REO - File Structure

This document was written in order to understand the structure and definitions of .reo files. Everything in this document is to be understood as “open for discussion” because all information were gathered by working with the model files and the editors. Some aspects of the model definitions are still unclear and need further investigation.

Unclear Parts:
BSphere / BBox hierarchy
The last single number row of each bbox definition block
fl 2S flag for faces

- BuXXe - 10.August.2016 -

Sample REO file:

```
# Riot Engine Object
# Exported by Riot Engine Level Editor

version 2.2
name arock's soul crystal
created by BuXXe on 07.08.2016

Lighting 32777

materials 1
0 texture Texture319.bmp

transform
1 0 0 0
0 1 0 0
0 0 1 0

vertices 32
0 -0.0036635 -0.041877 0.0037003
1 -0.0051794 0.041881 3.83e-007
2 0.0051723 -0.041877 3.8402e-007
3 0.0095602 -0.032051 3.839e-007
4 -0.0067662 -0.032051 0.006837

faces 32

polygon 0
vt 3:9 16 23
ma 0
tu 0.1259 0.25 0.3741
tv 0 0 0

polygon 1
vt 4:3 10 9 2
ma 0
tu 1 1.126 1.126 1
tv 0.1173 0.1173 0 0
```

```
fl 2S

polygon 2
vt 4:7 14 10 3
ma 0
tu 0.585106 0.329787 0.329787 0.585106
tv 0.8827 0.8827 0.121449 0.121449

bspheres 1

bsphere 0 : 0 0
-0.0028226 -6.55651e-006 0.00121637
0.0426698

bboxes 62

bbox 0 : 61 0
-1 -0.35416 0.2933
3.00055 0 0
0 3.58702 0
0 0 3.7293
1

bbox 1 : 5 50
-0.5813 0.30566 2.89728
0.95814 0 0
0 0 -0.16835
0 0.22616 0
1
```

version {Version Number Indicator (ex.: 2.2)}
This number indicates a version number. The modeler only accepts models with verson 2.0 - 2.2. It seems that the version number is related to the version of the .reo definition.

name {Model Name (ex.: arock's soul crystal)}
The name for the given model.

created by {Username (ex.: BuXXe)} on {Date Format DD.MM.YYYY (ex.: 07.08.2016)}
The author which created this model / exported it from the database using the level editor. The date gives the date on which it was exported.

Lighting {Special Lighting identifier (ex.: 32777)}
This number identifies the lighting method used to render this model. The Riot engine supports 3 different shading types (None, Flat, Smooth) with 4 different effects (Shiny (Specular Highlights), Blend Lighting with Landscape, Additive Blending, Environment Mapped). Not all effects are supported in all three shading types. The following table shows the possible combinations:

Model Lighting	None	Flat	Smooth
Blend Lighting with Landscape	Not supported	Not supported	Supported
Shiny (Specular Highlights)	Not supported	Supported	Supported
Additive Blending	Supported	Supported	Supported
Environment Mapped	Supported	Supported	Supported

Each combination results in a unique numeric identifier which is computed by giving each option a binary representation. The following list shows the binary representation for each option.

	Binary	Bits	Decimal
None	0	1-bit	0
Flat	1000000000000001	16-bit,1-bit	32769
Smooth	10	2-bit	2
Blend Lighting with Landscape	100	3-bit	4
Shiny (Specular Highlights)	1000	4-bit	8
Additive Blending	10000	5-bit	16
Environment Mapped	100000	6-bit	32

As an example for a calculated identifier we will have a look at:

Flat shading (32769) with the effects Shiny (8), Additive Blending (16) and Environment Mapped (32).
The binary representation would be: 1000000000111001 (Decimal: 32825)

If you just want to calculate the identifier, you can do so by summing up the decimal values. If you want to find out the lighting for a given identifier, you need to take a look at the binary representation.

