Animated Traffic Scenery Tutorial
(by Nikola Jovanovic)

While I was searching the internet about this subject I did find fair amount of info related to making dynamic scenery, but all the things I found were just fragments on how to do it and didn't really explain the whole process. After struggling for some time to make sense of this I've finally managed to understand how it's done.

What this tutorial will teach you? – It will tech you how to tweak exported gmax models and make them into dynamic scenery for use in FS2004 in a form of road traffic and moving airport support vehicles.

What this tutorial will not teach you? - It won't teach you to make interactive AI traffic nor any kind of conditional animation. This is beyond the scope of this tutorial and to be honest I have no knowledge on this subject what so ever.

This tutorial involves SCASM and BGL coding and tweaking and it is not recommended for beginners. If you haven't coded before you might have some difficulties following through and I strongly suggest you first get involved with code tweaking before attempting to go any further. I am not a programmer and I don't know a lot about coding so you may find a lot of things that I don't know how to explain and you need to take them as they are. I'm not saying that my method of dynamic scenery creation is the best way to do it, there are probably more efficient techniques, but one thing is certain-it works.

SOFTWARE NEEDED

To be able to complete this tutorial you'll need the following:

- Gmax with FS2002 SDK
- Notepad++ http://notepad-plus.sourceforge.net/ (we need this one to do the coding)
- SCASM 2.95 (this is to compile the final model)
- BGLAnalyze version 3.1 (we use this for decompiling)
- FSRegen version 0.31b
- FSDyn! Version 1.0.6
- Flight Data Recorder version 9.0.0.0 – fltrec90 (used to record the flight path)
- Imagetool (no need to explain what's this for)

All of the software mentioned above can be obtained from the internet and are all free-just search the web.

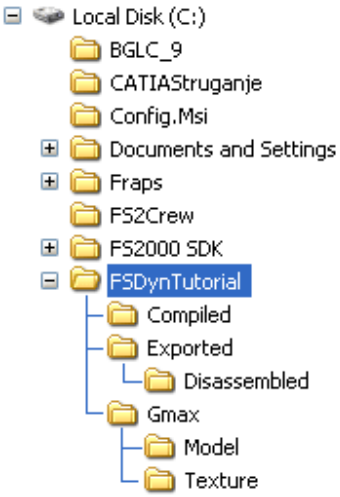
INTRO

Like I told you, I am no expert on this and I'll try to give my best in explaining the concept of dynamic scenery making. As Peter explained in his Wiki article (search for "Dynamic Object Creation" here on fsdeveloper.org) there are two files which make dynamic scenery. First file is the file that contains all the data about the object and represents the object itself (we'll call it the library or "lib" file). The second file holds the information about how will that object behave (move, show…) and is basically telling the Flight Simulator what to do with it (we'll refer to this one as "dynfile"). So now when we understand the concept we can move one to setting up workspace.

SETUP FILES

If you have installed all the programs needed then you're almost ready. Open the FSDyn! and go to **Setup->Install->FLTRec/FSDyn! Set.** A dialogue appears asking you to point to the FS2002 main folder. All dough it asks for the FS2002 guide it to your FS2004 main folder (example C:\Program Files\Microsoft Games\Flight Simulator 9) and press **Install.** It should give you a message that FLTRec version 8.0.1.0 has been installed. You can not use this version in FS2004 so what we need to do is copy the FLTRec 9.0.0.0 over the one that was installed by the FSDyn! (just copy the fltrec90.dll to the modules folder).

Now we'll setup out workspace. Make a folder called *C:\FSDynTutorial*. This will be our main folder in which all the work will be done. Inside this one make the following folder structure:



Now copy the "gspns.bmp" and "gsp.bmp" files from the assets.rar to the Gmax\Texture folder and we're ready to go.

