Caligari trueSpace2 file format

Document version 2.0

This document contains details on the geometry and material information stored in Caligari object (*.cob) and Scene (*.scn) files. It should provide enough information for you to be able read and write such files. Most of the information presented is the same for both 1.0 and 2.0 files. Parts that are new or changed for 2.0 have been marked ***New for trueSpace2.***

Caligari files are chunked files. They consist of standard header followed by 1 or more data chunks. The last chunk in the file should always be the special `'END '` chunk. Each chunk consists of a chunk header followed by chunk dependent data. Chunks may not be embedded in other chunks. Each chunk contains an ID and a Parent ID. This allows chunks to be "owned" by other chunks.

The file comes in two flavors, ASCII and binary. These formats encode the same information and have the same general layout. The ASCII files can be edited in a text editor while the binary files are faster for programs to load and save and are smaller.

There is also a provision for little-endian or big-endian formats. Currently all Caligari files are little-endian so you can safely ignore this for the moment.

Caligari Header

The header consists of 32 bytes as follows:

Size	Type	Value	Description
9 bytes	char	`"Caligari "`	Identifies Caligari file
6 bytes	char	`"V00.01"`	Version number of file. This document describes version 00.01 files
1 byte	char	`"A"` for ASCII or `"B"` for binary	Specifies whether file is in ASCII format or binary format
2 bytes	char	`"LH"` for little endian or `"HL"` for big endian	Intel based machines are little endian and currently all Caligari files are too.
13 bytes	char	`"  "`	Leave as blank spaces
1 byte	char	`"\n"` Newline character	Newline

When attempting to decide on the format of a file you are trying to read you should check if the first 15 bytes match the first two fields in the header. This should definitively identify the file as a Caligari file. You will also need to know if the file is ASCII or binary and read the appropiate format

Chunk Headers

Each chunk consists of a header followed by chunk dependent data. The chunk header consists of six fields.

The chunk type tells what type the chunk is and what data it contains. Several chunk types are listed later in this document. You should use this field to choose the appropiate routine to read this chunk or if you do not recognize the chunk type you should skip this chunk using the size field described below.

The major and minor version numbers tell you which version of the chunk type this is. If you get a version number other than the ones you know how to read you should most likely skip this chunk.

The chunk ID and the Parent ID fields let chunks be owned by other chunks. Each chunk has a unique chunk ID. You should keep track of these IDs as you read them in. If the Parent ID is zero then the chunk has no parent, otherwise the Parent ID is the chunk ID of a chunk located earlier in the file and this chunk should be considered to be owned by that previous chunk. An example of this is a file created from a cube which has been painted with two materials. The geometry of the cube will be saved in a `'PolH'` chunk, and following it will be two material `'Mat1'` chunks. Each of the material chunks will have their Parent ID fields set to the chunk ID of the `'PolH'` chunk to show that the materials are owned by the cube.

There is also a data size field which tells you how many data bytes are contained in the chunk exclusive of the header. This allows you to quickly skip chunks you don't recognize or are not interested in. Be aware however that it is permissible to set the data size to -1 to indicate that the size is unknown.

The ASCII chunk header is as follows:

```
"%4c V%d.%d Id %d Parent %d Size %d"
```

A typical ASCII chunk header looks like:

```
"PolH V0.02 Id 2490008 Parent 0 Size 00000947"
```

This chunk has the type `'PolH'` (polygonal data chunk), the major version is 0, the minor version is 2, the chunk id is 2490008, the parent id is 0 (no parent), and there are 947 data bytes following the header in this chunk. I would encourage you to save a few objects in ASCII format and take a look at them in your favorite text editor. This should give you a feel to the general format of Caligari files. To output ASCII files from trueSpace choose the "Save Object As..." option and check the ASCII box in the file dialog box.

The binary header is as follows:

Size	Type	Description
4 bytes	char	Chunk type
2 bytes	short	Major version
2 bytes	short	Minor version
4 bytes	long	Chunk ID
4 bytes	long	Parent ID
4 bytes	long	Number of bytes in chunk data

Some common elements in chunks

Here are some common elements which appear is several different chunks:

Strings

In ASCII format strings are represented as used by printf and scanf. That is 1 or more non-whitespace characters followed by a whitespace character. This allows you to use the `"%s"` format string to read and write strings using printf and scanf.

ASCII format:

```
"%s"
```

Binary format:

Size	Type	Description

2 bytes	short	String length
length	bytes	String in ASCII characters. Does not include a terminating NULL character.

