auch niemals abgewählt werden. Diese Komponenten repräsentieren die minimale Konfiguration, welche in etwa der QSB 3.9 entspricht. Es steht Dir natürlich frei, weitere Komponenten oder sogar alle Komponenten auszuwählen. Um Dir die Entscheidung zu erleichtern, kannst Du die Dokumentation anzeigen.

Die QSB wird im Downloads Ordner gespeichert. Du kannst das Verzeichnis aber frei wählen. Neben der erstellten QSB kannst Du auch die Rohskripte für das globale und lokale Skript mit ausgeben lassen. Diese Skripte werden ebenfalls benötigt, damit die QSB funktioniert! Darum müssen diese Dateien verwendet werden. Die Vorlage mapscript.lua ist für das globale Skript und die Vorlage localmapscript.lua für das lokale Skript gedacht. Diese Dateien müssen in Deine Map importiert werden, damit sie gespielt werden kann! Du kannst die Skripte unverändert verwenden und gleich beginnen Quests im Assistenten zu erstellen. Natürlich kannst Du auch eigene Funktionen hinzufügen.

Um dies zu tun, musst Du den Expertenmodus aktivieren. Im Expertenmodus können die Kartenskripte importiert und exportiert werden. Bitte lasse Dich nicht von der Bezeichnung „Expertenmodus“ oder dem unsinnigen Warnhinweis abschrecken! Klar besteht die Möglichkeit viel falsch zu machen, aber dafür gibt es ja die QSB. Mit ihr reduziert sich das Risiko auf ein Minimum! Sobald die Kartenskripte importiert sind, kann es losgehen!

3 Erste Schritte bei der Verwendung

Die QSB kann wie gewöhnt mit dem Assistenten im Karteneditor verwendet werden. Alle Standardfunktionen stehen Dir zur Verfügung. Die QSB kann natürlich noch mehr als nur Auftragsassistenten. Auf diese Dinge wird später eingegangen.

Für eine Anleitung zum Auftragsassistenten des Editors konsultiere bitte das SEED Handbuch.

3.1 DEBUG aktivieren

Bevor Du in den kreativen Schöpfungsprozess einsteigst, möchtest Du vielleicht den Debug[2] aktivieren, um Fehler schneller entdecken zu können. Im Debug-Mode können auch Cheats verwendet und Kommandos eingegeben werden. Der Debug wird aktiviert mit dem Behavior[3] `Reward_DEBUG`.

Die folgenden Funktionen können alle eigenständig aktiviert oder deaktiviert werden:

Option	**Beschreibung**
Quests zur Laufzeit prüfen	Die Behavior eines Auftrages werden zuerst geprüft, bevor die Ausgeführt werden. Tritt ein Fehler auf, wird der Auftrag gestoppt und eine Fehlermeldung angezeigt.
Questverfolgung	Bei jeder Änderung des Status eines Auftrages wird eine Meldung auf dem Bildschirm angezeigt.
Cheats aktivieren	Aktiviert die altbekannten Cheats des Spiels. Für eine Liste der Cheats siehe die Dokumentation.
Eingabe aktivieren	Aktiviert eine Eingabeaufforderung mit einfachen Befehlen. Die Eingabe wird mit `SHIFT + ^` begonnen und mit `Enter` bestätigt. Für eine Liste der Befehle siehe die Dokumentation.

Auch wenn es immer wieder gern getan wird, um den Spieler in Notlagen helfen zu können, ist der Debug nicht dafür gedacht in der finalen Map aktiv zu sein! Mit dem Debug ist es möglich tiefgreifende Manipulation an der Folge der Quests vorzunehmen. Ebenfalls kann man eigene Lua-Skripte ins Spiel laden.

Der unehrlichen Haut, die ein wenig vom Spiel und von LUA versteht, wird somit Tür und Tor zur Manipulation geöffnet. Dadurch können Probleme entstehen, die im normalen Spielverlauf niemals auftreten würden. Debug bitte vor Veröffentlichung deaktivieren.