MODELING

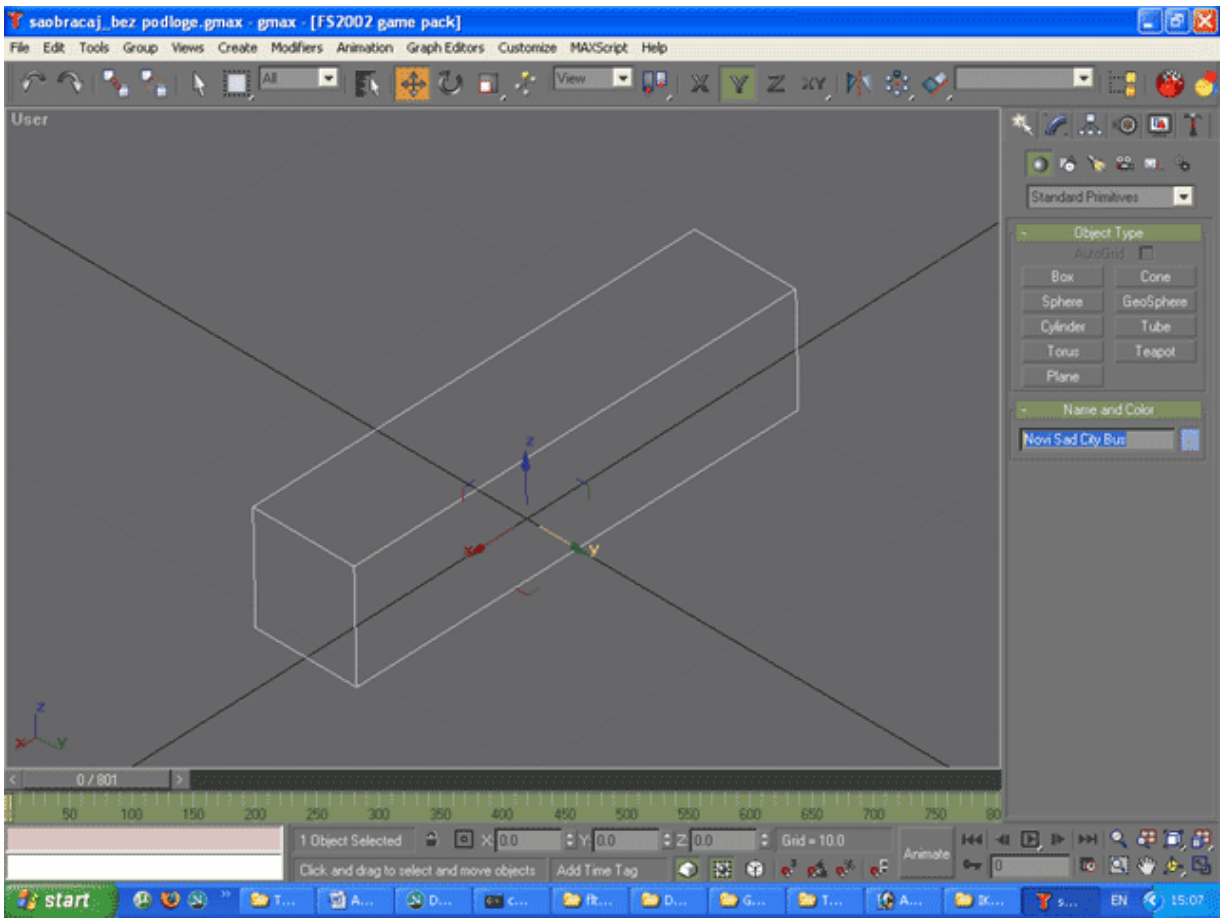
Open Gmax (FS2002) and make a box with the following dimensions:

