.WLD File Reference

by Windcatcher

Version 1.1

Overview

EQWldData.pas in the OpenZone source is a direct translation of the corresponding ZoneConverter file, but also has bug fixes and improvements.

Here's a rundown of the fragment types as I understand them and some basic documentation below:

0x03 - Texture Bitmap Name(s)
0x04 - Texture Bitmap Info
0x05 - Texture Bitmap Info Reference
0x06 - Two-dimensional Object
0x07 - Two-dimensional Object Reference
0x08 - Camera
0x09 - Camera Reference
0x10 - Skeleton Track Set
0x11 - Skeleton Track Set Reference
0x12 - Mob Skeleton Piece Track
0x13 - Mob Skeleton Piece Track Reference
0x14 - Static or Animated Model Reference/Player Info
0x15 - Object Location
0x16 - Zone Unknown
0x17 - Polygon Animation?
0x18 - Polygon Animation Reference?
0x1B - Light Source
0x1C - Light Source Reference
0x21 - BSP Tree
0x22 - BSP Region
0x28 - Light Info
0x29 - Region Flag
0x2A - Ambient Light
0x2C - Alternate Mesh
0x2D - Mesh Reference
0x2F - Mesh Animated Vertices Reference
0x30 - Texture
0x31 - Texture List
0x32 - Vertex Color
0x33 - Vertex Color Reference
0x35 - First Fragment
0x36 - Mesh
0x37 - Mesh Animated Vertices

There are other fragment types in the ZoneConverter source, but I haven't encountered them. Here I talk about all of the ones I've seen.

Miscellaneous

0x35 - First Fragment – Most .WLD files I've encountered begin with one of these. This is because the fragment reference mechanism cannot reference the very first fragment in the file by direct index, and so a placeholder is needed.

Textures

0x03 - Texture Bitmap Name(s) - Contains the names of one or more bitmaps used in a particular texture. When there is more than one bitmap in the fragment it means that the texture is animated (e.g. water).

0x04 - Texture Bitmap Info - Refers to a 0x03 fragment. Also contains flags to tell the client information about this particular texture (normal or animated).

0x05 - Texture Bitmap Info Reference - Refers to a 0x04 fragment so 0x04 fragments can be reused.

0x30 - Texture - Refers to a 0x05 fragment. Contains flags to tell the client information about this texture (normal, semitransparent, transparent, masked, etc.) Having this fragment separated from the 0x05 fragment means that the zone can have multiple flavors of the same texture bitmap (e.g. one that is normal, one that is semitransparent, etc.)

0x31 - Texture List - Contains references to all of the 0x30 textures used in the zone (or, in the case of placeable objects, in that particular object).

Meshes

0x36 - Mesh - Contains vertex, normal, color, and polygon information for a mesh. In the case of zones, the mesh is a subdivision of the zone. In the case of placeable objects, the mesh contains the entire information for the object.

Notes on 0x36 fragment:

1. Fragment1 refers to a 0x31 fragment to tell the client what textures are used.
2. Polygons are sorted by texture index. That is, all polygons in the Data5 area that use a particular texture are grouped together.
3. Fragment2 optionally refers to a 0x2F fragment if the mesh is animated (e.g. trees or flags that sway in the breeze).
4. Fragment4 always refers to the first 0x03 fragment in the file (I have no idea why).
5. I don't fully understand this fragment. The Data6 and Data9 areas have something to do with mob models, but I don't know how they work yet.
6. There are new-format and old-format .WLD files. They have different values in the .WLD header and the main difference is in the 0x36 fragment. In new-format files, the texture coordinate values are signed 32-bit values; in old-format files they're signed 16-bit values. At this time OpenZone only exports old-format files but it would be no great effort to switch it to new-format files.

0x37 - Mesh Animated Vertices - For a given 0x36 fragment, this fragment contains a number of animation "frames". For each frame it contains a complete vertex list that supercedes the vertex list in the 0x36 fragment. For instance, if there are three frames and 15 vertices, there will be three sets of 15 vertex values in the 0x37 fragment and they will be used in lieu of the 15 vertex values in the 0x36 fragment.

0x2F - Mesh Animated Vertices Reference - Refers to a 0x37 fragment so it can be reused (e.g. for flags that all have the same shape but have different textures).

Zones

0x21 - BSP Tree - BSP stands for "binary space partition". Basically the zone is broken up into regions so the client can quickly find polygons that are near the player or mobs for purposes of collision avoidance. Normally this is done along a simple grid, but the zone also has to be broken up so that "special" regions (water, lava, PvP, ambient lighting, and others) are distinct.

0x22 - BSP Region - For each node in the BSP tree there is a 0x22 fragment that describes the region. It contains information on the region's bounds and an optional reference to a 0x36 fragment if there are polygons in that region. Also contains an RLE-encoded array of data that tells the client which regions are "nearby". That way, if a player enters one of the "nearby" regions, the client knows that objects in the region are visible to the player. The client does this to know when it has to make sure mobs fall to the ground instead of stay at the spawn points, which might be in midair.

0x1B - Light Source - I suspect this defines the ambient light level in a zone (see "Light sources" below for info).

0x08 - Camera - I don't know what the parameters mean yet.

0x1C - Light Source Reference - See "Light sources" below for info.

0x2A - Ambient Light - Refers to a 0x1C fragment. Contains a list of numbers that refer to the 0x22 fragments in the zone (e.g. if there are 100 0x22 fragments then the numbers will be in the range 0-99). This fragment tells the client which regions have the particular light setting. I suspect that you could conceivably have some regions with one ambient light setting, other regions with another ambient light setting, etc. by having multiple 0x1B-0x1C-0x2A sets.

0x09 - Camera Reference - Refers to a 0x08 fragment. I don't know its purpose.

0x14 - Player Info - I don't know its purpose. Its Fragment1 value seems to use a “magic” string: “FLYCAMCALLBACK”.

0x16 - Zone Unknown - It's used in zone files for some reason...

0x15 - Object Location - In zone files, this might contain the safe point?

0x29 - Region Flag - This is similar to the 0x2A fragment in that it contains a list of numbers, where each number refers to a 0x22 region. It tells the client that those regions are "special". The name of the fragment is "magic" in that it determines how the regions are flagged:

"WT_ZONE" .... Regions are underwater

"LA_ZONE" .... Regions are lava

"DRP_ZONE" .... Regions are PvP areas

"DRNTP##########_ZONE" .... e.g. DRNTP00025-02698-645.6-00020999_ZONE. This seems to tell the client that these regions constitute a zoneline. If the player enters one of these regions the client knows the player is zoning and knows the destination. I don't know if the client makes use of this since I don't think every zone has this at all zone points, but it looks interesting. I don't understand the format of the numbered part of the name.

Animated (Mob) models

0x14 - Mob Model Reference - This is the starting point for all models, whether animated or static. For animated models its Data2 field refers to a 0x11 fragment. Fragment1 contains a “magic” string: “SPRITECALLBACK”.

0x11 - Skeleton Track Set Reference - Refers to a 0x10 fragment.

0x10 - Skeleton Track Set - Contains multiple references to 0x13 fragments as well as references to one or more 0x2D fragments (generally more than one). Note that this typically only refers to the basic stance of a mob model and not alternate animations (attacking, walking, running, etc.). Software reading the .WLD file should use the name of the first 0x13 fragment this references to discover alternate animations. Alternate animations will have sets of 0x13 and 0x12 fragments where the name of the alternate fragments have a prefix before their names (e.g. if the base 0x13 fragment’s name is “LION_TRACK” then an alternate could be “C01LION_TRACK”). Each 0x13 fragment in an alternate set will have the same prefix before its name, and the rest of the name will correspond to the analogous 0x13 fragment in the base animation set. Different animation sets will have different prefixes (e.g. “C01” for one combat animation, “C02” for another combat animation, etc.).

0x12 - Mob Skeleton Piece Track - Describes how an individual part of a skeleton (e.g. a forearm) is shifted and/or rotated relative to its “parent” piece.

0x13 - Mob Skeleton Piece Track Reference - Refers to a 0x12 fragment.

0x2D - Mesh Reference - Refers to a 0x36 fragment.

0x36 - Mesh - Contains the actual vertex and polygon data. Also contains information on textures, texture coordinates, shading, and skeleton piece assignment.

Light sources

0x1B - Light Source - Contains information on light color. I don't know what the other parameters in it mean.