2 Debug: Kommt aus der Sprache der Software-Entwickler. Fehler werden aus Bugs (Käfer) bezeichnet. Fehler beseitigen bedeutet also Ungeziefer auszumerzen, das einem das Leben schwer macht. Begebe dich auf Kakerlakenjagt und lösche die Fehler alle aus!

3 Behavior: (eng. „Verhalten“) hier: Steuert, wie sich ein Quest verhält.

3.2 Namenskonvention für Quests

Eine Map ist nichts ohne eine Geschichte. In einem Spiel werden Geschichten durch Quests erzählt. Der Spieler muss Aufträge erledigen um die Handlung voranzutreiben. Folglich musst Du Deine Geschichte in Aufträge gliedern. Man kann jedoch leicht den Überblick verlieren, da es vermutlich eine größere Anzahl geben wird. Deshalb bietet sich die Einteilung in Abschnitte an. Jeder Auftrag bekommt einen Abschnitts-Präfix vor seinen Namen.

```
K1Q1_Langvogt_werden
K1Q2_Verlorene_Schafe_finden
K1Q3_Belohnung_Schafe_gefunden
K1Q4_Wolfsangriff
...
```

Der Editor wird die Aufträge entsprechend geordnet im Assistenten anzeigen. Du solltest Dich nicht nur auf Abschnitts- und Auftragsnummer verlassen! Eine kurze Zusammenfassung was in dem Quest passiert, erhöht die Übersichtlichkeit.

Nicht alle Zeichen sind für einen Namen erlaubt. Auch wenn Du Questnamen auf Deutsch schreibst, bist du beschränkt auf Großbuchstaben (A-Z), Kleinbuchstaben (a-z), Zahlen (0-9) und Unterstrich (_). Umlaute (äöüß usw.), Sonderzeichen (!“§$ usw.), Leerzeichen und nicht druckbare Zeichen (\n\r usw.) sind nicht erlaubt! Solltest Du einen Namen mit nicht erlaubten Zeichen verwenden, wird der Quest nicht erzeugt werden.

3.3 Schnelles Testen

Nach *jeder* Änderung im Skript müsste man eigentlich das Spiel verlassen, den Editor starten und das Skript neu importieren. Das ist auf Dauer anstrengend und kostet viel Zeit! Daher bietet es sich für die Entwicklung an, das globale und lokale Skript auszulagern. Dazu wird ein Hilfsskript benötigt, das anstelle der eigentlichen Skripte in die Map importiert wird.

```
local PathToProject = "C:/MyProjects/MyMap";
if not GUI then
    Script.Load(PathToProject .. "/mapscript.lua");
else
    Script.Load(PathToProject .. "/localmapscript.lua");
end
```

Dieses Skript lädt automatisch die entsprechenden Dateien, da es zuerst prüft, ob es sich in der globalen oder lokalen Environment[4] befindet. Dadurch kannst Du während der Entwicklung viel Zeit sparen. Passe einfach den Pfad `PathToProject` an Deine Bedürfnisse an. Beachte: vor der Veröffentlichung musst Du aber die eigentlichen "richtigen" Skripte in die Map importieren.

Ein Beispiel für einen angepassten Pfad könnte wie folgt aussehen:

```
local PathToProject = "C:/Maps/Scripts/mf02_gutkirschenessenmitmanfred";
```

Solange der Pfad stimmt und alle Dateien vorhanden sind, wird es funktionieren.

4 Environment: (eng. „Umgebung“) Bezeichnet einen für sich abgeschlossenen Bereich.

4 Grundlagen des Skripten

4.1 DEBUG im Skript aktivieren

Genauso, wie der Debug im Auftragsassistent aktiviert werden kann, kannst Du ihn auch über einen Skriptbefehl aktivieren. Dazu genügt ein einfacher Aufruf in der `Mission_FirstMapAction`.

```
API.ActivateDebugMode(true, false, true, true);
```

Die Parameter sind exakt die gleichen wie beim Behavior `Reward_DEBUG`.