Names

Many chunks contain a name field. To keep each name unique within its context, each name has a number attached to it. We refer to this number as the names dupecount, and it is always >= 0. The ASCII format consists of a string which contains the name optionally followed by a comma and the dupecount (e.g. “Obj,1”). Note if the dupecount is absent then it should be assumed to be zero.

ASCII format:

```
"\nName %s"
```

Binary format:

Size	Type	Description
2 bytes	short	Name dupecount
2+length	string	Name excluding dupecount (see description of strings)

Local Axes

Some chunks define their own local set of axes. This consists of a position or center for the axes and directions for the X, Y, and Z axes. These are given in World coordinates. The X, Y, and Z axes should be perpendicular to each other and are not relative to the center of axes.

ASCII format:

```
"\ncenter %g %g %g"
"\nx axis %g %g %g"
"\ny axis %g %g %g"
"\nz axis %g %g %g"
```

Binary format:

Size	Type	Description
12 bytes	float triple	X, Y, and Z coordinates for center of axes.
12 bytes	float triple	X, Y, and Z coordinates for direction of local X axis.
12 bytes	float triple	X, Y, and Z coordinates for direction of local Y axis.
12 bytes	float triple	X, Y, and Z coordinates for direction of local Z axis.

Current Position

Most chunks which contain 3D geometry define a local space. The current position is a matrix which transforms from local space to world space. The local coordinates are treated as column vectors with a 1 in the fourth position and multiplied by the matrix to get the corresponding World coordinates. Note: trueSpace currently does not use the fourth row of the matrix and this row should always be [0,0,0,1] or the matrix may not be interpreted correctly.

ASCII format:

```
"\nTransform"
```

```
"\n%g %g %g %g"
"\n%g %g %g %g"
"\n%g %g %g %g"
"\n%g %g %g %g"
```

Binary format:

Size	Type	Description
16 bytes	float quad	First row of matrix
16 bytes	float quad	Second row of matrix
16 bytes	float quad	Third row of matrix

*Note: binary format does **not** include row 4 of the matrix. Its assumed to be [0,0,0,1].*

Chunk Types

`'END '` (End of file Chunk)

Chunk type: `'END '`
Major version 1
Minor version 0

This chunk contains no data. It signifies the end of the file and must always be the last chunk in the file.

`'Grou'` (Group Chunk)

Chunk type: `'Grou'`
Major version: 0
Minor version: 1

The group chunk consists of the following:
- Chunk Header
- Name
- Local Axes
- Current Postion

See the appropiate sections for descriptions of each of these.

`'PolH'` (Polygonal Data Chunk)

Chunk type: `'PolH'`
Major version: 0
Minor version: 2

The polygonal data chunk consists of the following:
- Chunk Header
- Name
- Local Axes
- Current Position
- Local Vertex list
- UV Vertex list

Face list

The Local Vertex list is a list of 3D positions in the local coordinate space defined by this chunk's Current Position (see appropiate section). It consists of a count of the number of vertices followed by the vertices. Each vertex consists of three floats for the X, Y, and Z values.

ASCII format:
`"\nWorld Vertices %ld"` - gives number of local vertices following (>= 0)

for each local vertex:
`"\n%f %f %f"`

Binary format:

Size	Type	Description
4 bytes	long	Number of local vertices
12 bytes each	float triples	X, Y, and Z coordinates for each local vertex.

The UV Vertex list is a list of UV (texture) positions. It consists of a count of the number of UV vertices followed by the UV vertices. Each UV vertex consists of two floats for the U and V values.

ASCII format:
`"\nTexture Vertices %ld"`- gives number of UV vertices following (>= 0)

for each UV vertex:
`"\n%f %f"`

Binary format:

Size	Type	Description
4 bytes	long	Number of UV vertices
8 bytes each	float pairs	U and V coordinates for each UV vertex.

The Face list is a list of faces and holes. Each face has a set of flags, a material number, a count of the number of face vertices, and a list of face vertices. Each hole has a count of the number of faces vertices it contains followed by the face vertices. The hole should be interpreted as being a hole in the most recently read face. Each face vertex consists of two integers; these are an index into the local Vertex list followed by an index into the UV Vertex list.

