Microsoft DirectX File Format Specification
Version 1.13

January 13, 1997

Disclaimer

Contents

1 Introduction

This document specifies the file format introduced with DirectX 2. A binary version of this format was subsequently released with DirectX 3 and this is also detailed in this reference.

The DirectX File Format provides a rich template-driven file format that enables the storage of meshes, textures, animations and user-definable objects. Support for animation sets allows predefined paths to be stored for playback in real time. Also supported are instancing which allows multiple references to an object, such as a mesh, while storing its data only once per file, and hierarchies to express relationships between data records. The DirectX file format is used natively by the Direct3D Retained Mode API, providing support for reading predefined objects into an application or writing mesh information constructed by the application in real time.

The File Format provides low-level data primitives, upon which applications define higher level primitives via templates. This is the method by which Direct3D defines higher level primitives such as Vectors, Meshes, Matrices and Colors.

This document details the API independent features of the file format and also the method by which Direct3D Retained Mode uses the file format.

---

intro

2 File Format Architecture

The DirectX file format is an architecture- and context-free file format. It is template driven and is free of any usage knowledge. The file format may be used by any client application and currently is used by Direct3D Retained Mode to describe geometry data, frame hierarchies and animations.

The rest of this section will deal with the content and syntax of the file format. The file format uses the extension **.x** when used with the DirectX SDK.

Reserved Words

The following words are reserved and must not be used:

ARRAY
BYTE
CHAR
CSTRING
DOUBLE
DWORD
FLOAT
STRING
TEMPLATE
UCHAR
UNICODE
WORD

Header

The variable length header is compulsory and must be at the beginning of the data stream. The header contains the following

Type	*Sub Type*	*Size*	*Contents*	*Content Meaning*
Magic Number - required		4 bytes	"xof "	
Version Number - required	Major Number	2 bytes	03	Major version 3
	Minor Number	2 bytes	02	Minor version 2
Format Type - required		4	"txt	Text File

architect
reserved
header

	bytes	" "bin" "com"	Binary File Compressed File
Compression Type - required ***if format type is compressed***	4 bytes	"lzw" "zip" etc…	
Float size - required	4 bytes	0064 0032	64 bit floats 32 bit floats

Example

```
xof 0302txt 0064
```

Comments

Comments are only applicable in text files. Comments may occur anywhere in the data stream. A comment begins with either C++ style double-slashes "//", or a hash character "#". The comment runs to the next new-line.

Example

```
# This is a comment.
// This is another comment.
```

Templates

Templates define how the data stream is interpreted – the data is modulated by the template definition. A template has the following form:

```
template <template-name> {
<UUID>
<member 1>;
…
```

comments
templates

```
<member n>;
[restrictions]
}
```

Template name

This is an alphanumeric name which may include the underscore character '_'. It must not begin with a digit.

UUID

A universally unique identifier formatted to the OSF DCE standard and surrounded by angle brackets '<' and '>'. For example:

<3D82AB43-62DA-11cf-AB39-0020AF71E433>

Members

Template members consist of a named data type followed by an optional name or an array of a named data type. Valid primitive data types are

Type	*Size*
WORD	16 bits
DWORD	32 bits
FLOAT	IEEE float
DOUBLE	64 bits
CHAR	8 bits
UCHAR	8 bits
BYTE	8 bits
STRING	NULL terminated string
CSTRING	Formatted C-string (currently unsupported)
UNICODE	UNICODE string (currently unsupported)

Additional data types defined by templates encountered earlier in the data stream may also be referenced within a template definition. No forward references are allowed.

Any valid data type can be expressed as an array in the template definition. The basic syntax is as follows

```
array <data-type> <name>[<dimension-size>];
```

Where <dimension-size> can either be an integer or a named reference to another template member whose value is then substituted.

Arrays may be n-dimensional where n is determined by the number of paired square brackets trailing the statement. For example

```
array DWORD FixedHerd[24];
array DWORD Herd[nCows];
array FLOAT Matrix4x4[4][4];
```

Restrictions

Templates may be *open, closed* or *restricted.* These restrictions determine which data types may appear in the immediate hierarchy of a data object defined by the template. An open template has no restrictions, a closed template rejects all data types and a restricted template allows a named list of data types. The syntax is as follows

Three periods enclosed by square brackets indicate an open template

```
[ … ]
```

A comma separated list of named data types followed optionally by their uuids enclosed by square brackets indicates a restricted template

```
[ { data-type [ UUID ] , }… ]
```

The absence of either of the above indicates a closed template.

Examples

```
template Mesh {
<3D82AB44-62DA-11cf-AB39-0020AF71E433>
DWORD nVertices;
array Vector vertices[nVertices];
DWORD nFaces;
array MeshFace faces[nFaces];
 [ ... ]              // An open template
}
template Vector {
<3D82AB5E-62DA-11cf-AB39-0020AF71E433>
FLOAT x;
FLOAT y;
FLOAT z;
}                     // A closed template
template FileSystem {
<UUID>
STRING name;
[ Directory <UUID>, File <UUID> ] // A restricted template
}
```

There is one special template – the *Header* template. It is recommended that each application define such a template and use it to define application specific information such as version information. If present, this header will be read by the File Format API and if a *flags* member is available, it will be used to determine how the following data is interpreted. The *flags* member, if defined, should be a DWORD. One bit is currently defined – bit 0. If this is clear, the following data in the file is binary, if set, the following data is text. Multiple header data objects can be used to switch between binary and text during the file.