4.2 Behavior selber schreiben

Manchmal sind die Behavior, die die QSB bereitstellt, einfach nicht genug!

In diesem Fall wirst Du in die Situation kommen, ein Behavior selbst schreiben zu müssen. Das ist überhaupt nicht schwer, wenn man sich dabei an ein paar Regeln hält.

1. Behavior haben 4 verschiedene Typen: Goal (Ziel), Reprisal (Strafe), Reward (Belohnung), Trigger (Auslöser). Ebenso unterschiedlich wie sie sich verhalten, werden sie auch im Skript geschrieben. Wobei Reprisal und Reward aber identisch aufgebaut sind.
2. Ein Goal kann 3 Zustände haben: unbestimmt, erfüllt und fehlgeschlagen. Diese Zustände werden durch einen Wahrheitswert bestimmt, den das Goal zurückgeben muss. Dabei wird für erfüllt `true` zurückgegeben, für fehlgeschlagen `false` und für unbestimmt nichts.
3. Ein Trigger hat 2 Zustände: ausgelöst und inaktiv. Diese Zustände werden ebenfalls durch Wahrheitswerte repräsentiert und müssen zurückgegeben werden. Um auszulösen muss `true` zurückgegeben werden. Für inaktiv entweder `false` oder einfach nichts.
4. Ein Reprisal oder ein Reward sind simple Callbacks[5] und geben nichts zurück. Alle Statements[6] werden nacheinander ausgeführt ohne Fragen zu stellen.

Wenn man diese Regeln beachtet, kann man ganz einfach eigene Behavior schreiben.

4.2.1 Ein Goal schreiben

Goals sind Ziele, die der Spieler erreichen muss.

Ob ein Ziel erreicht ist, wird durch eine Fallunterscheidung[7] überprüft und durch Rückgabe eines Wahrheitswertes dem Questsystem bekannt gemacht.

```
function HausGebaut()
    if Logic.IsConstructionComplete(HAUS_ID) == 1 then
        return true;
    end
end
```

5 Callback: Funktion, die als Reaktion auf ein Ereignis aufgerufen wird.
6 Statement: Ein Befehl im Code
7 Fallunterscheidung: Es wird, je nach Situation, entsprechend reagiert. Beispiel: Für Fall 1 wird Aktion A ausgeführt, für Fall 2 Aktion B. Fallunterscheidungen sind die wichtigsten Werkzeuge der Programmierung.

4.2.2 Einen Trigger schreiben

Ein Auslöser bestimmt, wenn ein Quest startet.

Auch hier wird wieder eine Fallunterscheidung durchgeführt. Damit ein Quest startet, muss der Trigger `false` zurückgeben.

```
function SollQuestGestartetWerden()
    if Logic.GetCurrentMonth() == 10 and Logic.GetMonthSeconds() > 60 then
        return true;
    end
end
```

4.2.3 Ein Reprisal/Reward schreiben

Ein Reprisal bzw. ein Reward sind Aktionen, die für Fehlschlag bzw. Erfolg ausgeführt werden. Hier müssen i.d.R. keine Fälle unterschieden werden.

```
function SchiffBewegen()
    local x, y, z = Logic.EntityGetPos(GetID("destination"));
    Logic.MoveEntity(GetID("ship"), x, y);
end
```

4.3 Aufträge in das Skript verlagern

Aufträge im Skript zu erzeugen, ist kein Hexenwerk! Du kannst sogar Parameter von Behavior und Eigenschaften des Quests weglassen. Sie werden mit Standardwerten aufgefüllt! Dies geht einher mit einer neuen Schreibweise.

