BusyObjects Manual v1.0

2003/03/19

Preface

BusyObjects is a technology that exploits a new way of animating objects in static FS2k+ sceneries. It allows you to play animations at varying speeds, begin (and end) animations at any time and have complex path definitions for the animated object. Since the object is rendered at different positions for (nearly) every frame, the object transformation appears very smooth.

This manual explains the installation of the *BusyObjects Compiler* and the format of the compiler parameter.

Installation

Unpacking the archive

To install the *BusyObjects* framework you simple un-zip the archive **`BusyObjects-1.0.zip`** into a directory of your choice (`<INSTDIR>`). The package will create a new sub-folder `BOC\`; just make sure you have activated the “Use folder names” option when unpacking the archive.

After installing the package, you will see the following files and folders:

```
<INSTDIR>\BOC\              // Base directory
    BOC.EXE                 // compiler executable
    BOC.ico                 // Windows icon
    Licence.txt             // please read this first
    GMAX\
        BusyObjects.ms      // gmax export script
        BO_ImportSCA.ms     // gmax import script
    DOCS\
        Manual-1.0.pdf      // this file
        Tutorial-1.0.pdf    // tutorial
    LIBRARY\
        Y-Complex.API       // complex animation example
        Y-Simple.API        // tutorial animation
    PROJECTS\
        Readme.txt
        EXAMPLE\            // tutorial example
            Example.bod     // compiler parameter
            Example.gani    // gmax animation data
            Example.API     // compiled API
            Example.gmax    // gmax model used
            Example.tpl     // export template
            Example.sca     // import SCASM file
```

Setting environment variables

To access the compiler from the command line without specifying the complete path every time, I also recommend you add the `<INST>\BOC\` directory to the PATH environment variable.

Later versions of the framework will also use a new environment variable called `BOCDIR`. This variable should hold the name of the top-level installation directory of the framework (`<INST>\BOC\`).

In a shell you can use the commands:

```
set BOCDIR=C:\FS2002\BOC
set PATH=%PATH%;%BOCDIR%
```

But you better make these settings permanently; If you are unsure how to do that – please consult the documentation for the OS version you use.

Running the compiler

The compiler is a generic “translator” between input data (the (raw) animation data) and BGL animations that can playback in MSFS.

The only format of input data currently handled is the format that is produced by the BusyObjects gmax script and the only output format is API macro creation. I will extent the compiler to support other input and output formats as needed.

To start the *BusyObjects* Compiler, enter the following command in a shell:

```
BOC parameter.file
```

where *parameter.file* specifies a file containing the parameters for the compiler run.

Usually you start the compiler in the project directory where parameter, import and export files are located. If you don’t add the compiler path (<INSTDIR>\BOC) to the environment variable PATH, you have to specify the absolute path to the *BOC* executable.

Format of parameter files

A parameter file is a plain ASCII file that contains definitions needed for a compiler run. All lines starting with '#' are treated as comments and are not further analysed.

There are three sections with their own set of parameters; all parameters are mandatory. Incomplete parameter files are treated invalid and cause the compiler to stop.

The sequence of definitions can vary; if a variable is set a second time in the same file, the last definition is used.

Import section

The import section defines the source and format of the raw animation data that is used to compile the animation:

`ImportFormat=`**`gmax`**

The **`ImportFormat`** variable is used to define the format for the raw animation data. Until now only **`gmax`** is a valid import format.

`ImportFile=`**`Object1.gani`**

The **`ImportFile`** variable specifies the file that contains the raw animation data. If the file is not located in the directory where you start the compiler, you might have to specify an absolute path for the file here. If you specify a relative path, it is always relative to the directory where the compiler is started.

Animation section

The Animation section defines parameters that influence the way the animation will be compiled and later rendered in MSFS:

`Invariants=`**`ZPB`**

The **`Invariants`** setting lists all degrees-of-freedom that are not animated and should be treated constant. If you move a truck around (over flattened ground), the Z-position of the vehicle never changes – it is invariant. Any of the six degrees-of-freedom that can be animated for a single object can be flagged invariant: **X**-pos, **Y**-pos, **Z**-pos, **P**itch, **B**ank and **H**eading (The beta version can't handle that yet. This only leads to more data being processed than necessary – next version will do).

`Scale=`**`0.01`**

The **`Scale`** parameter defines the scale used in the animation. Think of this value as the resolution of movement. A larger scale is less accurate, so it is best to choose the smallest scale possible for the animation. On the other hand, if the animation spans a larger area, the scale might be limited to a certain value. I recommend using a scale setting of 0.01; this results in a spatial resolution of 1 cm. This should be accurate enough for any animation in the MSFS environment.

It also makes modelling the animated object easier if you have a fixed scale; otherwise you will have to remodel the animated object for every path it moves.

If you want to calculate the optimum scale, use the following procedure: Take the longest span in any dimension (X, Y, Z) in meters and divide that value by 65536. The result is the smallest scale possible for the animation.

If you don't want to calculate this optimal scale yourself, you can specify a scale of –1.0 and the compiler will calculate the best scale for you. Using scale settings below 0.01 might be problematic with "Airport for Windows".

For more info on the scale and object modelling problematic can be found in the tutorial.

`ModelScale=`**`0.5`**

The **`ModelScale`** parameter defines the scale used for the model (object drawing code). This scale can be different from the animation scale (and certainly is in case if the animation is auto-scaled). This value is needed to glue the different scales automatically in the compiler.

`Type=`**`1`**

The **`Type`** parameter specifies what kind of animation is to be generated. Possible values are:

0 for cyclic animations (animations that run forever),
1 for idle/active animations (animations that start and end on a trigger) and
2 for multi-stage animations (hierarchical animations)

Currently only type **1** animations can be produced by *BOC*.

`Duration=`**`450`**

The **`Duration`** parameter specifies the desired duration of the animation. Even the flexible time code generation of the *BusyObjects* framework is not able to handle animation of varying length over a full time code interval.

There are eleven durations (in seconds) in BOC that span a full time code interval: 3641, 1820, 910, 455, 228, 114, 57, 28, 14, 7, 4 – that is ranging from 4 seconds to over one hour. If you can choose the duration freely, try to use one of the specified values or values slightly smaller (but never larger!) then the listed values. This yields best results during playback.

Export section

The export section defines the target and format of the compiled animation:

`ExportFormat=`**`API`**

The **`ExportFormat`** variable is used to define the format of the compiler output. Until now only **`API`** is a valid export format.

`ExportFile=`**`Object1.API`**

The **`ExportFile`** variable specifies the file that will be used to save the compiler output. If you want the file in another directory as where you start the compiler, you might have to specify an absolute path for the file here. If you specify a relative path, it is always relative to the directory where the compiler is started.

`ExportTemplate=`**`Object1.tpl`**

The **`ExportTemplate`** variable specifies the file that is used as a template for the macro header, object rendering code and the state machine.