0x1C - Light Source Reference - Refers to a 0x1B fragment.

0x28 - Light Info - Refers to a 0x1C fragment. Contains light position and radius. I don't know what the other parameters mean.

Static Models

0x14 - Mob Model Reference - This is the starting point for all models, whether animated or static. For static models its Data2 field refers to a 0x2D fragment. Fragment1 contains a “magic” string: “SPRITECALLBACK”.

0x2D - Mesh Reference - Refers to the 0x36 fragment.

0x36 - Mesh - Contains the actual vertex and polygon data. Also contains information on textures, texture coordinates, and shading.

0x32 - Vertex Color - For each object that has been placed, there is one of these (put 100 trees in your zone and there are 100 of these fragments). It contains vertex shading information for each object. For example, if you have a torch near some trees, those trees should have their polygons shaded based on the light color, angle of incidence, distance, and any intervening polygons. The EQ client does *not* dyamically shade polygons in a zone; all polygons must be shaded in this way (including 0x36 fragments in the main zone file).

0x33 - Vertex Color Reference - Refers to a 0x32 fragment.

0x15 - Object Location - Contains object position, rotation, and size. Refers to a 0x33 fragment. When used in the main zone file, this fragment contains information for the whole zone (see frmMainUnit.pas in the OpenZone source).

0x17 - Polygon Animation? - I don't know what this does. I suspect it has something to do with polygon animation. I've never seen this for mob models but I have seen it for some trees.

0x18 - Polygon Animation Reference? - Refers to a 0x17 fragment. I don't know its purpose.

Notes on Data Types Referenced in this Document

Generally all floating-point values (FLOATs) seen in .WLD files are signed values. When specifically noted, integral types (BYTE, WORD, and DWORD) are either signed or unsigned. In all other cases it might not be known (or might not matter) whether they are signed or unsigned.

FLOATs are 32-bits long (analogous to the "float" type in C).

There is also a data type called DATAPAIR. A DATAPAIR is a DWORD followed by a FLOAT, that is, a 32-bit integer followed by a 32-bit floating-point value. I'm not sure whether the DWORD in a DATAPAIR is signed or unsigned.

.WLD File Organization

.WLD files can be broken into three sections:

- Header
- String hash
- Fragments

.WLD Header

The .WLD header consists of seven DWORDs:

Magic : DWORD

This always contains 0x54503D02. It identifies the file as a .WLD file.

Version : DWORD

For old-format .WLD files, this always contains 0x00015500. For new-format .WLD files, this always contains 0x1000C800.

FragmentCount : DWORD

Contains the number of fragments in the .WLD file, minus 1 (that is, the highest fragment index, starting at 0).

Header3 : DWORD

Should contain the number of 0x22 BSP Region fragments in the file.

Header4 : DWORD

Unknown purpose. Should contain 0x000680D4.

StringHashSize : DWORD

Contains the size of the string hash in bytes. All strings in .WLD files are XOR-encoded using the following rotating set of flags:

```
0x95, 0x3A, 0xC5, 0x2A, 0x95, 0x7A, 0x95, 0x6A
```

That is, the first byte is XOR'ed with 0x95, the second byte with 0x3A, and so on. The set repeats at the ninth byte. Repeating the operation decodes the string. Kudos to the original author of ZoneConverter for figuring this out.

The first byte of the string hash is always a junk value (actually encoded zero which results in 0x95) and is used for fragments that have no string name. Encoded strings therefore start at position 1 in the string hash. The string hash is nothing more than a bunch of null-terminated strings that have been concatenated together and encoded.

Header6 : DWORD

Unknown purpose. This is a guess (that seems to work): it should contain the number of fragments in the file, minus the number of 0x03 fragments, minus 6.

Comprehensive Fragment Reference

Introduction

There are two basic kinds of fragments: plain and reference. Plain fragments are pure data structures that begin with three 32-bit fields, whereas reference fragments also come with a 32-bit reference to another fragment. The ID contains the specific fragment type. This type determines what follows the fragment header data and whether the fragment is a plain or reference fragment.

Basic fragments

All fragments (plain and reference) begin with the following data:

Size : DWORD

Size of the fragment in bytes. All fragments are padded such that Size is evenly divisible by 4 and Size should reflect the padded value.

ID : DWORD

The fragment type. This will typically be a value in the range 0x03 to 0x37 and tells the file reader which specific kind of fragment it is. Some fragment types are plain fragments and some are reference fragments: the ID determines which.

NameReference : DWORD

Each fragment can have a string name, which is stored in encoded form in the .WLD file's string hash. The NameReference gives a way to retrieve the name. If the fragment has a string name, its NameReference should contain the negative value of the string's index in the string hash. For example, if the string is at position 31 in the string hash, then NameReference should contain –31. Values greater than 0 mean that the fragment doesn't have a string name.

In reality, a value of 0 also means that the fragment doesn't have a string name, and the first byte in the string hash is always preallocated to reflect this (it's a null character that is encoded along with everything else). For all fragments that don't have a name their NameReference field should contain zero, except for the 0x35 fragment: the 0x35 fragment should contain 0xFF000000 in its NameReference field.

Reference fragments

Reference fragments contain this additional field:

Reference: DWORD

This can be either a string reference or a fragment reference. If it is a fragment reference it must reference a fragment that has already been loaded from the file. It can reference it using one of two ways:

1. By name
2. By fragment index

If Reference contains a value that is less than or equal to zero, the value is negated and 1 is subtracted from it. Then a null-terminated string is loaded from the string hash, starting at that position. Every fragment that has been loaded is checked to see if its name matches the string that was loaded. If a match is found, then the reference is considered to point to that fragment.

If a match is not found, then the reference is considered to point only to that string instead of to a fragment. There are cases where certain fragments point to “magic” strings that cause the client to do something special. This is how that happens.

If Reference contains a value greater than zero then it is considered to be a direct reference to another fragment, based on the order in which they were loaded from the file. It’s worth noting that the first fragment in the file (which would have an index of zero) cannot be referenced in this manner. This is the reason why the 0x35 fragment exists and why it’s a good idea to begin all .WLD files with one: it serves as a placeholder to ensure that the next fragment can always be referenced.

All fragments are padded to end on DWORD boundaries. The Size field above must reflect this padding.

0x03 — Texture Bitmap Name(s) — PLAIN

Notes

This fragment references one or more texture filenames. For most textures (every one I have ever seen) it only references one filename.

Fields

Size1 : DWORD

Contains the number of texture filenames in this fragment. Most of the time there is only one name.

Entries (there are Size1 of them):

NameLength: WORD

Contains the length of the filename in bytes.

FileName: BYTEs

The encoded filename. See the introduction above for a description of string coding.

The client apparently looks for certain filenames and substitutes built-in textures in their place. When using an animated fire texture where the names are fire1.bmp, fire2.bmp, fire3.bmp and fire4.bmp, respectively, the client always uses its built-in fire textures instead (they look great anyway). This only happens **when the textures are used by a placeable object** and not when the textures are in the main zone file. It is unknown whether the substitution depends on the presence and exact order of all four textures.

0x04 — Texture Bitmap Info — PLAIN

Notes

This fragment represents an entire texture rather than merely a bitmap used by that texture. The conceptual difference from 0x03 fragments is that textures may be animated; the 0x04 fragment represents the entire texture including all bitmaps that it uses, whereas an 0x03 fragment would represent only a single bitmap in the animated sequence.

Fields

Flags : DWORD

Bit 3........ If 1, texture is animated (has more than one 0x03 reference). Also means that Params1 exists.
Bit 4........ If 1, Params2 exists. Seems to always be set.

Size : DWORD

Contains the number of 0x03 fragment references. It will only reference one 0x03 fragment for most textures but should reference more than one for animated textures (e.g. water).

Params1 : DWORD

Only exists if Flags bit 3 is set to 1.

Params2 : DWORD

Only exists if Flags bit 4 is set to 1.

References: DWORDs

One or more references to 0x03 fragments.

0x05 — Texture Bitmap Info Reference — REFERENCE

Reference points to a 0x04 Texture Bitmap Info fragment.

Fields

Flags : DWORD

Its purpose is unknown, but it seems to always contain 0x50.

0x06 — Two-dimensional Object — PLAIN

Notes

This fragment is rarely used. It describes objects that are purely two-dimensional in nature. Examples are coins and blood spatters.