Data

Data objects contain the actual data or a reference to that data. Each has a corresponding template that specifies the data type.

Data objects have the following form

```
<Identifier> [name] {
<member 1>;
…
<member n>;
}
```

Identifier

This is compulsory and must match a previously defined data type or primitive.

Name

This is optional. (See above for syntax definition.)

Members

Data members can be one of the following

Data object

A nested data object. This allows the hierarchical nature of the file format to be expressed. The types of nested data objects allowed in the hierarchy may be restricted. See *Templates* above.

Data reference

A reference to a previously encountered data object. The syntax is as follows

```
{ name }
```

Integer list

A semicolon separated list of integers. For example

```
1; 2; 3;
```

data

Float list

A semicolon separated list of floats. For example

```
1.0; 2.0; 3.0;
```

String list

A semicolon separated list of strings. For example

```
"Moose"; "Goats"; "Sheep";
```

Use of commas and semicolons

This is perhaps the most complex syntax issue in the file format, but is very strict: Commas are used to separate array members; semicolons terminate every data item.

For example, if we have a template defined as

```
template foo {
DWORD bar;
}
```

then an instance of this would look like

```
foo dataFoo {
1;
}
```

Next, we have a template containing another template

```
template foo {
DWORD bar;
DWORD bar2;
}
template container {
FLOAT aFloat;
foo aFoo;
}
```

then an instance of this would look like

```
container dataContainer {
1.1;
2; 3;;
}
```

Note that the second line that represents the **foo** inside container has two semi-colons at the end of the line. The first indicates the end of the data item **aFoo** (inside container), and the second indicates the end of the **container**.

Next we consider arrays.

```
Template foo {
array DWORD bar[3];
}
```

then an instance of this would look like

```
foo aFoo {
1, 2, 3;
}
```

In the array case, there is no need for the data items to be separated by a semi-colon as they are delineated by a comma. The semi-colon at the end marks the end of the array.

Now consider a template that contains an array of data items defined by a template

```
template foo {
DWORD bar;
DWORD bar2;
}
template container {
DWORD count;
array foo fooArray[count];
}
```

then an instance of this would look like

```
container aContainer {
3;
1;2;,3;4;,5;6;;
}
```

Appendix A

Templates Used by Direct3D Retained Mode

This section details the templates used by the Direct3D Retained Mode API. A familiarity with Direct3D Retained mode data types is assumed.

	Template Name	***UUID***	
	Header	<3D82AB43-62DA-11cf-AB39-0020AF71E433>	
Member Name	***Type***	***Optional Array Size***	***Optional Data Objects***

A

major minor flags	WORD WORD DWORD		None
Description			
This template defines the application specific header for the Direct3D Retained mode usage of the DirectX File Format. The retained mode uses the major and minor flags to specify the current major and minor versions for the retained mode file format.			

	Template Name	***UUID***	
	Vector	<3D82AB5E-62DA-11cf-AB39-0020AF71E433>	
Member Name	***Type***	***Optional Array Size***	***Optional Data Objects***
x y z	FLOAT FLOAT FLOAT		None
Description			
This template defines a vector.			

	Template Name	***UUID***	
	Coords2d	<F6F23F44-7686-11cf-8F52-0040333594A3>	
Member Name	***Type***	***Optional Array Size***	***Optional Data Objects***
u v	FLOAT FLOAT		None
Description			
A two dimensional vector used to define a mesh's texture coordinates.			

	Template Name	***UUID***	
	Quaternion	<10DD46A3-775B-11cf-8F52-0040333594A3>	
Member Name	***Type***	***Optional Array Size***	***Optional Data Objects***
s	FLOAT		None
v	Vector		
Description			
Currently unused.			

	Template Name	***UUID***	
	Matrix4x4	<F6F23F45-7686-11cf-8F52-0040333594A3>	
Member Name	***Type***	***Optional Array Size***	***Optional Data Objects***
matrix	array FLOAT	16	None
Description			
This template defines a 4 by 4 matrix. This is used as a frame transformation matrix.			

	Template Name	***UUID***	
	ColorRGBA	<35FF44E0-6C7C-11cf-8F52-0040333594A3>	
Member Name	***Type***	***Optional Array Size***	***Optional Data Objects***
red	FLOAT		None
green	FLOAT		
blue	FLOAT		
alpha	FLOAT		

Description	
This template defines a color object with an alpha component. This is used for the face color in the material template definition.	

	Template Name	***UUID***	
	ColorRGB	<D3E16E81-7835-11cf-8F52-0040333594A3>	
Member Name	***Type***	***Optional Array Size***	***Optional Data Objects***
red green blue	FLOAT FLOAT FLOAT		None
Description			
This template defines the basic RGB color object.			