-Length 4.8 meters

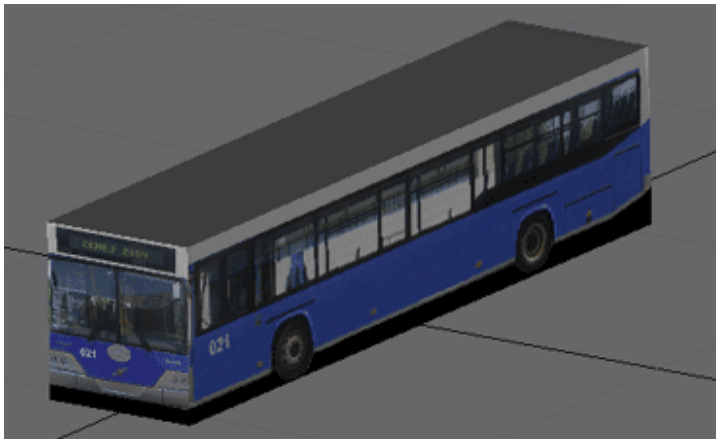
-Width 21 meters

-Height 5 meters

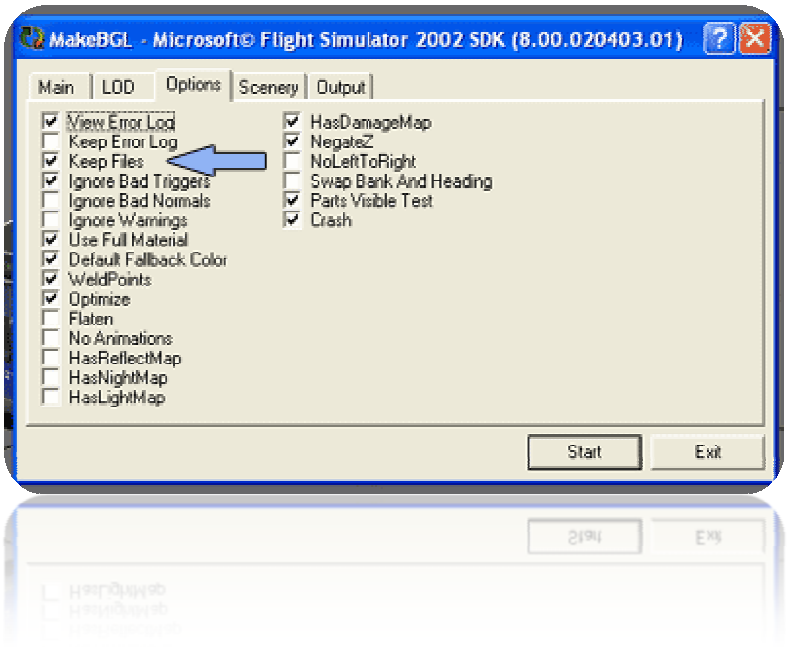
Make sure that the box has position 0,0,0 and that it's rotated along the X axis:



Change the name of the box to "Novi Sad City Bus". Now make a new material and apply the "gspns.bmp" texture and unwrap the texture until you get something like this:



You can save the gmax file in the *C:\FSDynTutorial\Gmax\Model* folder. Now export this new object to *C:\FSDynTutorial\Exported* and call it "NSCityBus.bgl" (make sure that the "Keep Files" option is checked-it's needed to get .asm files) Do the same for the other bus but this time use the "gsp.bmp" and export the model as "CityBus.bgl".