Fields

Flags : DWORD

Its purpose is unknown. The function of the known bits is as follows:

Bit 0........ If 1, Params3 exists.
Bit 1........ If 1, Params4 exists.
Bit 2........ If 1, Params5 exists.
Bit 3........ If 1, Params6 exists.
Bit 7........ If 1, Params2 exists.

SubSize1 : DWORD

Its purpose is unknown.

Size1 : DWORD

Its purpose is unknown.

Params1 : 2DWORDs

Its purpose is unknown, though I suspect it might be the object’s size.

Fragment : DWORD

Its purpose is unknown.

Params2 : FLOAT

Its purpose is unknown. It only exists if bit 7 of Flags is 1.

Params3 : 3 DWORDs

Their purpose is unknown. They only exist if bit 0 of Flags is 1.

Params4 : FLOAT

Its purpose is unknown. It only exists if bit 1 of Flags is 1.

Params5 : DWORD

Its purpose is unknown. It only exists if bit 2 of Flags is 1.

Params6 : DWORD

Its purpose is unknown, though it typically contains 100. It only exists if bit 3 of Flags is 1.

Data1 entries (there are Size1 of these):

> **Data6Params1 : DWORD**
>
> Its purpose is unknown. It typically contains 512 (0x200).
>
> **Data6Flags : DWORD**
>
> The most significant bit of this field (0x80000000) is a flag of some sort. The other bits constitute another size field which we shall call Data6Size here.
>
> Data6Data entries (there are Data6Size of these):
>
> **Data6DataParams1 : DWORD**
>
> Its purpose is unknown. It typically contains 64 (0x40).
>
> **Data6DataFragments : SubSize1 DWORDs**
>
> These point to one or more 0x03 Texture Bitmap Name fragments (one if the object is static or more than one if it has an animated texture, such as blood from a weapon strike).

Params7Params1 : DWORD

Its purpose is unknown.

Params7Flags : DWORD

Its purpose is unknown. The function of the known bits is as follows:

Bit 0........ If 1, Params7Params2 exists.
Bit 1........ If 1, Params7Params3 exists.
Bit 2........ If 1, Params7Params4 exists.
Bit 3........ If 1, Params7Fragment exists.
Bit 4........ If 1, Params7Matrix exists.
Bit 5........ If 1, Params7Size and Params7Data exist.

Params7Params2 : DWORD

Its purpose is unknown. Only exists if bit 0 of Params7Flags is 1.

Params7Params3 : FLOAT

Its purpose is unknown. Only exists if bit 1 of Params7Flags is 1.

Params7Params4 : FLOAT

Its purpose is unknown. Only exists if bit 2 of Params7Flags is 1.

Params7Fragment : DWORD

Its purpose is unknown. Only exists if bit 3 of Params7Flags is 1.

Params7Matrix : 9 DWORDs

Its purpose is unknown, though it looks like some sort of transformation matrix. Only exists if bit 4 of Params7Flags is 1.

Params7Size : DWORD

Its purpose is unknown. Only exists if bit 5 of Params7Flags is 1.

Params7Data : (Params7Size * 2) DWORDs

Their purpose is unknown. Only exists if bit 5 of Params7Flags is 1.

0x07 — Camera Reference — REFERENCE

Reference points to a 0x06 Two-dimensional Object fragment.

Fields

Flags : DWORD

Its purpose is unknown, but it always seems to contain 0.

0x08 — Camera — PLAIN

Notes

This fragment is poorly understood. It seems to contain 26 parameters, some of which are DWORDS (32-bit integers) and some of which are FLOATS (32-bit floating-point values). Until more is known, they are here described as Params[0..25] and their known values are documented.

In main zone files, the name of this fragment always seems to be CAMERA_DUMMY.

Fields

All fields not mentioned contain zero (0).

Params[1] :	0	DWORD
Params[2] :	1	DWORD
Params[5] :	-1.0	FLOAT
Params[6] :	1.0	FLOAT
Params[8] :	1.0	FLOAT
Params[9] :	1.0	FLOAT
Params[11] :	1.0	FLOAT
Params[12] :	-1.0	FLOAT
Params[14] :	-1.0	FLOAT
Params[15] :	-1.0	FLOAT
Params[16] :	4	DWORD
Params[20] :	1	DWORD
Params[21] :	2	DWORD
Params[22] :	3	DWORD
Params[24] :	1	DWORD
Params[25] :	11	DWORD

0x09 — Camera Reference — REFERENCE

Reference points to a 0x08 Camera fragment.

Fields

Flags : DWORD

Its purpose is unknown, but it always seems to contain 0.

0x10 — Skeleton Track Set — PLAIN

Notes

This fragment describes a skeleton for an entire animated model, and is used for mob models. The overall skeleton is contained in a 0x10 Skeleton Track Set fragment and is structured as a hierarchical tree. For example, a pelvis piece might connect to chest, left thigh, and right thigh pieces. The chest piece might connect to left bicep, right bicep, and neck pieces. The left bicep piece might connect to a left forearm piece. The left forearm piece might connect to a left hand piece. The idea is to start at the base “stem” piece in the skeleton and recursively walk the tree to each successive piece.

For each piece there is a 0x13 Mob Skeleton Piece Track Reference fragment, which references one 0x12 Mob Skeleton Piece Track fragment. Each 0x12 fragment defines how that piece is rotated and/or shifted relative to its parent piece.

Fields

Flags : DWORD

Bit 0........ If 1, Params1[0..2] fields exist.
Bit 1........ If 1, Params2 exists.
Bit 9........ If 1, Size2, Fragment3, and Data3 fields exist.

Size1 : DWORD

Number of track reference entries (see below)

Fragment : DWORD

Optionally points to a 0x18 Polygon Animation Reference? fragment.

Params1[0] : DWORD

Unknown purpose. Only exists if bit 0 of Flags is 1.

Params1[1] : DWORD

Unknown purpose. Only exists if bit 0 of Flags is 1.

Params1[2] : DWORD

Unknown purpose. Only exists if bit 0 of Flags is 1.

Params2 : FLOAT

Unknown purpose.

Entries (there are Size1 of them):

Entry1NameReference : DWORD

This seems to refer to the name of either this or another 0x10 fragment. It seems that at least one name reference points to the name of this fragment.

Entry1Flags : DWORD

Usually zero.

Entry1Fragment1 : DWORD

Reference to a 0x13 Mob Skeleton Piece Track Reference fragment.

Important: animated models generally only reference a basic set of fragments necessary to render the model but not animate it. There will generally be other sets of 0x13 fragments where each set corresponds to a different animation of the model. Software reading .WLD files must use the name of the first 0x13 fragment referenced by the 0x10 Skeleton Track Set to discover any other animation sets. The first fragment of any alternate animation set will have the same name as the first 0x13 fragment, with an additional prefix. All other 0x13 fragments in that same set will likewise correspond to their counterparts in the basic animation set. Different animation sets will have different prefixes (e.g. “C01” for one combat animation, “C02” for another combat animation, etc.). All alternate animation sets for a particular model generally immediately follow the 0x10 Skeleton Track Set fragment (with the 0x11 Skeleton Track Set Reference immediately following those). I don’t know if this is a necessary arrangement.

Entry1Fragment2 : DWORD

Sometimes refers to a 0x2D Mesh Reference fragment.

Entry1Size : DWORD

Tells how many Entry1Data entries there are.

Entry1Data : DWORDs

Each of these contains the index of the next piece in the skeleton tree. A Skeleton Track Set is a hierarchical tree of pieces in the skeleton. It generally starts with a central “stem” and branches out to a skeleton’s extremities. For instance, the first entry might be the stem; that entry might point to the pelvis entry; the pelvis entry might point to the left thigh, right thigh, and chest entries; and those entries would each point to other parts of the skeleton. The exact topography of the tree depends upon the overall structure of the skeleton. The proper way to use a Skeleton Track Set fragment is to start with the first entry and recursively walk the tree by following each entry’s Entry1Data field to other connected pieces.

It’s also worth noting that, although an entry might reference a 0x13 Mob Skeleton Piece Track Reference fragment in its EntityFragment1 field, that does not mean it will be valid for rendering (see the 0x12 Mob Skeleton Piece Track fragment for more information). Many model skeletons apparently contain extraneous pieces that have an unknown purpose, though I suspect that they are for determining attachment points for weapons and shields and are otherwise not meant to be rendered. These pieces are generally not referenced by the 0x36 Mesh fragments that the skeleton indirectly references (via 0x2D Mesh Reference fragments).

