

CHAPTER 5 - PARTICLE EFFECTS AND SCRIPTS

SMOKE EFFECTS

Introduction

Trainz version 1.3 (Service Pack 3) gives you the ability to add customizable smoke, steam, vapor and similar effects to your custom trains and scenery objects. For simplicity, this document will refer to this set of effects as simply smoke effects.

It is assumed the reader is already familiar with creating and exporting models from either 3D Studio Max or GMax.

Method

Smoke effects are added to custom trains and scenery objects in two steps:

1. *Add attachment points to the original model.*
2. *Add smoke tags to the object's config.txt file.*

Adding Attachment Points

Attachment points are added to the original model using 3D Studio Max or GMax wherever a smoke effect is desired. See figures 1 and 2 below to locate the Insert Point tool. After a point is inserted, it must be given a name with a prefix of 'a.' to identify it as an attachment point, e.g. a.smoke, a.steam, a.safety, a.mist, etc. The attachment point should also be rotated so that its Y axis is pointing in the direction that smoke particles will be emitted. (Ensure Axis Tripod is checked to see the point's orientation.) When finished, save and export the model as per normal.



3DS Max insert point



Gmax insert point

ADDING SMOKE TAGS

Smoke blocks are added to an object's config.txt file to describe each smoke effect that will be created on the object. Smoke blocks are named smoke# (where # is a number) and are sequentially numbered starting at 0. See Example 2 for an example.

Smoke blocks have two sections: main and sequence properties. Main properties describe the attributes that do not change based on the mode's key. Sequence properties describe a set of one or more phases/periods in the smoke emission sequence.

A smoke block has the following format:

```
smoke#
{
  mode          time | speed | anim | timeofday
  attachment    <name of attachment point>
  color         <red>, <green>, <blue>,
               <opacity>
  accel        <x>, <y>, <z>
  loop         <n>

  start        <n> [, <n>] ...
  period       <n> [, <n>] ...
  rate         <n> [, <n>] ...
  velocity     <n> [, <n>] ...
  lifetime     <n> [, <n>] ...
  minsize     <n> [, <n>] ...
  maxsize     <n> [, <n>] ...
}
```

Notation:

- # Is a number, starting with 0
- [] Means optional,
- ... Indicates a variable number of parameters,
- | Means or.

Breakdown:

<name of attachment point>

Is the name of an attachment point in the model. eg a.smoke, a.steam, a.chimney etc

<red>, <green>, <blue>

Are numbers from 0 to 255 describing the intensity of that color component.

<opacity>

Is a number from 0 to 255 describing the effect's initial opacity / transparency.

<x>, <y>, <z>

Are vector components pointing in the direction of the sum of all forces affecting this smoke effect. Essentially, <z> describes gravity, and <x>, <y> describe the force of wind.

<n>

Is a decimal number.

MAIN PROPERTIES:

mode

Describes the mode or type of this smoke effect. This affects how start and period are interpreted. Default is time. In all modes, period can be set to -1 (default) to imply the phase is active until the next phase begins.

If set to time, start is a set of time values in seconds after the creation of this effect's parent object when this phase of the effect will start. period is the duration of time this effect will remain active. Scenery objects currently only support time mode.

If set to speed, start is a speed in meters per second (m/s) and period is not used. (Note: 1 m/s = 3.6 km/hr.) All other sequence attributes (rate, velocity, lifetime, minsize, maxsize) are interpolated so there are smooth transitions between phases. See smoke3 in Example 2 for an example.

If set to anim, start is a value from 0.0 to 1.0 which describes the start time into the object's animation cycle. period is a value from 0.0 to 1.0 that describes the duration over which the effect is active. start + period must not exceed 1.0.

If set to timeofday, start is a value from 0.0 to 1.0 which describes the time of day when this effect will start. Values range as follow:

0 - midnight, 0.25 - 6am, 0.5 - midday, 0.75 6pm, 1.0 - midnight.

color

The color of the smoke effect. eg '150,150,150,255' for dark smoke; '255,255,255,150' for steam; '150,150,255,255' for water. Default is '255,255,255,255'.

accel

Acceleration. A vector pointing in the direction of the sum of all forces affecting this smoke effect. Essentially, <z> describes gravity, and <x>, <y> describe the force of wind. Default is 0,0,0.

loop

Time in seconds to loop the smoke sequence. Only valid if mode is set to time.

In general, it is better to use a low emission rate with large particles (ie min/max size) than using a high emission rate with small particles to reduce the impact on frame rate. Smoke effects can be quite stunning but are best used in moderation.