DYNAMIC LIBRARY MAKING

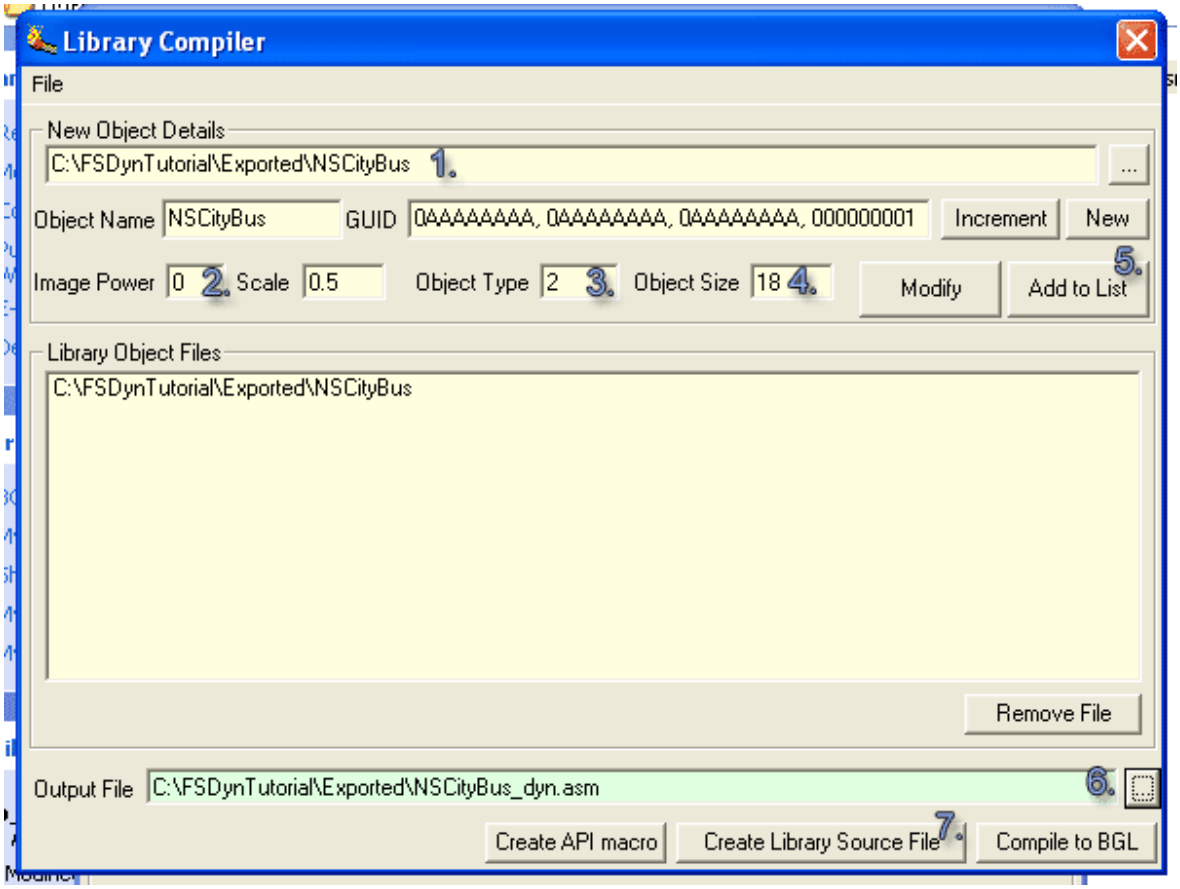
This file is a scenery object but in order to be able to make it into dynamic scenery we need to make it into dynamic object (remember the "lib" file?). Before doing that open the "NSCityBus_0.asm" and "CityBus_0.asm" with Notepad++ (change the Language to Assembler if not already selected-this helps see the code more clearly) and change the following line:

TEXTURE_DEF TEXTURE_AIRCRAFT , <255, 255, 255, 255>, 16.282492, …
with
TEXTURE_DEF TEXTURE_BUILDING , <255, 255, 255, 255>, 16.282492, …

Note you can use Ctrl+H and "Replace in all in all open documents" to make your life a little easier.

We did this step to make sure that the texture will not disappear after a sertain distance. Now compile the two files using BGLC.exe

We are ready now to make them into dynamic library objects. Open FSRegen and go to **Library Compiler** and select the appropriate file and input the values to match the screen below (the steps are numbered):



Repeat the procedure for the “CityBus.asm” but this time after step one press the **Increment** button next to the **GUID** input field.
After completing this you should now have two new files “NSCityBys_dyn.asm” and “CityBus_dyn.asm”. They are not dynamic libraries just yet.

Open the two new "_dyn" files with Notepad++ and modify the lines:

```
LIBRARY_0_HE label word
LIBRARY_0_START label word
**** - insert code here
   include C:\FSDynTutorial\Exported\NSCityBus_0.asm
LIBRARY_0_END label word
   dd 0,0,0,0
```

Code:

```
; Dynamic Code Start
SETWRD 18h, 0 ;LAT ?

SETWRD 20h, 0 ;LON ?

SETWRD 22h, 0 ;ALT ?

SHADOW_VPOSITION 0
SUPER_SCALE Draw_Setup0, 10000, 14, 15 ;V1, V2, SCALE
SHADOW_VICALL Draw_Routine0, 18h
Draw_Setup0 label word
VPOSITION Return_SetUp0, 0, 0, 0
SUPER_SCALE Return_SetUp0, 10000, 14, 15 ; V1, V2, SCALE
VINSTANCE_CALL Draw_Routine0, 18h
SETWRD 2Ch, 18 ; SIZE
Return_SetUp0 label word
BGL_RETURN
Draw_Routine0 label word
; Dynamic Code End
```

Save the files and compile them with BGLC.exe and you should have "NSCityBus_dyn.bgl" and "CityBus_dyn.bgl".