Ein Beispiel für die neue Schreibweise:

```
local QuestName = API.CreateQuest {
    Name        = "VisitCloister",
    Suggestion  = "Ich sollte die Mönchen des Klosters besuchen!",
    Success     = "Guten Tag, werter Herr!",

    Goal_DiscoverPlayer(4),
    Reward_Diplomacy(1, 4, "EstablishedContact"),
    Trigger_OnQuestSuccess("SomeOtherQuest"),
}
```

Für den Namen des Quest gelten die gleichen Regeln wie im Auftragsassistent.

Die Behavior werden als Funktionen geschrieben. Die Parameter, die Du zuvor von oben nach unten über die Maske eingegeben hast, werden jetzt von links nach rechts angegeben.

Die Texte eines Quest können auch zweisprachig angegeben werden. Es wird Deutsch (de) und Englisch (en) unterstützt. Ich zeige es Dir am Beispiel der Startnachricht:

```
Suggestion = {
    de = "Ich sollte die Mönche des Klosters besuchen!",
    en = "I should visit the monks of the monastery!",
},
```

Dies gilt natürlich für alle angezeigten Texte eines Quests.

4.3.1 Angaben für einen Auftrag

Aufträge haben viele mögliche Angaben. Die Eigenschaften eines Auftrags werden als Felder bezeichnet. Eigenschaften sind z.B. der Name des Auftrags. Du kannst die wichtigsten Eigenschaften eines Auftrags der nachfolgenden Tabelle entnehmen.

Eigenschaft	**Beschreibung**
Name	Über den Namen werden die Quests verwaltet
Suggestion	Der Text des Sprechers zu Beginn des Auftrages
Success	Der Text des Sprechers bei erfolgreichem Abschluss
Failure	Der Text des Sprechers im Falle eines Fehlschlages
Description	Die Beschreibung für benutzerdefinierte Auftragstypen
Visible	Erzwingt die Sichtbarkeit, wenn Suggestion nicht angegeben ist.
EndMessage	Erzwingt die Sichtbarkeit der Endnachricht.
Sender	ID des Auftraggebers
Receiver	ID des Auftragnehmers
Time	Zeit bis zum automatischen Fehlschlag/Erfolg

4.4 Interaktive Objekte

Mit den erweiterten interaktiven Objekten kannst Du neue Elemente in Deine Maps bringen. Helden können z.B. versteckte Schalter finden um geheime Gänge zu öffnen. Der große Vorteil ist, dass sich so ziemlich jedes Entity in ein interaktives Objekt umwandeln lässt.

```
API.CreateObject {
    Name                = "trigger1",
    Title               = "Versteckter Schalter",
    ConditionUnfulfilled = "Dir fehlt der benötigte Schlüssel!",
    Condition           = function(_Data)
        return HasFoundKey == true;
    end,
    Callback            = function(_Data)
    end,
};
```

Diese Konfiguration ist ein Beispiel für ein versteckten Schalter, der aber erst aktiviert werden kann, wenn die Variable `HasFoundKey` gesetzt ist. Vorher bekommt der Spieler eine Meldung, dass er einen Schlüssel benötigt.

Natürlich können interaktive Objekte auch Aktivierungskosten und Belohnungen haben.

```
API.CreateObject {
    Name   = "chest1",
    Title  = "Schatztruhe",
    Costs  = {Goods.G_Iron, 1, Goods.G_Gold, 50},
    Reward = {Goods.G_Gems, 300},
};
```

Die Kosten aller interaktiven Objekte werden sofort bezahlt und nicht angeliefert!

4.5 Interaktive Siedler

Nichtspieler-Charaktere (NPC) sind Siedler, die von einem Helden explizit angesprochen werden müssen, damit sie etwas auslösen. Du kannst sie Dir als lebende interaktive Objekte vorstellen. Achte darauf, dass die Position des NPC erreichbar ist, er also *nicht* im Blocking steht. Ein NPC ist dadurch zu erkennen, dass er auf der Spielwelt glitzert.

```
API.NpcCompose {
    Name     = "hakim",
    Callback = function(_Npc, _Hero)
       -- Hier kann was passieren
    end,
}
```

Dies ist ein Beispiel für einen einfachen NPC.