Try experimenting with the different values to get a feel of how they affect the smoke effects. Many different types of effects other than smoke are possible with only a little imagination, e.g. waterfalls, mist, toxic green clouds, fire by using a few effects at the same position to simulate the smoke and flames etc.

SEQUENCE PROPERTIES:

The following properties can be set to a single value or a set of values for multiple phases of the smoke effect. Please note that phases must not overlap as only one phase can be active at any one time. If a property has a set of values, it must be the same length as **start**. If a single value is given then it will be used for all phases of the effect. See Example 1 for an example of using multiple phases.

start, period

See mode.

rate

The rate of emission in particles per second for modes time, speed, and timeofday, or the number of particles to emit over the animation period for anim mode. Default is 4.

velocity

The initial speed of emitted smoke particles. Default is 1.

lifetime

Time in seconds that smoke particles exist for. Default is 3.

minsize

Start size of smoke particles. Default is 0.

maxsize

End size of smoke particles. Default is 3.

EXAMPLE 1

- SMOKE FROM A FACTORY'S CHIMNEY

Using a model of a factory with a chimney, an attachment point called 'a.smoke' is placed at the top of the chimney with it's Y axis pointing up. The factory is then exported as an indexed mesh (.im file type) to the Trainz\world\custom\scenery\factory folder and the model's art assets are copied to the same location. The following config.txt file will cause smoke to come out of the factory's chimney between 6am and midday and 3pm and 6pm. Please note the given KUID is invalid and should not be used in your own custom context.

Config.txt

```
kuid <KUID:###:#####>
region Custom
kind scenery
type Industrial
light 1

smoke0
{
  attachment a.smoke
  mode timeofday
  color 150,150,150,250
  accel 1,0.3,0

  start 0.25, 0.5
  period 0.25, 0.125
  rate 8
  velocity 3
  lifetime 5
  minsize 0.5
  maxsize 2
}
```

EXAMPLE 2

- STEAM TRAIN

An animated steam train model that requires four smoke points may be set up as follow:

- Dark smoke from the main chimney stack that is dependant on the trains velocity (a.smoke, Y pointing up),
- A constant steam trail from a small safety pipe on top (a.steam.safety, Y pointing up),
- 2 steam trails on each side of the train that alternately expel steam keyed to the animation of the trains wheels (a.steam.l, a.steam.r, Y pointing outwards).

The model is exported as a progressive mesh (.pm file type) to 'Trainz\world\custom\trains\steam_train\steam_train_body' folder and the model's art assets are copied to the same location. Please see the custom content creation guide for more information on creating your own custom trains. The following config.txt file in the parent folder will generate the desired smoke effects. Please note the given KUID is also invalid and should not be used in your own context.

For example purposes, the settings of an F7 train have been used.

Config.txt

```
kuid <KUID:###:#####>
kind traincar
bogey 0
engine 1
name Steam Train
mass 100000

enginespec <KUID:###:#####>
enginesound <KUID:###:#####>
hornsound <KUID:###:#####>
interior <KUID:###:#####>

smoke0
{
    attachment a.steam.l
    mode anim
    color 255,255,255,150

    start 0
```

```
    period 0.4
    rate 2
    velocity 1
    lifetime 2
    minsize 0.05
    maxsize 1
}

smoke1
{
    attachment a.steam.r
    mode anim
    color 255,255,255,150

    start 0.5
    period 0.4
    rate 2
    velocity 1
    lifetime 2
    minsize 0.05
    maxsize 1
}

smoke2
{
    attachment a.steam.safety
    mode time
    color 255,255,255,150

    rate 2
    velocity 1
    lifetime 2
    minsize 0.05
    maxsize 1
}

smoke3
{
    attachment a.smoke0
    mode speed
    color 100,100,100,200

    start 0,10,20,30
    rate 3,5,7,9
    velocity 3,4,5,5
    lifetime 4,3,2.5,2
    minsize 0.3
    maxsize 2
}
```

SOUND SCRIPTS

Soundscripts give ambient or directional sounds to objects. They cannot be used on track, bridge or spline objects. Wav files should be located within the same directory as the config.txt file. Examples as follows:

MOJUNCTION

Config.txt

```
kuid <KUID:####:#####>
kind mojunction
region Australia
trackside 2
light 1
mode0 lever1
mode1 lever2
soundscript
{
    toggle{
        trigger toggle
        distance 5, 100
        nostartdelay 1
        repeat-delay 1
        sound
        {
            points.wav
        }
    }
}
etc.
```

