SWEET TRANSIT HOW TO MOD

SWEET TRANSIT MODDING GUIDE

Modding in Sweet Transit is mostly done through .define files which are found in the Data folder. Mods are being loaded in a predefined order by updating existing files or adding new ones.

FIRST MOD

Creating an empty mod is quite simple. Only two steps are needed:

- Create a new folder in the Data folder

- Create a Header.txt file inside

Header.txt file is used by the game to understand the mod. It is using .define style of writing.

Header.txt should contain these lines:

name = mod_name; // The mod name. It will be translated using the localization file Mods collection;

mod_version = 1.0; // The mod version

game_version = 0.2.7; // A game version the mod is created for

dont_load_with_newer_version = false; // Is a failsafe in cases you know that this mod will break with never versions

weight = 1000; // Is used for automatic mod sorting. Mods with bigger weight will load later

before = mod_name, another; // The mod will load before specified mods if they are found

after = mod_name, another; // The mod will load after specified mods if they are found

SHARING

There are two ways you can share your mods. First one is simply by sharing the mod folder. Another option is using Steamworks. That can be done by going to the Main Menu and selecting Mods section at the left side of the screen. Hovering your cursor over a mod allows you to upload it. Once your mod is uploaded you will need to change visibility, default is hidden.

LOCALIZATION

Sweet Transit is using keys within the game that are then are looked up in localization files.

Localization files are found in Localization folder which can be found in any mod.

To create a new language simply create a new file in the folder. All files should follow ISO 639-1 standard.

To add translation for your new mod in English follow these steps:

- Create Localization folder in the mod base folder
- Create empty "en" text file
- Open it with any text editor
- Create Mods collection and write your mod keys:

```
Mods // Mods collection
{
  mod_name=Mod Name; // Use the same value as you specified in the Header.txt file
  mod_name_description=Describe what your mod does.;
}
```

To update or change any other word or sentence simply copy the collection and key:

```
Entities
{
  dirt=Not Dirt;
}
```

Collections:

- Fonts - Selects which defined fonts to use for that specific language
- Languages - Translates language ISO 639-1 codes. It is only done in Native language in en file
- Mods - Mostly for translating mods
- Entities - Entity names

- Descriptions - Entity can have a description and it is looked up here

Everything else should be self-explanatory.

DEFINITIONS

Definitions folder houses .define files which are the heart of the modding.

.define files uses custom syntax that is used to define collections and values.

DEFINITIONS VALUES

Value is defined like this:

```
name = value;
```

It needs to have 4 parts, which are the "name", "=", "value", ";".

name cannot have empty spaces.

value itself can be anything that the game expects. For example:

```
icon = Base\Icons\bread.png;
inhabitants_grow_speed = 1f / (60 * 10);
```

Values can span through multiple lines and they need to always end with ";".

Entity expects specific kinds of values with specific names. Most of these values are exposed inside original .define files.

DEFINITIONS COLLECTIONS

Collections are simply collections of values. They can be empty or nested.

Empty collection is defined like this:

```
CollectionName;
```

or

```
CollectionName { }
```

.define file is opened as a collection from the begining.

In a sense .define files are just collections.

The most common name for a collection is Definition. All collections named like this will be loaded as entities.

Entities are just a data structure to explain and load something. That could be a train, unlock rules, song.

Most entities always need at least 3 values:

- type - What sort of entity is this?

- name - Unique name to reference this entity elsewhere

- icon - Path to the icon file

DEFINITIONS EDITING

You can edit or remove existing Definition collections.

To edit any definition you need to create a name.define file in Definitions folder with any name.

For this example we will edit Locomotive ST10:

```
Definition
{
  type = TrainWagon; // Type of the entity we want to edit
  name = locomotive_st10; // The unique name of the definition
  edit = true; // Telling we want to update this entity. By default this is false which means we would override it.

  front_power = 2000; // Editing values are as simple as writing them again
  fuel_consumption =; // Removing value

  BuildCost // Editing collections are the same as values
  {
    coins = 500;
  }
```

```
 Upkeep.Remove; // Removing collections.
 Upkeep // You can add a collection again after removal to start fresh
 {
  coins = 1;
 }
}
```

DEFINITIONS ADVANCED

You can copy existing collections. If a collection is not named Definition it is as simple as:

```
 link(CollectionName);
```

This will copy the whole collection content.

To copy something from Definition use this:

```
 link(Definition, name, name_value, Collection);
```

This function uses 4 parameters:

- Definition - Name of the collection to search for
- name - To which value to compare
- name_value - Which value to search in the compare value
- Collection - Which collection to copy. This can be nested by ":" separator "Graphics:Base"

Some values have multiple values within separated by ",".

All values can have simple arithmetic for example valid ones are:

```
 value = -1 + 2 * 6 + 3; // 14
 value = (16 - 17f) * 2; // -2
 value = (int)(5 / 3); // 1
 value = (int)3.1f / (int)2.8f; // 1
 value = 1e5; // 100000
```

```
value = 9 % 8; // 1
```

To define a loop you can write:

```
value = 1|5; // Will result in 1, 2, 3, 4, 5
value = 0|3, 2|0; // 0, 1, 2, 3, 2, 1, 0;
value = r=0.2.6; // 0, 2, 4, 6;
```

OTHER

Defaults file is holds values for more technical information. In here you can change default sounds, game speed, buffs.

All in game shaders are exposed and can be modified at will. They are located in Other folder, definition that loads them is named Shaders.define.

It is the same with fonts, sounds, music and UI.

LIMITATIONS

Currently Sweet Transit does not officially support any kind of code injection or modification. This means that using official tools you will not be able to add new functionality, only use the existing one.