Wichtig ist, dass jeder Siedler nur eine dieser Konfigurationen haben kann. Wenn Du auf den gleichen Siedler einen anderen Aufruf von `API.NpcCompose` machst oder `Goal_NPC` auf ihn anwendest, wird die alte Konfiguration überschrieben. Skriptfuntionen werden dann nicht mehr gestartet und Quests, deren Goal `Goal_NPC` ist, funktionieren nicht mehr!

Ebenfalls muss beachtet werden, dass der Siedler während er als NPC aktiv ist, weder seine Entity-ID noch seine Spielerzuordnung verändert. Sollte dies passieren, verliert der Siedler sein Flag, welches ihn als NPC kennzeichnet!

Die Callback-Funktion des NPC kann eine völlig beliebige Aktion ausführen. Das reicht vom Anzeigen einer Nachricht bis zum Start eines Briefings. Bedenke, dass anders als bei `Goal_NPC`, ein über `API.NpcCompose` erzeugter NPC nicht mit dem Questsystem verbunden ist. Dies muss per Hand nachgeholt werden!

Beispiel für einen Trigger:

```
function HatHakimGesprochen()
    return API.NpcHasSpoken("hakim") == true;
end
```

NPCs haben noch weitere Funktionalitäten. Du kannst einen Helden vorgeben, der den NPC ansprechen muss. Diese Option ist allerdings nur sinnvoll, wenn der Spieler in Deiner Map mehr als einen Helden gleichzeitig steuert. Für die meisten Maps ist diese Angabe daher unnötig.

Wird der NPC nun mit einem Helden angesprochen, dessen Skriptname nicht `marcus` ist, wird das Callback nicht ausgelöst und der Text von `WrongHeroMessage` als Nachricht am linken Rand auf dem Bildschirm ausgegeben.

```
API.NpcCompose {
    Name             = "hakim",
    Hero             = "marcus",
    WrongHeroMessage = "Ich spreche nicht mit dir!",
    Callback         = function(_Npc, _Hero)
       -- Hier kann was passieren
    end,
}
```

4.6 Briefing

Zwar werden Quests und einher gehende Dialoge in Siedler 6 über den Questdialog angezeigt, aber manchmal möchte man etwas mehr. Eine schönere Präsentation der Umstände. Außerdem führen zu viele Nachrichten im Questdialog irgend wann zwangsläufig zum Rhian-over-the-sea-chapell-Bug. Dann ist das Nachrichtenfenster dauerhaft blockiert und kann u.U. nicht mehr genutzt werden.

Oder was machst Du, wenn ein angesprochener Siedler keinen animierten Kopf hat? Indirekte Rede durch den Helden? Das ist alles andere als schön! Zum Glück kann dem Abhilfe geschaffen werden! Die aus Siedler 5 bekannten Briefings stehen Dir mit der QSB-S auch in Siedler 6 zur Verfügung um die Handlung zu erzählen.

4.6.1 Grundlagen von Briefings

Ein Briefing besteht aus einer Abfolge von Seiten, die entweder automatisch weiter blättern oder explizit durch Button-Klick bestätige werden müssen. Jede Seite hat einen Titel, einen Text, eine Position und Kameraeinstellungen.

