1. Основы

Добавить новый скрипт можно через программу. Далее открываем файл **index.mml** в созданной папке **files/scripts/[название]** и изменяем его содержимое. В файле **description.txt** находится описание скрипта.

Скрипт исполняется слева-направо сверху-вниз. Все строки в скрипте пишутся в одинарных кавычках ''.

2. Написание скрипты

2.1 file и комментарии

Любой скрипт содержит изменение файлов, оно пишется как `file '[путь_к_файлу]' { [изменения_файла] }`. Путь к файлу пишется относительно папки **[media]** в **initial.pak**. Файл указываем без расширения **.xml**.

Пример:

```
// строчный комментарий, не выполняется
/*
  Многострочный
  комментарий
*/

file 'classes/trucks/ank_mk38' {
  // изменения файла (change / change_add / add).
}
```

Может содержать в себе `change` и `change_add`.

2.2 изменение тегов

2.2.1 change

Используется ТОЛЬКО ДЛЯ ИЗМЕНЕНИЯ тегов: `change '[путь_к_тегу]' { [изменения_тега] }`.

Путь к тегу это *XPath* (можно забить в гугле). В кратце: нам надо изменить тег "TruckData" в теге "Truck", путь будет "Truck/TruckData". Чем-то похоже на пути к файлам.

Пример:

```
change 'Truck/TruckData' {
  // изменения тега
}
```

Для изменения тега в конкретной позиции можно использовать атрибут в XPath: в квадратных скобках указывается индекс по порядку сверху вниз. Пример "Gearbox/Gear[3]" или "SuspensionSet[1]/Suspension[2]".

В стандартном виде изменяется только ПЕРВЫЙ подходящий элемент. Для изменения ВСЕХ подходящих элементов используется `all`.

Пример:

```
change all 'Gearbox/Gear' {
  // изменения тега
}
```

Может содержать в себе change / change_add / add / set / remove.

2.2.2 change_add

Ипользуется ДЛЯ ИЗМЕНЕНИЯ ИЛИ ДОБАВЛЕНИЯ В СЛУЧАЕ ОТСУТСТВИЯ: change_add '[тег]' { [изменения] }. Работает и пишется аналогично change, но не может использоваться all и требуется указать только тег, а не путь к нему. Для уточнения позиции используется all: change_add '[тег]' at [индекс] { [изменения] }

Пример:

```
change_add 'Gear' at 3 {
  // изменения
}
```

Может содержать в себе change / change_add / add / set / remove.

2.2.3 add

Используется ТОЛЬКО ДЛЯ ДОБАВЛЕНИЯ тегов: add '[тег]' { [изначальные_параметры] } Тег добавляется в конец текущего активного тега (об этом ниже).

Пример:

```
add 'Gear' {
  // параметры
}
```

Может содержать в себе **add** / **set** / **remove**.

2.2.4 remove

Ипользуется ДЛЯ УДАЛЕНИЯ ТЕГОВ/АТРИБУТОВ: remove '[путь_к_тегу]'" или "remove [атрибут]. Для удаления тега его путь пишется в скобках, для атрибута пишется нахвание без скобок. Для удаления всех подходящих тегов используется **all**: remove all '[тег]'.

Пример:

```
remove 'Gear[3]' // удаляется третий тег Gear
remove Country   // удаляется атрибут Country у текущего тега
```

2.2.5 set

Используется ДЛЯ ИЗМЕНЕНИЯ ЗНАЧЕНИЯ АТРИБУТА: set BackSteerSpeed = 0.35. После равно указывается значение атрибута. Доступно числовое (5 / 2 / 0.35), строковые ('abc' / 'hello'), булевые (true / false).

Пример:

```
set AngVel = 10
set IsIgnoreIce = true
set UiName = 'MY TRUCK NAME'
```

2.3 ПРИМЕРЫ СКРИПТОВ

Скрипт, добавляющий 5-ю передачу в снегоходную коробку:

```
file 'classes/gearboxes/gearboxes_trucks' {
  change 'GearboxVariants/Gearbox[Name="g_truck_offroad"]' {
```

```
    change_add 'Gear' at 5 {
      set AngVel = 16
      set FuelModifier = 0.9
    }
  }
}
```

change_add в данном случае используется чтобы можно было исполнять скрипт несколько раз без последствий.

Скрипт, добавляющий полный привод *ANK MK 38*:

```
file 'classes/trucks/ank_mk38' {
  change all 'Truck/TruckData/Wheels/Wheel' {
    set Torque = 'default'
  }
}
```

2.4 Продвинутый функционал

2.4.1 Константы

Для объявления константы используется `const`: `const [имя] = [значение]` Объявлять можно в любом блоке скрипта, но область видимости ограничена фигурными скобками `{}`. После объявления можно использовать по имени.

Пример:

```
const wheel_torque = 12
const truck_wheel = 'Truck/TruckData/Wheels/Wheel'

file 'classes/trucks/ank_mk38' {
  change all truck_wheel {
    set Torque = wheel_torque
  }
}
```

Строковые константы могут быть интерполированы или содержать параметры. При интерполяции название указывается между фигурными скобками `{}`. Для подставки параметров их надо объявить в квадратных скобках `[]`. Параметры передаются в скобках через запятую или без неё.

Пример:

```
const name = 'Сергей'
const surname = 'Петров'
const full_name = '{name} {surname}' // Сергей Петров

const dlc_truck = '_dlc/[1]/classes/trucks/[2]'
const dlc_3_truck = dlc_truck('dlc_3', '[1]') // _dlc/dlc_3/classes/trucks/[1]

const boar = dlc_3_truck('boar_45318') // _dlc/dlc_3/classes/trucks/boar_45318

file boar {
  // изменения
```

```
}
```

Пресеты

Для обобщения общих действий используются пресеты. Объявить пресет можно с помощью `preset: preset [имя] { [тело_пресета] }`. Пресет также может принимать параметры. Их названия указываются после имени пресета. Использовать можно в любой части программы. Параметры передаются в круглых скобках.

Пример:

```
const main = 'classes/trucks/[1]'
const dlc_3 = '_dlc/dlc_3/classes/trucks/[1]'

preset unlock_truck(truck) {
  file truck {
    change 'Truck/GameData' {
      set UnlockByRank = 1
      set UnlockByExploration = false
      set Country = ''
    }
  }
}

unlock_truck(main('ank_mk38'))
unlock_truck(main('don_71'))
unlock_truck(main('hummer_h2'))
unlock_truck(dlc_3('boar_45318'))
```