ASCII format:
`"\nFaces %ld"` - gives the number of faces following (>=0)

for each face:
either: `"\nFace verts %hd flags %hd mat %hd"`
or: `"\nHole verts %hd"`

for each face vertex in the face or hole
`"<%ld,%ld> "`

Binary format:

Size	Type	Description
4 bytes	long	Number of faces and holes

?	?	Faces and holes

A face is:

Size	Type	Description
1 bytes	byte	Flags byte (`F_HOLE` flag is not set)
2 bytes	short	Number of vertices in face (not including any holes)
2 bytes	short	Material index for face
8 bytes each	long pairs	Face vertices. Each consists of a long index in the local Vertex list followed by a long index into the UV Vertex list

A hole is:

Size	Type	Description
1 bytes	byte	Flags byte (`F_HOLE` flag is set)
2 bytes	short	Number of vertices in face (not including any holes)
8 bytes each	long pairs	Face vertices. Each consists of a long index in the local Vertex list followed by a long index into the UV Vertex list

There are two flags currently defined for faces:

```
#define F_HOLE          0x08  // (bit 3)
#define F_BACKCULL      0x10  // (bit 4)
```

The `F_HOLE` flag is used to distinguish holes from faces in the binary format. The `F_BACKCULL` flag means that this face should not be displayed if it's facing away from the camera or view. This flag is currently only used on two dimensional objects, where the front and back face of an object occupy the same space.

The normal of a face is obtained by using the right hand rule. Some operations like quad-divide will only work on objects which trueSpace considers to be solids. For an object to be solid, the number of faces in an object minus two must equal the number of vertices plus the number of edges. In addition each edge must be shared by exactly two faces. Thus when a flat polygon is created in trueSpace, it is represented by two polygons; one facing up and one facing down.

`'Mat1'` (Material Chunk)

Chunk type: `'Mat1'`
Major version: 0
Minor version: 5

The material chunk consists of the following:

- Chunk Header
- Material data
- Environment map data (optional)
- Texture map data (optional)
- Bump map data (optional)

The material data contains the shader type, facet type, color (red, green, blue), and some shader attributes: opacity (alpha), ambient coefficient (ka), specular coefficient (ks), hilight size coefficient (exp), and index of refraction (ior). It also contains a material number. All material chunks have a parent chunk and the material number identifies which child material this is. For instance if a material chunk has material number 2 and its parent is a `'PolH'` or polygonal chunk, then this is material number 2 of that chunk and all faces which have their material id set to 2 in the `'PolH'` chunk should use this material.

ASCII format:

```
"\nmat# %d"                          - material number
"\nshader: %s  facet: %s"
"\nrgb %g,%g,%g"                     - material color
"\nalpha %g  ka %g  ks %g  exp %g  ior %g"
```

Current possibilities for the shader field are: "`flat`", "`phong`" and "`metal`".
Current possibilities for the facet field are: "`faceted`", "`auto%d`", and "`smooth`".
The autofacet choice contains a value for the autofacet angle. Thus autofacet with an angle of 45 degrees would be "`auto45`".

Binary format:

Size	Type	Description
2 bytes	short	Material number
1 byte	char	Shader type ('`f`'-flat, '`p`'-phong, '`m`'-metal)
1 byte	char	Facet type ('`f`'-faceted, '`a`'-autofacet, '`s`'-smooth)
1 byte	char	Autofacet angle (0 - 179 degrees)
4 bytes	float	Red component of color (0.0 - 1.0)
4 bytes	float	Green component of color (0.0 - 1.0)
4 bytes	float	Blue component of color (0.0 - 1.0)
4 bytes	float	Opacity (0.0 - 1.0)
4 bytes	float	Ambient coefficient (0.0 - 1.0)
4 bytes	float	Specular coefficient (0.0 - 1.0)
4 bytes	float	Hilight size coefficient (0.0 - 1.0)
4 bytes	float	Index of refraction

If environment map data is present then it starts with an identifying field, the full pathname of the environment map file, and a set of flags. The flags for environment maps are:

```
#define ENVR_CUBIC      0x01  // (bit 0)
#define ENVR_USE        0x02  // (bit 1)
```

The `ENVR_USE` flag is always set. The `ENVR_CUBIC` flag is set to indicate that the environment map is a cubic map, otherwise its assumed to be a spherical environment map.