Kameraeinstellungen können traditionell oder als Richtungsvektor angegeben werden.

```
local Briefing = {
    HideBorderPins = true,    -- Grenzsteine verstecken
    ShowSky = true,           -- Himmel anzeigen
    RestoreGameSpeed = true, -- Spielgeschwindigkeit wiederherstellen
    RestoreCamera = true,     -- Kameraposition wiederherstellen
    SkippingAllowed = true,   -- Seiten überspringen erlauben
    ReturnForbidden = true,   -- Zurückspringen ist deaktiviert
}
Local AP, ASP = API.AddPages(Briefing);
```

Mit diesem Block wird ein Briefing eingeleitet. Mit der Funktion `API.AddPages` werden die Funktionen zur Erzeugung von Seiten erzeugt und an das Briefung gebunden. Ohne diesen Block gibt es auch kein Briefing!

Nachstehend werden die Seiten des Briefings definiert.

```
AP {
    Title        = "Titel der Seite",
    Text         = "Text der Seite",
    Position     = "pos1",
    DialogCamera = true,
    Action       = function(_Data)
    end
}
```

Hier siehst ein einfaches Beispiel für eine Seite im Briefing. Die grundlegenden Angaben sind immer `Titel`, `Text` und `Position`. Mittels `DialodCamera` werden Defaults für `Angle`, `Rotation` und `Zoom` gesetzt. Wenn Du diese Felder separat setzt, überschreiben sie die Defaults. Mit der `Action` kann eine beliebige Aktion ausgeführt werden, wenn die Seite angezeigt wird.

```
ASP("pos1","Titel der Seite","Text der Seite", true, function(_Data) end);
```

Die Funktion ASP bietet eine Vereinfachung, die dem obigen Beispiel von AP entspricht. Das ist vor allem dann Hilfreich, wenn Du viele Dialoge schreiben willst.

Anschließend werden die Methoden `Starting` und `Finished` definiert.

```
Briefing.Starting = function(_Data)
end
Briefing.Finished = function(_Data)
end
```

Die Methode `Starting` wird aufgerufen, bevor die erste Seite des Briefings angezeigt wird.

Die Methode `Finished` wird aufgerufen, nachdem die letzte Seite verlassen wurde.

```
return API.StartBriefing(Briefing);
```

Dies ist die letzte Zeile des Briefings. Hier wird der Befehl zum Start des Briefings gegeben. Besonders wichtig ist das `return` davor. Nur wenn die Funktion, in der das Briefing steht, den Wert zurück gibt, kann das Briefing mit den entsprechenden Behavior an Quests gebunden werden.

Für weitere Optionen für Seiten eines Briefings sieh Dir die Dokumentation von AP an.

5 Das Burglager

In DIE SIEDLER - Aufstieg eines Königreichs ist es nicht vorgesehen endlos viele Rohstoffe und Waren zu hamstern, wie in den Vorgängern der Reihe. Über die Jahre gab es viele Anläufe ein zusätzliches Lager bereitzustellen. Dies wurde geläufig als „Burglager“ bezeichnet. Symfonia bringt ein eigenes Burglager mit.

Das Burglager wird für den angegebenen Spieler initialisiert und kann sofort danach verwendet werden. Diese Funktion muss immer zuerst aufgerufen werden, bevor irgend etwas mit dem Burgtlager gemacht wird.

```
API.CastleStoreCreate(1);
```

Ein Burglager kann natürlich jeder Zeit wieder entfernt werden. Alle Waren im Burglager sind dann jedoch unwiederbringlich verloren.

```
API.CastleStoreDestroy(1);
```

Dem Burglager können auch direkt Waren hinzugefügt oder daraus entfernt werden.

```
API.CastleStoreAddGood(1, Goods.G_Wool, 25);
API.CastleStoreRemoveGood(1, Goods.G_Wool, 25);
```

Die Menge an Waren im Burglager können abgefragt werden.

```
local WoolAmount  = API.CastleStoreGetGoodAmount(1, Goods.G_Wool);
local TotalAmount = API.CastleStoreGetTotalAmount(1);
```

Der voreingestellte Basiswert zur Berechnung des Lagerlimits kann geändert werden.

```
API.CastleStoreSetBaseCapacity(1, 150);
```