PEOPLE CROWD

Config.txt :

```
kind scenery
region Australia
kuid <KUID:####:#####>
type People

soundscript
{
    daysingle {
        repeat-delay 0
        distance 3,150
        sound
        {
            crowd_1.wav
        }
    }
}
etc.
```

MAP

Config.txt

```
kind map
kuid <KUID:####:#####>
soundscript
{
    morning {
        ambient 1
        value-range 1, 0.1
        volume 0.3
        sound {
            ctry_day_1.wav
        }
    }
    night {
        ambient 1
        value-range 0, 0.9
        volume 0.3
        sound {
            night_loop.wav
        }
    }
}
username Britain
workingscale 0
workingunits 0
water <KUID:-1:8009>
region Britain
etc.
```

THUNDERBOX

Config.txt

```
kuid <KUID:####:#####>
region Australia
light 1
kind scenery
type Residential

soundscript
{
    dayloop {
        repeat-delay 15,50
        distance 5, 50
        sound
        {
            strain_1.wav
        }
    }
}
etc.
```

Breakdown of Soundscripts:**repeat-delay**

1 or 2 numbers (min, max, in sec) time to delay between the end of the sound playing, and playing it again randomised between (min .. max)

default min = 0 default max = min

attachment

attachment point on the object to attach the sound to.

default: origin of parent object
(not used for ambient sound)

distance

2 numbers (meters)

1st number = the distance at which the sound is played at 100%

2nd number = the cut-off distance.
doesn't affect the volume of the sound
default: 50m, 150m

sound

list of .wav files to play (randomly picked)

volume

gain of the sound
default 1.0 = 100%

ambient

0 or 1, default 0

Ambient sounds have no 3d "position" and may be stereo non-ambient (positional) sounds are positioned on the object and must be mono

value-range

2 numbers, currently used only for day/night sound effects.

midnight is 0.5, midday = 0.0 or 1.0

Where the numbers are not the same, this sets the start and end times for the sound to play.

default 0,0 (off)

trigger

Currently used only for levers. the sound doesn't play until the trigger message happens

nostartdelay

0 or 1, default 0

If not set, the sound will have a short delay before playing, this stops flanging.
(flanging = really nasty sound caused when several copies of the same sound are played at once)

dayloop, daysingle, morning, night, toggle

These have no function in Trainz and have only been put in for user reference.

Note: Single word only. Do not use a space.

TRAINZSCRIPT TUTORIAL

Introduction

Welcome to the first TrainzScript tutorial.

TrainzScript is the scripting language developed for Auran Trainz. This document will teach you how to create a very simple scenario - it does not aim to teach you how to program, or teach you programming concepts. TrainzScript may be used from Version 1 Service Pack 3 onward to create scenario content.

If you do not understand programming concepts, you may need to read further tutorials before trying to create Scenarios for Trainz. We will be releasing a user-friendly interface for compiling powerful scripts in a later version.

Please take the time to follow the steps of this document from start to finish. The tutorial is presented in an informal (non text book) manner. More help can be found by visiting the Scenarios forum at <http://www.auran.com/trainz/forum/default.htm>

Where do I find TrainzScript

TrainzScript is a scripting language used to drive the scenarios. Each scenario will have one or more TrainzScript files (.gs) located in its directory in the **World\Custom\Scenarios** folder. The supporting TrainzScript files are found in the **\Scripts** folder. The script files in this folder give you all of the supported functions required to control Trainz. Over time, you will learn most of the functions provided in these files. The TrainzScript compiler (gsc.exe) is located in the **\Bin** folder.

Go to the bin folder in your dos prompt, and run the compiler with the -d flag as follows.

```
gsc -d > reference.txt
```

This will copy the TrainzScript documentation to the file reference.txt. Consult this document as a reference manual for the TrainzScript language.

Creating A Scenario

Lets create our very first scenario. This will teach you how to use the TrainzScript compiler and guide you through the process required to make a simple scenario that just loads a map.

1. Launch Trainz, and create a new map in Surveyor. Lay a small track loop, and place a track mark named "START" somewhere on your track. Make sure you name the track mark using UPPERCASE, as all named object in TrainzScript are case-sensitive. Place a lever somewhere on the track so we have something for the camera to focus on. Save your map and call it "Tutorial1".
2. Before exiting out of Surveyor, select the "Export Scenario TSO" from the Trainz Main Menu, and type in "Tutorial1".
3. Quit out of Trainz go to your **Trainz\World** folder. Locate the config.txt file in the folder **World\Custom\Maps\Tutorial1**.
4. Open the config.txt file of your Tutorial1map. It should look something like the following.

```
kuid <KUID:-2:2023211879>
kind map
username Tutorial1
workingscale 0
workingunits 0
water <KUID:-1:110015>
region Australia
```

Note the KUID for your new layout. In this case, it is -2:2023211879.