The identifier 50 results in the binary 1 1 0 0 1 0 which is a Smooth shading (2) with Additive Blending (16) and Environment Mapped (32).

materials {material count (ex.: 1)}
The number after the materials keyword indicates the amount of materials which will be defined right under it. Each definition starts with an id, starting from 0, and incrementing for each entry. After the id, we have the keyword “texture” which seems to serve as a data type. Separated by a space, the concrete filename including the path to the texture is given. If the texture resides in the same directory as the .reo file, only the filename is used.
An example definition would look like: 0 texture Texture319.bmp
If the model is opened in the modeler without the necessary textures being present and is then saved again, the texture definition may be overwritten by an “empty” texture definition. These look like: 0 Texture(0)

transform
This keyword indicates some kind of transformation matrix. The position, scaling and rotation of an object can be represented by a matrix. In this context, this transform matrix could define a new origin for the model, meaning that all vertex definitions are based upon these alterations. The default transformation matrix applies no scaling, rotation or translation to the defined vertices. The matrix is written row by row right after the transform keyword and is 3 rows long and 4 columns wide.
Default transform matrix:

```
1 0 0 0
0 1 0 0
0 0 1 0
```

vertices {vertex count (ex.: 32)}
This keyword indicates the vertex definition section. The keyword is followed by a number giving the count of vertices defined under it. Like in the materials definition section, all entries for a vertex start with its id, starting from 0 and incrementing for each item. Separated by spaces, the coordinates for the current vertex are given in X Y Z order. An example definition would be:

```
0 -0.0051794 0.041881 3.83e-007
```

It could be possible that the .reo files support vertex definitions with 4 entries, which would include the W component, but there is no evidence yet.

faces {face count (ex.: 32)}
This section gives the definitions for the faces of the model. The keyword is followed by a number giving the count of faces defined under it. It is important that, unlike the other sections above, the faces keyword has an empty row after it and then the block with the face definition starts.
Each face definition block starts with the keyword polygon followed by the current face id, starting from 0 and incrementing for each face definition.
The following line starts with the keyword vt and, separated by a space, the amount of vertices involved in this face. After a colon (:) we have a list of vertex ids corresponding to

the vertices which are used by this face. It is important that the colon has no spaces next to it.
A valid vt definition looks like this:

```
vt 3:9 16 23
```

The next line starts with the keyword ma and defines the material this face uses. After a space, the id for the material which shall be used, needs to be set. For example:

```
ma 0
```

In the following 2 lines, each vertex will be given its U and V texture coordinate. Starting with the keyword tu or tv followed by a space separated list of coordinates, each vertex is connected to its UV coordinate. The order of the coordinates corresponds to the order given in the vt row.

```
tu 0.1259 0.25 0.3741
tv 0 0 0
```

There is another optional entry with the keyword fl. In the model “InnerTemple03.reo” There are several face definitions with the line “fl 2S”. It seems that this definition indicates a double sided face. This means, that the renderer shall interpret this particular face as double sided, so the face normal points in both directions.

bspheres {bsphere count (ex.: 1)}
This section gives the definitions for the bspheres of the model. The keyword is followed by a number giving the count of bspheres defined under it. It is important that, unlike the other sections above, the bspheres keyword has an empty row after it and then the block with the bsphere definition starts.
Bspheres are bounding spheres which represent an abstract estimation of the model and are used by the collision detection. They may be hierarchically organized, meaning that there may be one big sphere containing other smaller spheres.

A bsphere entry starts with the keyword bsphere followed by a space and the id of the current sphere, starting from 0 and incrementing for each sphere.
Separated by a space and a colon (:) the area for hierarchy information begins. There are two numbers separated by a space which give information about the hierarchical order of the defined bspheres.
The function of these numbers if not completely clear but investigations of bbox hierarchies have shown a possible pattern.
The first number is the id of the child which shall be subordinated to the current bsphere.
The second number is the id of the next neighbor sphere, on the same level as the current bsphere.
The id 0 seems to be a special case, as spheres with no further neighbors or children will have 0 0 as entries.
In the bboxes chapter we will have a look at an example for a hierarchy.

The following two lines set the position and the radius of the current bsphere. Each coordinate of the position is separated by a space.
Example:

