SWEndor Scripting Guide – Custom Campaigns and Scenarios

Version: v0.1
Created: 29 December 2019

CONTENTS

INTRODUCTION

SWEndor supports a scripting engine and is capable of playing user-defined custom scenarios.

Organization

There are three sets of files

- Campaign list file – the first file that the SWEndor will read, which determines all the scenarios that will be loaded.
- Campaign file – referenced by the campaign list life, this file contains basic information about the campaign and references to each of its scenarios.
- Scenario file – referenced by the campaign life, this file contains basic information about the scenario and references to the scripts.
- Script files – referenced by the scenario file, each of these files contain scripts that control runtime behavior of the scenario.

CAMPAIGN LIST FILE

Introduced in v0.1

Format: INI File.

Location: <executable_path>\data\scenarios\campaigns.ini

The campaign list file defines all custom scenarios to be loaded in order of appearance.

[Scenarios]

The entries in Scenarios define all scenario files to be loaded. Each new line corresponds to a new scenario file.
The list order determines the order of the custom scenarios in the scenario selection screen.
Custom campaigns are always placed after the built-in campaign.
Each entry is a relative path from this file's location.

Example

```
[Campaigns]
Imperial\campaign.ini
Smuggler\campaign.ini

#The above entries indicate that the following custom campaign files will be loaded in order:
#
# <executable_path>\data\scenarios\Imperial\campaign.ini
# <executable_path>\data\scenarios\Smuggler\campaign.ini
```

CAMPAIGN FILE

Format: INI File.
Location: Defined by the Campaign List file

Campaign files group several playable scenarios under a single campaign.
Each scenario file must be placed in the /Data/Scenarios directory for the game to recognize it.

[General]

Key	Type	Default Value	Description
Name	string		The name of the scenario. This will be displayed in the campaign selection screen.
Description	string		The description of the scenario. This will be displayed in the campaign selection screen.

[Scenarios]

The entries in Scenarios define all scenario files to be loaded. Each new line corresponds to a new scenario file.
The list order determines the order of the custom scenarios in the scenario selection screen.

Each entry is a relative path from this file's location.

Example

```
[Scenarios]
1_1\Imperial_1_1.scen
1_2\Imperial_1_2.scen
13_8\Imperial_13_8.scen

#Suppose that this scenario file is located in <executable_path>\data\scenarios\Imperial\
#The above entries indicate that the following custom scenario files will be loaded in order:
#
# <executable_path>\data\scenarios\Imperial\1_1\Imperial_1_1.scen
# <executable_path>\data\scenarios\Imperial\1_2\Imperial_1_2.scen
# <executable_path>\data\scenarios\Imperial\13_8\Imperial_13_8.scen
```

SCENARIO FILE

Format: INI File.

Location: Defined by the Campaign file

Scenario files link scripts together into a playable scenario.

Each scenario file must be placed in the /Data/Scenarios directory for the game to recognize it.

[General]

Key	Type	Default Value	Description
`Name`	`string`		The name of the scenario. This will be displayed in the mission selection screen.
`PlayerName`	`string`	"Luke"	The name other actors will call the player craft.
`Description`	`string`		The description of the scenario. This will be displayed in the mission selection screen.
`Wings`	`string[]`		The list of craft IDs the player is allowed to choose for this mission.
`Difficulties`	`string[]`		The list of difficulties the player is allowed to choose for this mission. The difficulty adjustment must be implemented by the scenario itself.

[Bindings]

Key	Type	Default Value	Description
`Fn_load`	`string`	"load"	The name of the load function. This function is called at initialisation, after Fn_loadfaction and Fn_loadscene.
`Fn_loadfaction`	`string`	"loadfaction"	The name of the loadfaction function. This function is called at initialisation.
`Fn_loadscene`	`string`	"loadscene"	The name of the loadscene function. This function is called at initialisation, after Fn_loadfaction.
`Fn_makeplayer`	`string`	"makeplayer"	The name of the makeplayer function. Defaults to 'makeplayer'. This function is called every time a player is supposed to respawn, both initially and after each death. The actual respawn logic must be implemented by the scenario itself.
`Fns_gametick`	`string[]`		The list of the tick functions. This function is called every tick of the game.

[Audio]

Key	Type	Default Value	Description
`Win`	`string`		The audio played with the scenario is won. This path is relative to the /assets/music folder, and excludes the *.mp3 extension.

Lose	`string`	The audio played with the scenario is lost. This path is relative to the /assets/music folder, and excludes the *.mp3 extension.

[Scripts]

The entries in Scripts are a list of script files to be included in this scenario. Each new line corresponds to a new file.
Each entry is a relative path from this file's location.
Each file must be in a script file format.

Example

```
[Scripts]
..\..\Common\Spawn.sw
..\..\Common\ActorHealth.sw
Script.sw
Message.sw

#Suppose that this scenario file is located in <executable_path>\data\scenarios\Imperial\13_8
#The above entries indicate that the following script files will be loaded in order:
#
# <executable_path>\data\scenarios\Common\Spawn.sw
# <executable_path>\data\scenarios\Common\ActorHealth.sw
# <executable_path>\data\scenarios\Imperial\13_8\Script.sw
# <executable_path>\data\scenarios\Imperial\13_8\Message.sw
```

Execution order

The game, when loading a custom scenario, will load script files defined in its Scenario file.
It will then execute the following functions in order:

- The script whose name is defined under **`Fn_loadfaction`**. If a script by this name is not found, this step is skipped.
- The script whose name is defined under **`Fn_loadscene`**. If a script by this name is not found, this step is skipped.
- The script whose name is defined under **`Fn_load`**. If a script by this name is not found, this step is skipped.

When attempting to generate a player (at initialization or after player death), it will execute the following:

- The script whose name is defined under **`Fn_makeplayer`**. If a script by this name is not found, this step is skipped.

Every game tick, the game will execute the following:

- The scripts whose names are defined under **`Fns_gametick`**.

SCRIPT FILE

Format: Script File.
Location: Defined by the Scenario file

Script files are used by the game to define programmable logic.
They have a custom syntax that is interpreted by the game engine to generate behaviour. This way you can define your own scenario without needing to modify the game program.

Syntax

Script files take a pseudo-C style language syntax.

A script may look a bit like this:

Example

```
float3 faction_empire_color = { 0, 0.8, 0 };
int tiea1;

loadfaction:
	Faction.Add("Empire", faction_empire_color);

load:
	// spawns the player
	string tiea1_type = GetPlayerActorType();
	float3 tiea1_pos = {100, 0, 10000};
	float3 tiea1_rot = {0, -180, 0};
	tiea1 = Actor.Spawn(tiea1_type, "Alpha-1", "", "", 0, "Empire", tiea1_pos, tiea1_rot);
	Actor.SetProperty(tiea1, " Health.MaxShd", 25);
	Actor.QueueLast(tiea1, "wait", 2.5);
	Actor.AddToSquad(Player.GetActor(), tiea1);
```

Refer to the **Scripting Guide for Writing Scripts** to understand the syntax structure of the script.