5. Now go to the **World\Custom\Scenarios\Tutorial1** folder and open the config.txt file of your Tutorial1 Scenario. It should look something like the following.

```
kind activity
username Tutorial1
scriptlibrary Tutorial1
scriptclass MyTutorial1
kuid <KUID:-2:2023211880>
kuid-table {
tutorial1 <KUID:-2:2023211879>
}
description "Tutorial1"
```

A few notes on this. The kuid-table is a name-KUID translation table used by the scenario. All objects loaded by the scenario must be entered in this table. The scenario will reference the KUID by name, which is case-sensitive. For example, when the script loads the map, it will reference it by the name "Tutorial1", which will be looked up in the kuid-table, and found as KUID -2:2023211879. Trains and rolling stock etc are referenced in the same manner. You will also notice that the scenario has its own unique KUID. The description text is displayed in the scenario selection screen.

6. Next, we will begin editing the TrainzScript file. Open the Tutorial1.gs file with your desired programming editor. This file is created from the template.gst in the **scripts** folder when you export the TSO. The syntax of the following script is explained in the compiler documentation, but if you have a read through it, it is pretty self-explanatory. The file looks like this.

```
include "trainz.gs"

//
// class MyTutorial1
// brief This is the scenario class. Modify this
// class with
// your own gameplay.
//
```

```
game class MyTutorial1 isclass Scenario
{
    Train myConsist;
    bool scenarioDone = false;

    //
    // Load will be called by Trainz to load the
    // scenario map, and when the user presses Ctrl-L
    // param data is the save game data if loading a
    // saved game.
    //
    bool Load(string data)
    {
        if(data and data.size())
        {
            Interface.Load(data);
        }

        // load the map
        if(!World.LoadMap(World.FindKUID("Tutorial1")))
        {
            Interface.Log("Error loading scenario map");
            return false;
        }

        return true;
    }

    //
    // Save will be called by Trainz when the user
    // presses Ctrl-S.
    // return the save game string, such that load
    // will be able to restore the save game
    // from the last save check point.
    //
    string Save()
    {
        return Interface.Save();
    }

    //
    // TrainDerailed will be called by Trainz when a
    // train derails
    //
    void TrainDerailed(int trainId)
    {
        if(!scenarioDone)
        {
            World.EndScenario(10);
            scenarioDone = true;
        }
    }
}
```

```

//
// TrainCollided will be called by Trainz when a
train collides
//
void TrainCollided(int trainId)
{
    if(!scenarioDone)
    {
        World.EndScenario(10);
        scenarioDone = true;
    }
}

//
// TrainSpeedingFine() is called by Trainz every
second your trains speed exceeds the floating limit
//
void TrainSpeedingFine()
{
    //Interface.AdjustScore(-10);
}

//
// TrainBadCouple() is called by Trainz when
vehicles couple greater than 8KPH.
//
void TrainBadCouple(int vehicleId)
{
    //Interface.AdjustScore(-200);
}

//
//
// main thread
// brief main is executed automatically after
Load() is called. edit
// main to contain your scenarios gameplay.
//
//

thread void main(void)
{
    // Start the monitor thread to monitor speeding,
derailing etc.
    Monitor();

    //
    // create consist specs
    //

    //
    // create consists

```

```

//
//
// gameplay
//
scenarioDone = true;
}
};

```

7. The final step is to compile the scenario. Copy the ***makescript.bat*** file from ***Scripts\Docs*** folder into your ***World\Custom\Scenarios\Tutorial1*** folder. Run this batch file and make sure no errors are reported (consult the forum for assistance). Notice that this batch file uses the TrainzScript compiler (gsc) to create the Tutorial1.gsl file.
8. You are now ready to launch your very first scenario. The scenario should load your Tutorial1 map. You will notice that the camera is focused on the junction we placed on the map. Had we not placed that junction, the map would not be visible, as Trainz has nowhere to focus the camera.

You have now successfully created your very first scenario. Study the Highland Valley scenarios and the script .gs files for further TrainzScript information. Scripting questions may also be asked on the forums.

Good luck, and we hope you will enjoy your scripting efforts