Size2 : DWORD

Tells how many Fragment3 and Data3 entries there are. This field only exists if the proper bit in the Flags field is set.

Fragment3 : DWORDs

There are Size2 of these. This field only exists if the proper bit in the Flags field is set. These entries generally point to 0x2D Mesh Reference fragments and outline all of the meshes in the animated model. For example, there might be a mesh for a model’s body and another one for the head.

Data3 : DWORDs

There are Size2 of these. It’s unknown what they typically contain. This field only exists if the proper bit in the Flags field is set.

0x11 — Skeleton Track Set Reference — REFERENCE

Reference points to a 0x10 Skeleton Track Set fragment.

Fields

Params1 : DWORD

Apparently must be zero.

0x12 — Mob Skeleton Piece Track — PLAIN

Notes

This fragment describes how a skeleton piece is shifted or rotated relative to its parent piece. The overall skeleton is contained in a 0x10 Skeleton Track Set fragment and is structured as a hierarchical tree (see that fragment for information on how skeletons are structured). The 0x12 fragment contains information on how that particular skeleton piece is rotated and/or shifted relative to its parent piece.

Rotation and shifting information is contained as a series of fractions. The fragment contains one denominator value for rotation and another for translation (X, Y, Z, shift). It contains one numerator each for X, Y, Z rotation and shift, for a total of eight values. For rotation, the resulting value should be multiplied by Pi / 2 radians (i.e. 1 corresponds to 90 degrees, 2 corresponds to 180 degrees, etc.).

For rendering polygons, the X, Y, Z rotation and shift information in this fragment should be taken into account by adding them to the rotation and shift values passed from the parent piece (that is, rotation and shift are cumulative). However, before adding the shift values, the X, Y, and Z shift values should first be rotated according to the **parent's** rotation values. The rotation values in this fragment represent the orientation of this piece relative to the parent so calculating its starting position should **not** take its own rotation into account. Software rendering a skeleton piece should perform the following steps in this order:

- Calculate the X, Y, and Z shift values from this fragment
- Rotate the shift values according to the rotation values from the parent piece
- Add the shift values to the shift values from the parent piece
- Calculate the X, Y, and Z rotation values from this fragment
- Add the rotation values to the rotation values from the parent piece
- Adjust the vertices for this piece by rotating them using the new rotation values and then shifting them by the new shift values (or save the rotation and shift values for this piece to be looked up later on when rendering)
- Process the next piece in the tree with the new rotation and shift values
- When all pieces have been processed, render all meshes in the model, using either the adjusted vertex values (more efficient) or looking up the corresponding piece for each vertex and adjusting the vertex values according to the adjusted rotation and shift values calculated above (less efficient).

Fields

Flags : DWORD

Bit 3........ If 1, Data2 exists (though I'm not at all sure about this since I have yet to see an example). It could instead mean that the rotation and shift entries are unsigned DWORDs or it could mean that they're 32-bit FLOATs.

Size : DWORD

Tells how many Data1 and Data2 entries there are.

RotateDenominator : SIGNED WORD (signed 16-bit value)

This represents the denominator for the piece's X, Y, and Z rotation values. It's vital to note that it is possible to encounter situations where this value is zero. I have seen this for pieces with no vertices or polygons and in this case rotation should be ignored (just use the existing rotation value as passed from the parent piece). My belief is that such pieces represent attachment points for weapons or items (e.g. shields) and otherwise don't represent a part of the model to be rendered.

RotateXNumerator : SIGNED WORD (signed 16-bit value)

The numerator for rotation about the X axis.

RotateYNumerator : SIGNED WORD (signed 16-bit value)

The numerator for rotation about the Y axis.

RotateZNumerator : SIGNED WORD (signed 16-bit value)

The numerator for rotation about the Z axis.

ShiftXNumerator : SIGNED WORD (signed 16-bit value)

The numerator for translation along the X axis.

ShiftYNumerator : SIGNED WORD (signed 16-bit value)

The numerator for translation along the Y axis.

ShiftZNumerator : SIGNED WORD (signed 16-bit value)

The numerator for translation along the Z axis.

ShiftDenominator : SIGNED WORD (signed 16-bit value)

The denominator for the piece X, Y, and Z shift values. Like the rotation denominator, software should check to see if this is zero and ignore translation in that case.

Data2 : 4 DWORDs

There are (4 x Size) DWORDs here. Their purpose is unknown. This field exists only if the proper bit in Flags is set. It's possible that this is a bogus field and really just represents the above fields in some sort of 32-bit form.

0x13 — Mob Skeleton Piece Track Reference — REFERENCE

Reference points to a 0x12 Mob Skeleton Piece Track fragment.

Fields

Flags : DWORD

Bit 0........ If 1, Params1 exists.
Bit 2........ Usually set to 1.

Params1 : DWORD

Unknown purpose. It's usually set to 1000, but I've also seen it set to 100. My guess is that it might have to do with animation speed.

0x14 — Static or Animated Model Reference/Player Info — PLAIN

Notes

When this fragment is used in a main zone file, the name of the fragment seems to always be PLAYER_1.

Fields

Flags : DWORD

Bit 0........ If 1, Params1 exists.
Bit 1........ If 1, Params2 exists.
Bit 7........ If 0, Fragment2 must contain 0.

Fragment1 : DWORD

This isn't really a fragment reference but a string reference. It points to a "magic" string. When this fragment is used in main zone files the string is "FLYCAMCALLBACK". When used as a placeable object reference, the string is "SPRITECALLBACK". When creating a 0x14 fragment this is currently accomplished by creating a fragment reference, setting the fragment to null, and setting the reference name to the magic string.

Size1 : DWORD

Tells how many entries there are (see below).

Size2 : DWORD

Tells how many Fragment3 entries there are (see below):

Fragment2 : DWORD

Unknown purpose.

Params1 : DWORD

This seems to always contain 0. It seems to only be used in main zone files.

Params2 : 7 DWORDs

These seem to always contain zeroes. They seem to only be used in main zone files.

Entries (there are Size1 of these):

Entry1Size : DWORD

Tells how many Entry1Data DATAPAIRs there are.

Entry1Data : DATAPAIRs

Unknown purpose.

Fragment3: DWORDs

There are Size2 fragment references here. These references can point to several different kinds of fragments. In main zone files, there seems to be only one entry, which points to a 0x09 Camera Reference fragment. When this is instead a static object reference, the entry points to either a 0x2D Mesh Reference fragment. If this is an animated (mob) object reference, it points to a 0x11 Skeleton Track Set Reference fragment. This also has been seen to point to a 0x07 Two-dimensional Object Reference fragment (e.g. coins and blood spots).

Size3 : DWORD

Tells how many bytes are in the Name3 field.

Name3 : BYTEs

An encoded string. It's purpose and possible values are unknown.

0x15 — Object Location — REFERENCE

When used in main zone files, the reference points to a 0x14 Player Info fragment. When used for static (placeable) objects, the reference is a string reference (not a fragment reference) and points to a "magic" string. It typically contains the name of the object with "_ACTORDEF" appended to the end.

Fields

Flags : DWORD

Typically 0x2E when used in main zone files and 0x32E when used for placeable objects.

Fragment1 : DWORD

When used in main zone files, points to a 0x16 fragment. When used for placeable objects, seems to always contain 0. This might be due to the difference in the Flags value.

X : FLOAT

When used in main zone files, contains the minimum X value of the entire zone. When used for placeable objects, contains the X value of the object's location.

Y : FLOAT

When used in main zone files, contains the minimum Y value of the entire zone. When used for placeable objects, contains the Y value of the object's location.

Z : FLOAT

When used in main zone files, contains the minimum Z value of the entire zone. When used for placeable objects, contains the Z value of the object's location.

RotateZ : FLOAT

When used in main zone files, typically contains 0. When used for placeable objects, contains a value describing rotation around the Z axis, scaled as Degrees x (512 / 360).

RotateY : FLOAT

When used in main zone files, typically contains 0. When used for placeable objects, contains a value describing rotation around the Y axis, scaled as Degrees x (512 / 360).

RotateX : FLOAT

When used in main zone files, typically contains 0. When used for placeable objects, contains a value describing rotation around the X axis, scaled as Degrees x (512 / 360).