Run the BGLAnalyzer and open the first file ("NSCityBus_dyn.bgl") go to **Disassemble->BGL** and save the file so you get *C:\FSDynTutorial\Exported\Disassembled\NSCityBus_dyn.sca* . Now open this file with Notepad++ and in the **Language** menu select **Assembler**.

Change the code as shown:

```
ObjID( AAAAAAAA AAAAAAAA AAAAAAAA 00000001 )
LibObj(
        PWR   0
        SIZE  18
        SCALE 0.500
        TYPE  2
        NAME   "NSCityBus"    )
        SetVar( 0018 0 )
        ;SetVar( 0020 0 ) these two lines are not needed
        ;SetVar( 0022 0 )
        ;ShadowPosInd( 0000 ) this line controls the shadow
        SuperScale( :L000080 10000 14 15 )
        ShadowCallVI( :L0000A4 0018 )
:L000080
        RefPoint( ofs :L0000A2 0 0 0000 )
        SuperScale( :L0000A2 10000 14 15 )
        PBHCall( :L0000A4 0018 )
        SetVar( 002C 18 )
:L0000A2
        Return
:L0000A4
        BGLVersion( 0800 )
        TextureList( 0
                1 FF 255 255 255 0 15.47295
                GSPNS.BMP ; texture 0
        )
        MaterialList( 0
                        ; material 0
                1.000 1.000 1.000 1.000 ; diffuse color
                0.392 0.392 0.392 1.000 ; ambient color
                0.000 0.000 0.000 1.000 ; specular color
                0.000 0.000 0.000 1.000 ; emissive color
                0.000 ; specular power
        )



        SetMaterial( 0 0 )
```

```
Transform_Mat ( a          ;These lines make the object move and rotate otherwise
0 -3 0                     ;the object is "hovering" couple of meters above ground
0 -3 -90                   ;and is rotated in the wrong direction
)                          ;translation X=0, Z=-3, Y=0; rotation X=0, Y=-3, Z=-90
DrawTriList( 0             ;to get this values you need to use "trial and error" method
        5  14  10 ;  0     ;just play around with the values until you get it right
        6   9  18 ;  1
       13  11  16 ;  2
       16  19  13 ;  3
       10   1   5 ;  4
       18  15   6 ;  5
        0   2   7 ;  6
        7   4   0 ;  7
        3  12  17 ;  8
       17   8   3 ;  9
)
TransformEnd               ;Closes the transform label
EndVersion
Return
EndObj
```

Note some of the code is not included here for the purpose of shortening the text. For full code example check the assets.rar. Do the same for the "CityBus_dyn.bgl", but for this one use:

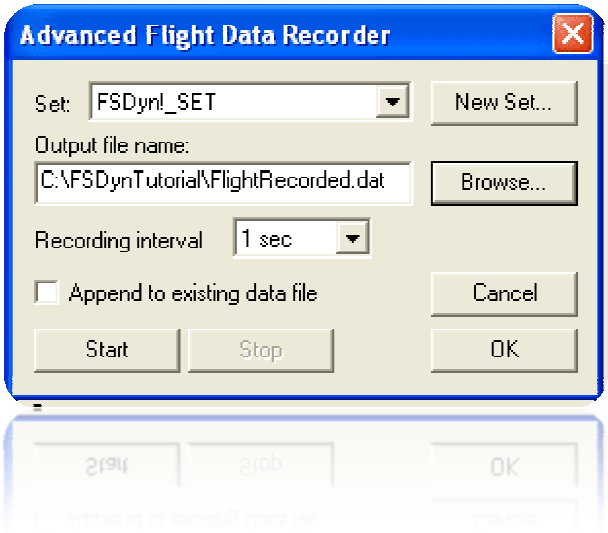
```
Transform_Mat ( a
-20 -3  0
0 3 90
)
```

This one has different values because it is going from the opposite direction to the first one, so it has lateral offset to it, otherwise it would end up crashing into the first object since both of them will use the same route (path) and we don't want to make them run into each other, do we ☺. It is also rotated 180 degrees compared to the first one – otherwise it would end up going back side first.