ASCII format:

```
"\nenvironment: %s"              - full pathname of environment file
"\nflags %d"
```

Binary format:

Size	Type	Description
2 bytes	chars	Environment map data identifier ('`e:`')
1 byte	char	Environment map flags
2+length	string	Pathname of environment map file (see **Strings** section)

If texture map data is present then it starts with an identifying field, the full pathname of the texture map file, an offset in U and V, the number of times to repeat the texture map in U and V, and a set of flags. The flags for texture maps are:

```
#define TXTR_OVERLAY    0x01  // (bit 0)
#define TXTR_USE        0x02  // (bit 1)
```

The TXTR_USE flag is always set. If TXTR_OVERLAY flag is set then the object's color will show through where the texture map is at least partially transparent. Otherwise the object will be genuinely transparent in those areas.

ASCII format:

```
"\ntexture: %s"                    - full pathname of texture file
"\noffset %g,%g  repeats %g,%g  flags %hd"
```

Binary format:

Size	Type	Description
2 bytes	chars	Texture map data identifier ('t:')
1 byte	char	Texture map flags
2+length	string	Pathname of texture map file (see **Strings** section)
4 bytes	float	U offset for texture map
4 bytes	float	V offset for texture map
4 bytes	float	Number of times to repeat texture map in U direction
4 bytes	float	Number of times to repeat texture map in V direction

If bump map data is present then it starts with an identifying field, the full pathname of the bump map file, an amplitude for the bump map, an offset in U and V, the number of times to repeat the bump map in U and V, and a set of flags. The flags for bumps maps are:

```
#define BUMP_USE         0x02  // (bit 1)
```

The BUMP_USE flag is always set.

ASCII format:

```
"\nbump: %s"                       - full pathname of bump file
"\noffset %g,%g  repeats %g,%g  amp %g  flags %hd"
```

Binary format:

Size	Type	Description
2 bytes	chars	Bump map data identifier ('b:')
1 byte	char	Bump map flags
2+length	string	Pathname of bump map file (see **Strings** section)
4 bytes	float	U offset for bump map
4 bytes	float	V offset for bump map
4 bytes	float	Number of times to repeat bump map in U direction
4 bytes	float	Number of times to repeat bump map in V direction
4 bytes	float	Relative amplitude of bump map

'PrTx' (Procedural Texture Chunk) ***New for trueSpace2***

Chunk type: 'PrTx'
Major version: 0
Minor version: 1

The procedural texture chunk consists of the following:

Chunk Header
Material data
Environment map data (optional)
One of Granite, Marble or Wood data (optional)

Bump map data (optional)

The procedural texture chunk is very similar to the material chunk. If a material is to be procedurally textured, then this chunk will appear instead of a `'Mat1'` chunk. The only difference between the two chunks is that where a `'Mat1'` has an optional texture map specification, the `'PrTx'` chunk has an optional Granite, Marble or Wood texture specification. Other than this, everything else is the same. Please see the material chunk description above for information about all chunk elements except the Granite, Wood and Marble data. The RST solid texture coordinates are based on each object's untransformed world vertices.

If Granite data is present then it begins with and identifying field, four RGBA colors, four contribution amounts (one for each color), sharpness, R, S and T scale values and a seed value.

ASCII format:

```
"\ngranite:"
"\nrgba1 %g,%g,%g,%g rgba2 %g,%g,%g,%g"
"\nrgba3 %g,%g,%g,%g rgba4 %g,%g,%g,%g"
"\namount1 %g amount2 %g amount3 %g amount4 %g"
"\nsharpness %g scale %g,%g,%g seed %hd"
```

Binary format:

Size	Type	Description
2 bytes	chars	Granite texture data identifier (`'g:'`)
1 byte	char	Reserved (set to 0 for now)
4 bytes	float	Color 1 red (0.0 - 1.0)
4 bytes	float	Color 1 green (0.0 - 1.0)
4 bytes	float	Color 1 blue (0.0 - 1.0)
4 bytes	float	Color 1 alpha (0.0 - 1.0)
4 bytes	float	Color 2 red (0.0 - 1.0)
4 bytes	float	Color 2 green (0.0 - 1.0)
4 bytes	float	Color 2 blue (0.0 - 1.0)
4 bytes	float	Color 2 alpha (0.0 - 1.0)
4 bytes	float	Color 3 red (0.0 - 1.0)
4 bytes	float	Color 3 green (0.0 - 1.0)
4 bytes	float	Color 3 blue (0.0 - 1.0)
4 bytes	float	Color 3 alpha (0.0 - 1.0)
4 bytes	float	Color 4 red (0.0 - 1.0)
4 bytes	float	Color 4 green (0.0 - 1.0)
4 bytes	float	Color 4 blue (0.0 - 1.0)
4 bytes	float	Color 4 alpha (0.0 - 1.0)
4 bytes	float	Amount of color 1 (0.0 - 1.0)
4 bytes	float	Amount of color 2 (0.0 - 1.0)
4 bytes	float	Amount of color 3 (0.0 - 1.0)
4 bytes	float	Amount of color 4 (0.0 - 1.0)
4 bytes	float	Sharpness (0.0 - 1.0)
4 bytes	float	R scale (0.0 - 10.0)
4 bytes	float	S scale (0.0 - 10.0)
4 bytes	float	T scale (0.0 - 10.0)
2 bytes	short	Seed (1 - 32767)