Params1[3] : FLOAT

Typically contains 0 (though might be more significant for placeable objects).

ScaleY : FLOAT

When used in main zone files, typically contains 0.5. When used for placeable objects, contains the object's scaling factor in the Y direction (e.g. 2.0 would make the object twice as big in the Y direction).

ScaleX : FLOAT

When used in main zone files, typically contains 0.5. When used for placeable objects, contains the object's scaling factor in the X direction (e.g. 2.0 would make the object twice as big in the X direction).

Fragment2 : DWORD

When used in main zone files, typically contains 0 (might be related to the Flags value). When used for placeable objects, points to a 0x33 Vertex Color Reference fragment.

Params2 : DWORD

Typically contains 30 when used in main zone files and 0 when used for placeable objects. This field only exists if Fragment2 points to a fragment.

0x16 — Zone Unknown — PLAIN

Fields

Params1 : FLOAT

Typically contains 0.1. This fragment's purpose is unknown.

0x17 — Polygon Animation? — PLAIN

Fields

Flags : DWORD

Bit 0........ If 0, Params2 must be 1.0.

Size1 : DWORD

Tells how many entries there are (see below).

Size2 : DWORD

Tells how many entries there are (see below).

Params1 : FLOAT

Unknown purpose.

Params2 : FLOAT

Usually 1.

Entries (there are Size1 of these):

Entry1X : FLOAT

Unknown purpose.

Entry1Y : FLOAT

Unknown purpose.

Entry1Z : FLOAT

Unknown purpose.

Entries (there are Size2 of these):

Entry2Size : DWORD

Tells how many DWORDs there are in Entry2Data.

Entry2Data : DWORDs

There are Entry2Size of these. These appear to be indices into the X, Y, Z entries above.

0x18 — Polygon Animation Reference? — REFERENCE

Reference points to a 0x17 Polygon Animation? fragment.

Fields

Flags : DWORD

Bit 0 .... If 1, Params1 exists.

Params1 : FLOAT

Unknown purpose.

0x1B — Light Source — PLAIN

When used in main zone files, the name of this fragment is typically DEFAULT_LIGHTDEF.

Fields

Flags : DWORD

Bit 1 .... Typically 1 when dealing with placed light sources.
Bit 2 .... Typically 1.
Bit 3 .... Typically 1 when dealing with placed light sources. If Bit 4 is 1 then Params3b only exists if this bit is also 1 (not sure about this).
Bit 4 .... If 0, Params3a exists but Params3b and Params4[0..3] don't exist. Otherwise, Params3a doesn't exist but Params3b and Params4[0..3] do exist. This flag seems to determine whether the light is just a simple white light or a light with its own color values.

Params2 : DWORD

Typically contains 1.

Params3a : FLOAT

Typically contains 1.

Params3b : DWORD

Typically contains 200 (attenuation?).

Params4[0] : FLOAT

Typically contains 1.

Params4[1] : FLOAT

Light red component, scaled from 0 (no red component) to 1 (100% red).

Params4[2] : FLOAT

Light green component, scaled from 0 (no green component) to 1 (100% green).

Params4[3] : FLOAT

Light blue component, scaled from 0 (no blue component) to 1 (100% blue).

0x1C — Light Source Reference — REFERENCE

Reference points to a 0x1B Light Source fragment.

Fields:

Flags : DWORD

Typically contains zero.

0x21 — BSP Tree — PLAIN

Fields:

Size1 : DWORD

Entries (there are Size1 of them):

Entry1NormalX : FLOAT

X component of the normal to the split plane.

Entry1NormalY : FLOAT

Y component of the normal to the split plane.

Entry1NormalZ : FLOAT

Z component of the normal to the split plane.

Entry1SplitDistance : FLOAT

Distance from the splitting plane to the origin (0, 0, 0) in x-y-z-space. With the above fields, the splitting plane is represented in Hessian Normal Form.

Entry1RegionID : DWORD

If this is a leaf node, contains the index of the 0x22 BSP Region fragment that this refers to (with the lowest index being 1). Otherwise contains zero.

Entry1Node1 : DWORD

If this is not a leaf node, contains the index of the entry in the tree corresponding to everything in the remaining area on one side of the splitting plane (with the lowest index containing zero). Otherwise contains zero.

Entry1Node2 : DWORD

If this is not a leaf node, contains the index of the entry in the tree corresponding to everything in the remaining area on the other side of the splitting plane (with the lowest index containing zero). Otherwise contains zero.

0x22 — BSP Region — PLAIN

Fields

Flags : DWORD

Typically contains 0x181 for regions that contain polygons and 0x81 for regions that are empty.

Bit 5........ If 1, then the Data6Data consists of WORDs.
Bit 7........ If 1, then the Data6Data consists of BYTEs (the usual).

Fragment1 : DWORD

It is unknown what this references. It usually doesn’t reference anything.

Size1 : DWORD

Tells how many bytes are in the Data1 field.

Size2 : DWORD

Tells how many bytes are in the Data2 field.

Params1 : DWORD

Typically contains zero. It’s purpose is unknown.

Size3 : DWORD

Tells how many Data3 entries there are (usually none).

Size4 : DWORD

Tells how many Data4 entries there are (usually none).

Params2 : DWORD

Typically contains zero. It’s purpose is unknown.

Size5 : DWORD

Tells how many Data5 entries there are (usually only 1).

Size6 : DWORD

Tells how many Data6 entries there are (usually only 1).

Data1 : BYTEs

According to the ZoneConverter source there are 12 * Size1 bytes here. Their format is unknown, for lack of sample data to figure it out.

Data2 : BYTEs

According to the ZoneConverter source there are 8 * Size2 bytes here. Their format is unknown, for lack of sample data to figure it out.

Data3 entries (There are Size3 of these):

Data3Flags : DWORD

Bit 1 .... If 1, then the Data3Params1[0..2] and Data3Params2 fields exist.

Data3Size1 : DWORD

Tells how many Data3Data1 DWORDs there are.

Data3Data1 : DWORDs

There are Data3Size1 DWORDs. Their purpose is unknown.

Data3Params1[0] : DWORD

Unknown purpose. Only exists if Data3Flags Bit 1 is set to 1.

Data3Params1[1] : DWORD

Unknown purpose. Only exists if Data3Flags Bit 1 is set to 1.

Data3Params1[2] : DWORD

Unknown purpose. Only exists if Data3Flags Bit 1 is set to 1.

Data3Params2 : DWORD

Unknown purpose. Only exists if Data3Flags Bit 1 is set to 1.

Data4 entries (There are Size4 of these):

Data4Flags : DWORD

Bit 2 .... If 1, then the Data4Size1 and Data4Data1 fields exist.

Data4Params1 : DWORD

Unknown purpose.

Data4Type : DWORD

This seems to determine whether Data4Params2a and/or Data4Params2b exist.

Data4Params2a : DWORD

Unknown purpose. Only exists if Data4Type is greater than 7.

Data4Params2b : DWORD

Unknown purpose. Only exists if Data4Type is one of the following: 0x0A, 0x0B, 0x0C (though I’m not at all sure about this, due to a lack of sample data).

Data4NameSize : DWORD

Tells the number of bytes in the Data4Name field.

Data4Name : BYTEs

Contains an encoded string. This field is Data4NameSize bytes long.

Data5 entries (There are Size5 of these):

Data5Params1[0] : DWORD

Unknown purpose. Typically contains zero.

Data5Params1[1] : DWORD

Unknown purpose. Typically contains zero.

Data5Params1[2] : DWORD

Unknown purpose. Typically contains zero.

Data5Params2 : DWORD

Unknown purpose. Typically contains zero.

Data5Params3 : DWORD

Unknown purpose. Typically contains 1.

Data5Params4 : DWORD

Unknown purpose. Typically contains zero.

Data5Params5 : DWORD

Unknown purpose. Typically contains zero.

Data6 entries (There are Size6 of these):

Data6Size1 : WORD

Tells the number of entries in the Data6Data field.

Data6Data : Either BYTEs or WORDs

This is a complicated field. It contains run-length-encoded data that tells the client which regions are "nearby". The purpose appears to be so that the client can determine which mobs in the zone have to have their Z coordinates checked, so that they will fall to the ground (or until they land on something). Since it's expensive to do this, it makes sense to only do it for regions that are visible to the player instead of doing it for all mobs in the entire zone (repeatedly).