Once done compile both of the .sca files using SCASM and if everything goes well you'll get two ready to use dynamic scenery "lib" files.
Put them in *C:\FSDynTutorial\Compiled*

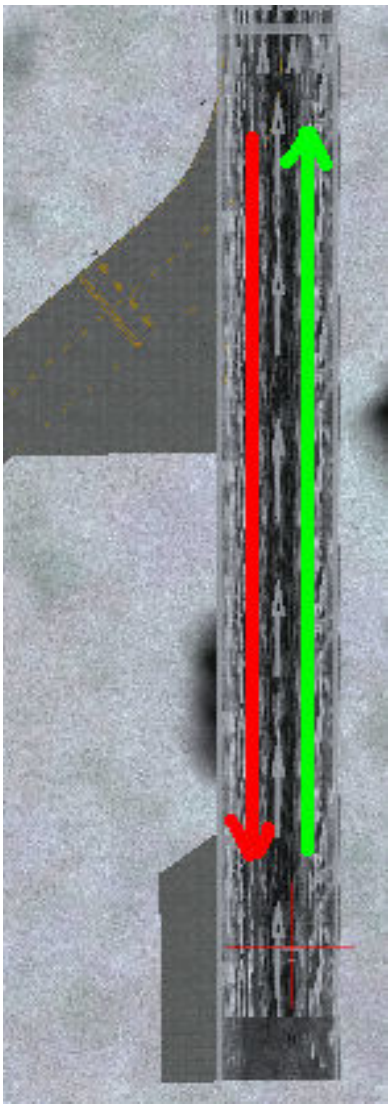
RECORDING THE PATH

As I mentioned earlier the “dynfile” controls our objects and among everything else it also sets the path that this object will follow (hopefully). To record our path (you don’t need to record a path if you’re a freak and you like to torture yourself by coding your eyes out) we’ll use the FLTRec that we had setup before. Open the sim and start the flight at LYBE Belgrade Airport. Press alt and in the top bar of the sim window go to **Recorder->Settings…** and make sure it matches the photo:



After this we are ready to record our path.

Position your aircraft to be in a position similar to this (right side of rwy 30):



Now you can press “Y” to start slewing. Go to **Recorder** and start recording. Move the aircraft in a straight line as shown on the photo in green. After completing the movement, stop the recorder and that’s it you’ve got yourself a recorded flight path. This will represent one lane of the traffic. The red one will be made through coding rather then recording a new flight path.

DECOMPILING THE PATH

Now that we have the path what to do with it? FSDyn! is a great tool for making dynamic scenery. It's easy, fast and gets the job done but there's a catch. It's only good if you want to use it to put default dynamic objects, which is not the case here. We are going to use FSDyn! to make us a template from which we are going make our own "dynfile".

Open the FSDyn! and in the first step go to **Add,** choose the Cessna and click **OK.** In the second step select the Cessna and click **Assign…** and then select the flight path we have recorded earlier. In the cycle frame select **Start Again**. Finally in the third step input the file name ("Traffic") and choose our working directory and then press **Compile**. Open then traffic.bgl with BGLAnalyzer decompile it to traffic_dyn.sca. Open this using Notepad++ and don't forget to choose Assembler Language.

Delete all the lines that are commented including the **Mif…. MifEnd.**
At the beginning of the file add **Set(Area15mx 32)** just above the **Header** label.
Next delete Pattern(:L000045 0FE0B 4 2)
This line calls the Cessna. In its place put:
CallDLibObj(:L000045 4 0 AAAAAAAA AAAAAAAA AAAAAAAA 00000001) 'NSCityBus
This line calls our "NSCityBus_dyn.bgl" file. Let's hang around this part for a sec to analyze it. The "AAAAAAAA AAAAAAAA AAAAAAAA 00000001" is our object's ID so the sim knows what to load, similar to a cars license plate. Then the numbers "4 0" mean that the object will show, regardless of the Dynamic Scenery density setting in the game. Last but most important is ":L000045". This references which label will be used to control the motion. If you go down two rows you can see that it appears again (the first time it was called) and defines what will happen to the object:

```
:L000045
        SetPos(  N44:49:01.70 E020:17:44.56 -58.04)
        Heading( 29 )
        ACS( 1 )
        Call( :L000066 )
        Jump( :L000045 )
```