If Marble data is present then it begins with an identifying field, stone color and alpha, vein color and alpha, sharpness, R, S and T scale values, turbulence, grain axis, and a seed value. The grain axis is one of the following selections:

```
#define PROC_GRAIN_R    0x01  // (bit 1)
#define PROC_GRAIN_S    0x02  // (bit 2)
#define PROC_GRAIN_T    0x04  // (bit 3)
```

ASCII format:

```
"\nmarble:"
"\nstonergba %g,%g,%g,%g  veinrgba %g,%g,%g,%g"
"\nsharpness %g  scale %g,%g,%g"
"\nturbulence %hd  grain %hd  seed %hd"
```

Binary format:

Size	Type	Description
2 bytes	chars	Marble texture data identifier ('m:')
1 byte	char	Reserved (set to 0 for now)
4 bytes	float	Stone color red (0.0 - 1.0)
4 bytes	float	Stone color green (0.0 - 1.0)
4 bytes	float	Stone color blue (0.0 - 1.0)
4 bytes	float	Stone color alpha (0.0 - 1.0)
4 bytes	float	Vein color red (0.0 - 1.0)
4 bytes	float	Vein color green (0.0 - 1.0)
4 bytes	float	Vein color blue (0.0 - 1.0)
4 bytes	float	Vein color alpha (0.0 - 1.0)
4 bytes	float	Sharpness (0.0 - 1.0)
4 bytes	float	R scale (0.0 - 10.0)
4 bytes	float	S scale (0.0 - 10.0)
4 bytes	float	T scale (0.0 - 10.0)
2 bytes	short	Vein turbulence (1 - 10)
2 bytes	short	RST grain flag (see above)
2 bytes	short	Seed (1 - 32767)

If Wood data is present then it begins with an identifying field, spring and summer wood RGBA colors, ratio of spring to summer wood, ring density, ring width variation, ring shape variation, R, S and T center for wood rings, R, S and T scale, grain axis and a seed value. See the Marble description for information about the grain axis. Note that the RST center values are not presently exposed in the trueSpace2 user interface and should be set to (1.0, 1.0, 1.0).

ASCII format:

```
"\nwood:"
"\nspringrgba %g,%g,%g,%g  summerrgba %g,%g,%g,%g"
"\nldratio %g  rdensity %g  rwidthvar %g  rshapevar %g"
"\ncenter %g,%g,%g  scale %g,%g,%g"
"\ngrain %hd  seed %hd"
```

Binary format:

Size	Type	Description
2 bytes	chars	Wood texture data identifier ('w:')
1 byte	char	Reserved (set to 0 for now)
4 bytes	float	Spring wood color red (0.0 - 1.0)
4 bytes	float	Spring wood color green (0.0 - 1.0)
4 bytes	float	Spring wood color blue (0.0 - 1.0)

4 bytes	float	Spring wood color alpha (0.0 - 1.0)
4 bytes	float	Summer wood color red (0.0 - 1.0)
4 bytes	float	Summer wood color green (0.0 - 1.0)
4 bytes	float	Summer wood color blue (0.0 - 1.0)
4 bytes	float	Summer wood color alpha (0.0 - 1.0)
4 bytes	float	Spring to summer ratio (0.0 - 1.0)
4 bytes	float	Ring density (0.0 - 10.0)
4 bytes	float	Ring width variation (0.0 - 1.0)
4 bytes	float	Ring shape variation (0.0 - 1.0)
4 bytes	float	R ring center (set to 1.0)
4 bytes	float	S ring center (set to 1.0)
4 bytes	float	T ring center (set to 1.0)
4 bytes	float	R scale (0.0 - 10.0)
4 bytes	float	S scale (0.0 - 10.0)
4 bytes	float	T scale (0.0 - 10.0)
2 bytes	short	RST grain flag (see above)
2 bytes	short	Seed (1 - 32767)