I've only encountered data where the stream is a list of BYTEs instead of WORDs. The following discussion describes RLE encoding a BYTE stream.

The idea here is to form a sorted list of all region IDs that are within a certain distance, and then write that list as an RLE-encoded stream to save space. The procedure is as follows:

1. Set an initial region ID value to zero.
2. If this region ID is not present in the (sorted) list, skip forward to the first one that is in the list. Write something to the stream that tells it how many IDs were skipped.
3. Form a block of consecutive IDs that are in the list and write something to the stream that tells the client that there are this many IDs that are in the list.
4. If there are more region IDs in the list, go back to step 2.

When writing to the stream, either one or three bytes are written:

0x00..0x3E	skip forward by this many region IDs
0x3F, WORD	skip forward by the amount given in the following 16-bit WORD
0x40..0x7F	skip forward based on bits 3..5, then include the number of IDs based on bits 0..2
0x80..0xBF	include the number of IDs based on bits 3..5, then skip forward based on bits 0..2
0xC0..0xFE	subtracting 0xC0, this many region IDs are nearby
0xFF, WORD	the number of region IDs given by the following WORD are nearby

It should be noted that the values in the range 0x40..0xBF allow skipping and including of no more than seven IDs at a time. Also, they are not necessary to encode a region list: they merely allow better compression.

Size7 : DWORD

Tells how many bytes are in the Name7 field.

Name7 : BYTEs

An encoded string. It's purpose and possible values are unknown.

Fragment2 : DWORD

It is unknown what this references. It usually doesn't reference anything.

Fragment3 : DWORD

If there are any polygons in this region, then this region points to a 0x36 Mesh fragment that contains only those polygons. That mesh must contain all geometry information contained within the volume that this region represents and nothing that lies outside that volume.

0x28 — Light Info — REFERENCE

Reference points to a 0x1C Light Source Reference fragment.

Fields

Flags : DWORD

Typically contains 256 (0x100).

X : FLOAT

X component of the light location.

Y : FLOAT

Y component of the light location.

Z : FLOAT

Z component of the light location.

Radius : FLOAT

Contains the light radius.

0x29 — Region Flag — PLAIN

Notes

This fragment lets you flag certain regions (as defined by 0x22 BSP Region fragments) in a particular way. The flagging is done by setting the name of this fragment to a particular “magic” value. The possible values are:

WT_ZONE	Flag all regions in the list as underwater regions.
LA_ZONE	Flag all regions in the list as lava regions.
DRP_ZONE	Flag all regions in the list as PvP regions.
DRNTP##########_ZONE	Flag all regions in the list as zone point regions. The ####’s are actually numbers and hyphens that somehow tell the client the zone destination. This method of setting zone points may or may not be obsolete.

Fields

Flags : DWORD

Typically contains zero.

Size1 : DWORD

Tells how many region IDs follow.

Regions : DWORDs

There are Size1 DWORDs here. Each isn’t a fragment reference per se, but the ID of a 0x22 BSP region fragment. For example, if there are 100 0x22 BSP Region fragments, then the possible values are in the range 0-99. This constitutes a list of regions that are to be flagged in the particular way.

Size2 : DWORD

Tells how many bytes follow in the Data2 field.

Data2 : BYTEs

An encoded string. An alternate way of using this fragment is to call this fragment Z####_ZONE, where #### is a four-digit number starting with zero. Then Data2 would contain a "magic" string that told the client what was special about the included regions (e.g. WTN__01521000000000000000000000___000000000000). This field is padded with nulls to make it end on a DWORD boundary.

0x2A — Ambient Light — REFERENCE

Reference points to a 0x1C Light Source Reference fragment.

Fields

Flags : DWORD

Typically contains zero.

Size1 : DWORD

Tells how many region IDs follow.

Regions : DWORDs

There are Size1 DWORDs here. Each isn't a fragment reference per se, but the ID of a 0x22 BSP region fragment. For example, if there are 100 0x22 BSP Region fragments, then the possible values are in the range 0-99. This constitutes a list of regions that have the ambient lighting given by the 0x1C fragment that this fragment references.

0x2C — Alternate Mesh — PLAIN

Notes

This fragment is rarely seen. It is very similar to the 0x36 Mesh fragment. I believe that this might have been the original type and was later replaced by the 0x36 Mesh fragment. I've only seen one example of this fragment so far so the information here is uncertain.

Fields

Flags : DWORD

Typically contains 0x00001803. The meaning of the known bits is believed to be as follows:

Bit 0	If 1, then CenterX, CenterY, and CenterZ are valid. Otherwise they must contain zero.
Bit 1	If 1, then Params2 is valid. Otherwise it must contain zero.
Bit 9	If 1, then the Size8 field and Data8 entries exist.
Bit 11	If 1, then the PolygonTexCount field and PolygonTex entries exist.
Bit 12	If 1, then the VertexTexCount field and VertexTex entries exist.
Bit 13	If 1, then the Params3[] fields exist.

VertexCount : DWORD

Tells how many vertices there are in the mesh. Normally this is three times the number of polygons, but this is by no means necessary as polygons can share vertices. However, sharing vertices degrades the ability to use vertex normals to make a mesh look more rounded (with shading).

TexCoordsCount : DWORD

Tells how many texture coordinate pairs there are in the mesh. This should equal the number of vertices in the mesh. Presumably this could contain zero if none of the polygons have textures mapped to them (but why would anyone do that?)

NormalsCount : DWORD

Tells how many vertex normal entries there are in the mesh. This should equal the number of vertices in the mesh. Presumably this could contain zero if vertices should use polygon normals instead, but I haven't tried it (vertex normals are preferable anyway).

Size4 : DWORD

Its purpose is unknown (though if the pattern with the 0x36 fragment holds then it should contain color information).

PolygonsCount : DWORD

Tells how many polygons there are in the mesh.

Size6 : WORD

This seems to only be used when dealing with animated (mob) models. It tells how many entries are in the Data6 area.

VertexPieceCount : WORD

This seems to only be used when dealing with animated (mob) models. It tells how many VertexPiece entries there are. Vertices are grouped together by skeleton piece in this case and VertexPiece entries tell the client how many vertices are in each piece. It’s possible that there could be more pieces in the skeleton than are in the meshes it references. Extra pieces have no polygons or vertices and I suspect they are there to define attachment points for objects (e.g. weapons or shields).

Fragment1 : DWORD

References a 0x31 Texture List fragment. It tells the client which textures this mesh uses. For zone meshes, a single 0x31 fragment should be built that contains all the textures used in the entire zone. For placeable objects, there should be a 0x31 fragment that references only those textures used in that particular object.

Fragment2 : DWORD

Its purpose is unknown.

Fragment3 : DWORD

Its purpose is unknown.

CenterX : FLOAT

This seems to define the center of the model along the X axis and is used for positioning (I think).

CenterY : FLOAT

This is similar to CenterX but references the Y axis.

CenterZ : FLOAT

This is similar to CenterX but references the Z axis.

Params2 : DWORD or FLOAT (not sure)

Its purpose is unknown.

Vertex entries (there are VertexCount of these):

X : FLOAT

X component of the vertex position.

Y : FLOAT

Y component of the vertex position.

Z : FLOAT

Z component of the vertex position.

Texture coordinate entries (there are TexCoordsCount of these)

TX : FLOAT

Contains a 32-bit floating-point texture value ranging from 0 to 1. This represents the horizontal position along a texture bitmap.

TZ : FLOAT

Contains a 32-bit floating-point texture value ranging from 0 to 1. This represents the vertical position along a texture bitmap.

Vertex normal entries (there are NormalsCount of these)

NX : FLOAT

Contains a 32-bit floating-point number representing the X component of the vertex normal.

NY : FLOAT

Contains a 32-bit floating-point number representing the Y component of the vertex normal.

NZ : FLOAT

Contains a 32-bit floating-point number representing the Z component of the vertex normal.

Data4 entries (there are Size4 of these)

Data4Data : DWORD

Its purpose is unknown.

Polygon entries (there are PolygonsCount of these)

PolygonFlag : WORD

Normally contains 0x004B for polygons.

PolygonData : 4 WORDs

Usually contain zero. Their purpose is unknown.

Vertex1 : WORD

Index of the polygon’s first vertex.

Vertex2 : WORD

Index of the polygon’s second vertex.

Vertex3 : WORD

Index of the polygon’s third vertex.