SetPos and Heading are self explanatory. ACS has something to do with the collision but I'm not completely sure what it does. Now we come to the Call(:L000066) command. Again something is called by the simulator. This time it's the label that contains the path and all the other object's parameters like the heading, pitch and bank. This really is our recorded flight path broken down into 1 second intervals and it ends with a **Return** command meaning "go back to when I was called", and it was called in the :L000045 label. When it returns it comes across a

final command in this label which says Jump(:L000045). The computer jumps back to the beginning of the :L000045 label and the process starts over again (this is because we used **Start Again** command in FSDyn!). I hope you follow me so far and you didn't give up on dynamic scenery. This is not difficult, you just need to get use to the code. OK so we called the first (blue) bus and made it move around. How are we going to put the other one traveling in the opposite direction?

Go back up in the file to the line:
CallDLibObj(:L000045 4 0 AAAAAAAA AAAAAAAA AAAAAAAA 00000001) 'NSCityBus
and let's call the other bus, so add the following line below the first one:
CallDLibObj(:L000055 4 0 AAAAAAAB AAAAAAAA AAAAAAAA 00000001) 'CityBus

You can see that it's almost the same as the previous one, but not quite. Notice that the object ID has changed "AAAAAAAB AAAAAAAA AAAAAAAA 00000001" (makes sense since we want the other bus to appear) and also the label called is now :L000055 (the label name can be anything like :BUS2 or similar, but I decided not to complicate things with renaming the labels). If we were to compile this file now, we would get an error 'cause in the second object we called the label :L000055, but we didn't define it. So below the first label, insert the second one that responds to the object no. 2 –the CityBus:

```
:L000055
        SetPos(  N44:48:44.11 E020:19:23.72 -238.04)
        Heading( 302 )
        ACS( 1 )
        Call( :L000033 )
        Jump( :L000055 )
```

SetPosition has to be changed now since the staring position of this bus is basically the last position of the first one. This makes sense if you're making a highway traffic, so when one vehicle starts from one side of the road the other one starts moving from the opposite side and it's staring position is the first one's ending position (remember we only use one recorded path to make the vehicles go in the opposite directions)

If we were to call now the same label like for the first object (NSCityBus), we would just end up having two overlapping vehicles following the same path. If you want two objects to follow the same route then you need to separate them with a **Hold ()** command, like so:

```
:L000055
        SetPos(  N44:49:01.47 E020:17:44.33 -238.04)
        Heading( 302 )
        ACS( 1 )
        Hold ( 10 )
        Call( :L000066 )
        Jump( :L000055 )
```

This means that the sim will wait 10 seconds before calling this object making a 10 second gap between them and so on.
Back to our example, since we want our object to go on a second path we need to call a different label. In this case we'll name this label :L000033 . Just add this label bellow the Return command form the previous one and you get:

```
:L000066            ;the label which controls the first (blue) bus – one direction
        ACS( 1 )
        Gear( up )
        StartTimer
        MoveToPos1(  …)
         MoveToPos1(  …)
         ….
        Return
:L000033            ;the label which controls the second (blue/white) bus – opposite direction
        ACS( 1 )
        Gear( up )
        StartTimer
        MoveToPos1(  …)
         MoveToPos1(  …)
         ….
        Return
```

Now comes the tough part☺. You need to put all of the **MoveToPos1** lines of the :L000033 label in the reverse order. Meaning that the last line of this label (the one that is before **Return**) goes up the stack to become the first one (the one that is after **StartTimer**) and so on until all of them are reversed so you get a reversed path.

One final thing left to do here and is to add the line at the very end of this file:
Set (Logfile 1)
It will make SCASM generate a log error file which will tell us what we've done wrong. That's it. Now compile the "traffic.sca" with SCASM and you should get a fresh new "traffic.bgl" file.

Copy now "NSCityBus_dyn.BGL", "CityBus_dyn.BGL" and "traffic.bgl" to the active scenery folder, and also don't forget to put the two bmp files in texture folder. Start the simulator and go to LYBE rwy 30 and you should see the two buses.

That's all. I hope this tutorial was helpful and clear enough. Just to say once more, I picked up most of the coding from the internet, all dough the last part is my work.

Thank you for reading and happy scenery making!

January 2009. Nikola Jovanovic