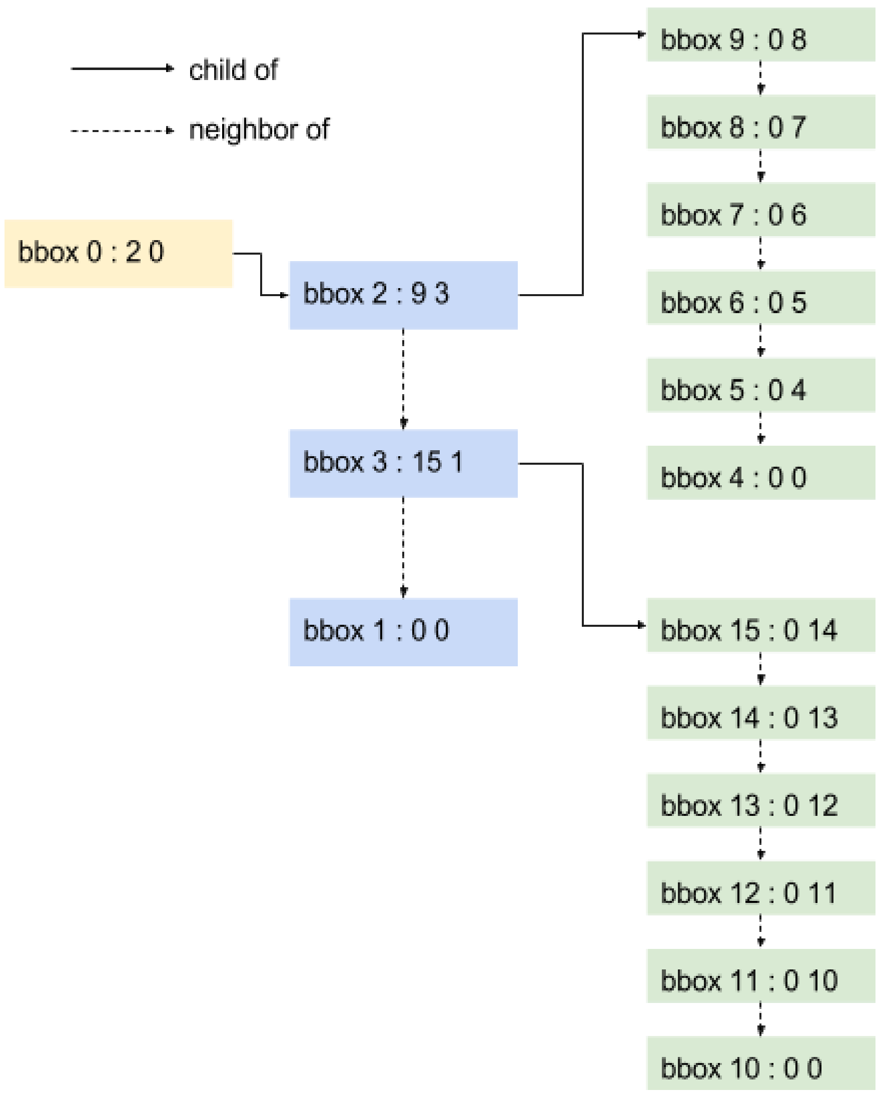
```
-0.0028226 -6.55651e-006 0.00121637
0.0426698
```

bboxes {bbox count (ex.: 16)}
This section gives the definitions for the bboxes of the model. The keyword is followed by a number giving the count of bboxes defined under it. It is important that, unlike the other sections above, the bboxes keyword has an empty row after it and then the block with the bbox definition starts.
Bboxes are bounding boxes which represent an abstract estimation of the model and are used by the collision detection. They may be hierarchically organized, meaning that there may be one big box containing other smaller boxes.

A bbox entry starts with the keyword bbox followed by a space and the id of the current box, starting from 0 and incrementing for each box.
Separated by a space and a colon (:) the area for hierarchy information begins. There are two numbers separated by a space which give information about the hierarchical order of the defined bboxes.
The function of these numbers if not completely clear but investigations of bbox hierarchies have shown a possible pattern.
The first number is the id of the child which shall be subordinated to the current bbox. The second number is the id of the next neighbor box, on the same level as the current bbox.
The id 0 seems to be a special case, as boxes with no further neighbors or children will have 0 0 as entries.

Lets have a look at a complex example of bbox hierarchies:

bbox 0 : 2 0
bbox 1 : 0 0
bbox 2 : 9 3
bbox 3 : 15 1
bbox 4 : 0 0
bbox 5 : 0 4
bbox 6 : 0 5
bbox 7 : 0 6
bbox 8 : 0 7
bbox 9 : 0 8
bbox 10 : 0 0
bbox 11 : 0 10
bbox 12 : 0 11
bbox 13 : 0 12
bbox 14 : 0 13
bbox 15 : 0 14



The following rows define a transformation matrix which represents the bbox. The translation, rotation, scale as well as height, width and length are encoded in this matrix. There seems to be a bug in the modeler which results in a correctly rotated displayed bbox but the parameters for rotation are set to 0. If you rotate the box further, the error is still not corrected which means that the rotation is applied on top of the current rotation.
An example transform matrix entry for a bbox looks like this:

-2.3726 -0.00033228 -1
3.3726 0 0
0 2.65033 0
0 0 3

At the end of the bbox block there is a number which has been observed to change between 0 and 1. Other values may be possible but were not yet observed. The function of this number is completely unclear.