Data6 entries (there are Size9 of these)

Data6Type : DWORD

The purpose of this field is unknown, but it seems to control whether VertexIndex1, VertexIndex2, and Offset exist. It can only contain values in the range 1 to 4. It looks like the Data9 entries are broken up into blocks, where each block is terminated by an entry where Data9Type is 4.

VertexIndex : DWORD

This seems to reference one of the vertex entries. This field only exists if Data6Type contains a value in the range 1 to 3.

Offset : FLOAT

If Data6Type contains 4, then this field exists instead of VertexIndex (this field only exists if Data6Type contains 4). Its purpose is unknown. Data6 entries seem to be sorted by this value.

Data6Param1 : WORD

The purpose of this field is unknown. It seems to only contain values in the range 0 to 2.

Data6Param2 : WORD

The purpose of this field is unknown.

VertexPiece entries (there are VertexPieceCount of these)

VertexCount : WORD

Number of vertices in the skeleton piece.

PieceIndex : WORD

This is the index of the piece according to the 0x10 Skeleton Track Set fragment. The very first piece (index 0) is usually not referenced here as it is usually just a “stem” starting point for the skeleton. Only those pieces referenced here in the mesh should actually be rendered. Any other pieces in the skeleton contain no vertices or polygons and have other purposes.

Size8 : DWORD

Its purpose is unknown. This field only exists if bit 9 of Flags is 1.

Data8 entries (there are Size8 of these)

Data8Data : DWORD

Its purpose is unknown. This field only exists if bit 9 of Flags is 1.

PolygonTexCount : DWORD

Tells how many PolygonTex entries there are. Polygons are grouped together by texture and PolygonTex entries tell the client how many polygons there are that use a particular texture. This field only exists if bit 11 of Flags is 1.

PolygonTex entries (there are PolygonTexCount of these)

PolygonCount : WORD

Number of polygons that use the same texture. All polygon entries are sorted by texture index so that polygons that use the same texture are together. This field only exists if bit 11 of Flags is 1.

TextureIndex : WORD

The index of the texture that the polygons use, according to the 0x31Texture List fragment that this fragment references. This field only exists if bit 11 of Flags is 1.

VertexTexCount : DWORD

Tells how many VertexTex entries there are. Vertices are grouped together by texture and VertexTex entries tell the client how many vertices there are that use a particular texture. This field only exists if bit 12 of Flags is 1.

VertexTex entries (there are VertexTexCount of these)

VertexCount : WORD

Number of vertices that use the same texture. Vertex entries, like polygon entries, are sorted by texture index so that vertices that use the same texture are together. This field only exists if bit 12 of Flags is 1.

TextureIndex : WORD

The index of the texture that the vertices use, according to the 0x31Texture List fragment that this fragment references. This field only exists if bit 12 of Flags is 1.

Params3[0] : DWORD

Its purpose is unknown. This field only exists if bit 13 of Flags is 1.

Params3[1] : DWORD

Its purpose is unknown. This field only exists if bit 13 of Flags is 1.

Params3[2] : DWORD

Its purpose is unknown. This field only exists if bit 13 of Flags is 1.

0x2D — Mesh Reference — REFERENCE

Reference points to either a 0x36 Mesh or 0x2C Alternate Mesh fragment.

Fields

Params1 : DWORD

Apparently must be zero.

0x2F — Mesh Animated Vertices Reference — REFERENCE

Reference points to a 0x37 Mesh Animated Vertices fragment.

Fields

Flags : DWORD

Typically contains zero.

0x30 — Texture — REFERENCE

Reference points to a 0x05 Texture Bitmap Info Reference fragment.

Fields

Flags : DWORD

Bit 1........ Typically 1. If set to 1, then the Pair field exists.

Params1 : DWORD

Bit 0........ Apparently must be 1 if the texture isn't transparent.
Bit 1........ Set to 1 if the texture is masked (e.g. tree leaves).
Bit 2........ Set to 1 if the texture is semi-transparent but not masked.
Bit 3........ Set to 1 if the texture is masked and semi-transparent.
Bit 4........ Set to 1 if the texture is masked but not semi-transparent.
Bit 31...... Apparently must be 1 if the texture isn't transparent.

To make a fully transparent texture, set Params1 to 0.

Params2 : DWORD

Typically contains 0x004E4E4E, but I've also seen 0xB2B2B2. Could this be an RGB reflectivity value?

Params3[0] : FLOAT

Typically contains 0. Its purpose is unknown.

Params3[1] : FLOAT

Typically contains 0 for transparent textures and 0.75 for all others. Its purpose is unknown.

Pair : DATAPAIR

Only exists if Bit 1 of Flags is set. Typically contains 0 in both fields. Its purpose is unknown.

0x31 — Texture List — PLAIN

Fields

Flags : DWORD

Must contain zero.

Size1 : DWORD

Tells how many fragment references this fragment contains.

Fragments : DWORDs

There are Size1 fragment references. Each refers to a 0x30 Texture fragment.

0x32 — Vertex Color — PLAIN

Fields

Data1 : DWORD

Typically contains 1. Its purpose is unknown.

Size1 : DWORD

Tells how many color values are in the VertexColors list. It should be equal to the number of vertices in the placeable object, as contained in its 0x36 Mesh fragment.

Data2 : DWORD

Typically contains 1. Its purpose is unknown.

Data3 : DWORD

Typically contains 200. Its purpose is unknown.

Data4 : DWORD

Typically contains 0. Its purpose is unknown.

VertexColors : DWORDs

This contains an RGBA color value for each vertex in the placeable object. It specifies the additional color to be applied to the vertex, as if that vertex has been illuminated by a nearby light source. The A value isn't fully understood; I believe it represents an alpha as applied to the texture, such that 0 makes the polygon a pure color and 0xFF either illuminates an unaltered texture or mutes the illumination completely. That is, it's either a blending value or an alpha value. Further experimentation is required. 0xD9 seems to be a good (typical) A value for most illuminated vertices.

This field works in exactly the same way as it does in the 0x36 Mesh fragment.

0x33 — Vertex Color Reference — REFERENCE

Reference points to a 0x32 Vertex Color fragment

Fields

Flags : DWORD

Typically contains zero.

0x35 — First Fragment — PLAIN

There are no fields. This fragment's NameReference field should be set to 0xFF000000.

0x36 — Mesh — PLAIN

Notes

This is the fragment most often used for models but another one I've encountered is the 0x2C Alternate Mesh fragment.

Fields

Flags : DWORD

Typically contains 0x00018003 for zone meshes and 0x00014003 for placeable objects. The meaning of the bits is unknown.

Fragment1 : DWORD

References a 0x31 Texture List fragment. It tells the client which textures this mesh uses. For zone meshes, a single 0x31 fragment should be built that contains all the textures used in the entire zone. For placeable objects, there should be a 0x31 fragment that references only those textures used in that particular object.

Fragment2 : DWORD

If this mesh is animated (not character models, but things like swaying flags and trees), this references a 0x2F Mesh Animated Vertices Reference fragment.

Fragment3 : DWORD

It is unknown what this references. It typically doesn't reference anything.

Fragment4 : DWORD

This typically references the very first 0x03 Texture Bitmap Name(s) fragment in the .WLD file. I have no idea why.

CenterX : FLOAT

For zone meshes this typically contains the X coordinate of the center of the mesh. This allows vertex coordinates in the mesh to be relative to the center instead of having absolute coordinates. This is important for preserving precision when encoding vertex coordinate values. For placeable objects this seems to define where the vertices will lie relative to the object's local origin. This seems to allow placeable objects to be created that lie at some distance from their position as given in a 0x15 Object Location fragment (why one would do this is a mystery, though).

CenterY : FLOAT

This is similar to CenterX but references the Y axis.

CenterZ : FLOAT

This is similar to CenterX but references the Z axis.

Params2[0] : DWORD

Typically contains zero. Its purpose is unknown.

Params2[1] : DWORD

Typically contains zero. Its purpose is unknown.

Params2[2] : DWORD

Typically contains zero. Its purpose is unknown.

MaxDist : FLOAT

Given the values in CenterX, CenterY, and CenterZ, this seems to contain the maximum distance between any vertex and that position. It seems to define a radius from that position within which the mesh lies.

MinX : FLOAT

Seems to contain the minimum X coordinate of any vertex in the mesh, in absolute coordinates.

MinY : FLOAT

Seems to contain the minimum Y coordinate of any vertex in the mesh, in absolute coordinates.

