1 Introduction

This document describes the structure of the configuration file for the Particle trail script (suggestions for a new name are always welcome). The script will search for its configuration file inside the *data/config* directory. There it expects a file called *particles.cfg*. This will be a normal text file which values are described in the following document.
Disclaimer: This is only a »quick and dirty« documentation that may contain mistakes (spelling, grammar, content ...).
This document should come with a few other documents which will each describe the structure of one type of configuration definition. Please refer to these documents for actual information on how to define a particle effect.

2 Basic structure

The script parses the configuration file by breaking it up into different chunks that are identified using the *$Type* token. After this token all parsed values will be handed on to the individual chunks parsers that will set-up a particle information from this chunk.

2.1 Input types

Name	Description
None	This options doesn't use a value. If there is one it will most likely be ignored
String	A normal string of characters. Leading and trailing whitespaces are deleted.
Number	A decimal number. There is no difference between integers and float types. Decimal character is the ».« Character.
StringList	A list of normal strings that is separated by commas (,).
NumberList	Same as StringList but the strings are converted to numbers. Entries that can't be converted to a number will be disregarded.
ShipClass	A string specifying a ship class defined in *ships.tbl* or a modular table.
WeaponClass	Same as ShipClass but for weapon classes.
Vector	A NumberList consisting of three entries that build up a vector.
Effect	A string specifying an effect (no extension).
Boolean	A boolean »true«/ »false« expression. The following strings are valid (case-insensitive): **String** / **Boolean Value** **YES** True **NO** False **True** True **False** False
LuaCode	A piece of lua code. Currently every piece of code gets prefixed with *return* if it doesn't start with that sequence.

It is possible that an option can take multiple input types. In that the input types will be separated by a comma (»,«) character.

2.2 Multiline statements

If you want to use multiline statements you have to identify these by prefixing it with »**%[**«. To end this block add »**]%**« at the end of the statement. The ending identifier is automatically

added when the end of the file is reached.

3 Structure Description

Important: All option names are *case-sensitive*.

3.1 Global Fields

These fields will be parsed when the parser is not within a *$Type* chunk.

Option name	Argument type	Description
$Minimum Framerate	**Number**	If the framerate of FreeSpace drops below this value then the script will not create new particles to avoid further FPS drops.
$EndType	**None**	Ends the current *$Type* section. Use it if you want to add global fields after a *$Type*. You may need it when you want to include other files using *$Include*.
$Include	**String**	Includes the specified filename as a additional configuration file. **Warning:** This field does not check for recursion! If this file is a lua file (ends with *.lua*) then this file will be executed. *(a valid filename)*
$Execute	**LuaCode**	Executed the specified piece of lua code at the current position in the parsing process
$If	**LuaCode**	The code will be turned into a lua function that should either return a *boolean* or another *function* which will return a *boolean*. If that boolean is *true* then the following configuration block will be used. If it is false the configuration block will be ed and (if present) the corresponding *$Else* block will be evaluated. The end of this statement must be marked with *$EndIf*.
$Else	**None**	See *$If*
$EndIf	**None**	See *$If*

3.1.1 The *$If* statement

Although the *$If* statement can process any piece of code you may want to use some utility functions that will make reading the configuration file easier. The functions that are shipped with the script are listed below (you can still use your own function):

Function name	Arguments	Usage
inMission	*String*	Takes a mission name (without ».fs2« extension) and returns *true* if the current missions name matches the given String and *false* otherwise. **Example:** *$If: inMission("SM3-04")*

4 External configuration files

If you need it you can specify additional configuration files to keep the configuration as modular as possible. To do so please take a look at the example code below:

```
#Conditional Hooks

$Application: FS2_Open
$On Game Init:
[
if (deathParticleScript) then
    deathParticleScript:addConfigFile("<your config file here>")
    -- add the line above as many times you want for
    -- every configuration file you need
else
    ba.print("No particle trail script found!\\n")
end
]

#End
```

Save this file inside your *data/tables* directory as something that ends with *-sct.tbm* and »<your config file here>« will be used as soon as the script parses its configuration files.

Note: This feature can also be used without a scripting table using the *$Include* Option.