MinZ : FLOAT

Seems to contain the minimum Z coordinate of any vertex in the mesh, in absolute coordinates.

MaxX : FLOAT

Seems to contain the maximum X coordinate of any vertex in the mesh, in absolute coordinates.

MaxY : FLOAT

Seems to contain the maximum Y coordinate of any vertex in the mesh, in absolute coordinates.

MaxZ : FLOAT

Seems to contain the maximum Y coordinate of any vertex in the mesh, in absolute coordinates.

VertexCount : WORD

Tells how many vertices there are in the mesh. Normally this is three times the number of polygons, but this is by no means necessary as polygons can share vertices. However, sharing vertices degrades the ability to use vertex normals to make a mesh look more rounded (with shading).

TexCoordsCount : WORD

Tells how many texture coordinate pairs there are in the mesh. This should equal the number of vertices in the mesh. Presumably this could contain zero if none of the polygons have textures mapped to them (but why would anyone do that?)

NormalsCount : WORD

Tells how many vertex normal entries there are in the mesh. This should equal the number of vertices in the mesh. Presumably this could contain zero if vertices should use polygon normals instead, but I haven’t tried it (vertex normals are preferable anyway).

ColorCount : WORD

Tells how many vertex color entries are in the mesh. This should equal the number of vertices in the mesh, or zero if there are no vertex color entries. Meshes do not require color entries to work. Color entries are used for illuminating polygons when there is a nearby light source.

PolygonsCount : WORD

Tells how many polygons there are in the mesh.

VertexPieceCount : WORD

This seems to only be used when dealing with animated (mob) models. It tells how many VertexPiece entries there are. Vertices are grouped together by skeleton piece in this case and VertexPiece entries tell the client how many vertices are in each piece. It’s possible that there could be more pieces in the skeleton than are in the meshes it references. Extra pieces have no polygons or vertices and I suspect they are there to define attachment points for objects (e.g. weapons or shields).

PolygonTexCount : WORD

Tells how many PolygonTex entries there are. Polygons are grouped together by texture and PolygonTex entries tell the client how many polygons there are that use a particular texture.

VertexTexCount : WORD

Tells how many VertexTex entries there are. Vertices are grouped together by texture and VertexTex entries tell the client how many vertices there are that use a particular texture.

Size9 : WORD

This seems to only be used when dealing with animated (mob) models. It tells how many entries are in the Data9 area.

Scale : WORD

This allows vertex coordinates to be stored as integral values instead of floating-point values, without losing precision based on mesh size. Vertex values are multiplied by (1 shl Scale) and stored in the vertex entries.

Vertex entries (there are VertexCount of these):

X : SIGNED WORD (signed 16-bit value)

X component of the vertex position, multiplied by (1 shl Scale).

Y : SIGNED WORD (signed 16-bit value)

Y component of the vertex position, multiplied by (1 shl Scale).

Z : SIGNED WORD (signed 16-bit value)

Z component of the vertex position, multiplied by (1 shl Scale).

Texture coordinate entries (there are TexCoordsCount of these)

TX : SIGNED WORD (old-format file) or SIGNED DWORD (new-format file)

In old-format .WLD files, this contains a signed 16-bit texture value in pixels (most textures are 256 pixels in size). In new-format .WLD files this is a signed 32-bit value instead.

TZ : SIGNED WORD (old-format file) or SIGNED DWORD (new-format file)

In old-format .WLD files, this contains a signed 16-bit texture value in pixels (most textures are 256 pixels in size). In new-format .WLD files this is a signed 32-bit value instead.

Vertex normal entries (there are NormalsCount of these)

NX : SIGNED BYTE

Contains a signed byte representing the X component of the vertex normal, scaled such that –127 represents –1 and 127 represents 1.

NY : SIGNED BYTE

Contains a signed byte representing the Y component of the vertex normal, scaled such that –127 represents –1 and 127 represents 1.

NZ : SIGNED BYTE

Contains a signed byte representing the Z component of the vertex normal, scaled such that –127 represents –1 and 127 represents 1.

Vertex color entries (there are ColorCount of these)

Color : DWORD

This contains an RGBA color value for each vertex in the mesh. It specifies the additional color to be applied to the vertex, as if that vertex has been illuminated by a nearby light source. The A value isn’t fully understood; I believe it represents an alpha as applied to the texture, such that 0 makes the polygon a pure color and 0xFF either illuminates an unaltered texture or mutes the illumination completely. That is, it’s either a blending value or an alpha value. Further experimentation is required. 0xD9 seems to be a good (typical) A value for most illuminated vertices.

Polygon entries (there are PolygonsCount of these)

PolygonFlag : WORD

Normally contains zero for polygons but contains 0x0010 for polygons that the player can pass through (like water and tree leaves).

Vertex1 : WORD

Index of the polygon’s first vertex.

Vertex2 : WORD

Index of the polygon’s second vertex.

Vertex3 : WORD

Index of the polygon’s third vertex.

VertexPiece entries (there are VertexPieceCount of these)

VertexCount : WORD

Number of vertices in the skeleton piece.

PieceIndex : WORD

This is the index of the piece according to the 0x10 Skeleton Track Set fragment. The very first piece (index 0) is usually not referenced here as it is usually just a “stem” starting point for the skeleton. Only those pieces referenced here in the mesh should actually be rendered. Any other pieces in the skeleton contain no vertices or polygons and have other purposes.

PolygonTex entries (there are PolygonTexCount of these)

PolygonCount : WORD

Number of polygons that use the same texture. All polygon entries are sorted by texture index so that polygons that use the same texture are together.

TextureIndex : WORD

The index of the texture that the polygons use, according to the 0x31Texture List fragment that this fragment references.

VertexTex entries (there are VertexTexCount of these)

VertexCount : WORD

Number of vertices that use the same texture. Vertex entries, like polygon entries, are sorted by texture index so that vertices that use the same texture are together.

TextureIndex : WORD

The index of the texture that the vertices use, according to the 0x31Texture List fragment that this fragment references.

Data9 entries (there are Size9 of these)

VertexIndex1 : WORD

This seems to reference one of the vertex entries. This field only exists if Data9Type contains a value in the range 1 to 3.

VertexIndex2 : WORD

This seems to reference one of the vertex entries. This field is only valid if Data9Type contains 1. Otherwise, this field must contain zero.

Offset : FLOAT

If Data9Type contains 4, then this field exists instead of VertexIndex1 and VertexIndex2 (this field only exists if Data9Type contains 4). Its purpose is unknown. Data9 entries seem to be sorted by this value.

Data9Param1 : WORD

The purpose of this field is unknown. It seems to only contain values in the range 0 to 2.

Data9Type : WORD

The purpose of this field is unknown, but it seems to control whether VertexIndex1, VertexIndex2, and Offset exist. It can only contain values in the range 1 to 4. It looks like the Data9 entries are broken up into blocks, where each block is terminated by an entry where Data9Type is 4.

0x37 — Mesh Animated Vertices — PLAIN

Notes

This fragment contains sets of vertex values to be substituted for the vertex values in a 0x36 Mesh fragment if that mesh is animated. For example, if a mesh has 50 vertices then this fragment will have one or more sets of 50 vertices, one set for each animation frame. The vertex values in this fragment will then be used instead of the vertex values in the 0x36 Mesh fragment as the client cycles through the animation frames.

Fields

Flags : DWORD

Typically contains zero. Its purpose is unknown.

VertexCount : WORD

Should be equal to the number of vertices in the mesh, as contained in its 0x36 Mesh fragment.

FrameCount : WORD

The number of animation frames.

Param1 : WORD

Typically contains 100. Its purpose is unknown.

Param2 : WORD

Typically contains zero. Its purpose is unknown.

Scale : WORD

This works in exactly the same way as the Scale field in the 0x36 Mesh fragment. By dividing the vertex values by (1 shl Scale), real vertex values are created.

Frame entries (there are FrameCount of these):

Vertex entries (there are VertexCount of these):

X : SIGNED WORD (signed 16-bit value)

X component of the vertex position, multiplied by (1 shl Scale).

Y : SIGNED WORD (signed 16-bit value)

Y component of the vertex position, multiplied by (1 shl Scale).

Z : SIGNED WORD (signed 16-bit value)

Z component of the vertex position, multiplied by (1 shl Scale).

Size6 : WORD

Typically contains zero. Its purpose is unknown.