	Template Name	***UUID***
	Indexed Color	<1630B820-7842-11cf-8F52-0040333594A3>
Member Name	***Type***	***Optional Array Size***
index ColorRGBA	DWORD indexColor	
Description		
This template consists of an index parameter and a RGBA color and is used in for defining mesh vertex colors. The index defines the vertex to which the color is applied.		

Template Name	***UUID***
Boolean	<4885AE61-78E8-11cf-8F52-0040333594A3>

Member Name	*Type*	*Optional Array Size*	*Optional Data Objects*
WORD	truefalse		None
Description			
Defines a simple boolean type. Should be set to 0 or 1.			

	Template Name	*UUID*
	Boolean2d	<4885AE63-78E8-11cf-8F52-0040333594A3>

Member Name	*Type*	*Optional Array Size*	*Optional Data Objects*
u v	Boolean Boolean		None
Description			
This defines a set of 2 boolean values used in the MeshFaceWraps template in order to define the texture topology of an individual face.			

	Template Name	*UUID*
	Material	<3D82AB4D-62DA-11cf-AB39-0020AF71E433>

Member Name	*Type*	*Optional Array Size*	*Optional Data Objects*
faceColor power specularColor emissiveColor	ColorRGBA FLOAT ColorRGB ColorRGB		Any
Description			
This template defines a basic material color which can be applied to either a complete mesh or a mesh's individual faces. The power is the specular exponent of the			

material. Note that the ambient color requires an alpha component.

Option Data Objects used by Direct3DRM	
TextureFilename	Is this is not present, the face is untextured.

Template Name	***UUID***
TextureFilename	<A42790E1-7810-11cf-8F52-0040333594A3>

Member Name	***Type***	***Optional Array Size***	***Optional Data Objects***
filename	STRING		None

Description
This template allows you to specify the filename of a texture to apply to a mesh or a face. This should appear within a material object.

Template Name	***UUID***
MeshFace	<3D82AB5F-62DA-11cf-AB39-0020AF71E433>

Member Name	***Type***	***Optional Array Size***	***Optional Data Objects***
nFaceVertexIndices	DWORD		None
faceVertexIndices	array DWORD	nFaceVertexIndicies	

Description
This template is used by the Mesh template to define a mesh's faces. Each element of the nFaceVertexIndices array references a mesh vertex used to build the face.

Template Name	***UUID***
MeshFaceWraps	<4885AE62-78E8-11cf-8F52-0040333594A3>

Member Name	*Type*	*Optional Array Size*	*Optional Data Objects*
nFaceWrapValues	DWORD		None
faceWrapValues	Boolean2d		
Description			
This template is used to define the texture topology of each face in a wrap. nFaceWrapValues should be equal to the number of faces in a mesh.			

	Template Name	*UUID*	
	MeshTextureCoords	<F6F23F40-7686-11cf-8F52-0040333594A3>	
Member Name	***Type***	***Optional Array Size***	***Optional Data Objects***
nTextureCoords	DWORD		None
textureCoords	array Coords2d	nTextureCoords	
Description			
This template defines a mesh's texture coordinates.			

	Template Name	*UUID*	
	MeshNormals	<F6F23F43-7686-11cf-8F52-0040333594A3>	
Member Name	***Type***	***Optional Array Size***	***Optional Data Objects***
nNormals	DWORD		None
normals	array Vector	nNormals	
nFaceNormals	DWORD		
faceNormals	array MeshFace	nFaceNormals	
Description			

This template defines normals for a mesh. The first array of vectors are the normal vectors themselves, and the second array is an array of indexes specifying which normals should be applied to a given face. nFaceNormals should be equal to the number of faces in a mesh.

Template Name	***UUID***
MeshVertexColors	<1630B821-7842-11cf-8F52-0040333594A3>

Member Name	***Type***	***Optional Array Size***	***Optional Data Objects***
nVertexColors	DWORD		None
vertexColors	array IndexedColor	nVertexColors	

Description

This template specifies vertex colors for a mesh, as opposed to applying a material per face or per mesh.

Template Name	***UUID***
MeshMaterialList	<F6F23F42-7686-11cf-8F52-0040333594A3>

Member Name	***Type***	***Optional Array Size***	***Optional Data Objects***
nMaterials	DWORD		Material
nFaceIndexes	DWORD		
FaceIndexes	array DWORD	nFaceIndexes	

Description

This template is used in a mesh object to specify which material applies to which faces. nMaterials specifies how many materials are present, and materials specify which material to apply.

Template Name	***UUID***

	Mesh	<3D82AB44-62DA-11cf-AB39-0020AF71E433>	
Member Name	***Type***	***Optional Array Size***	***Optional Data Objects***
nVertices	DWORD		Any
vertices	array Vector	nVertices	
nFaces	DWORD		
faces	array MeshFace	nFaces	
Description			
This template defines a simple mesh. The first array is a list of vertices and the second array defines the faces of the mesh by indexing into the vertex array.			
Option Data Objects used by Direct3DRM			
MeshFaceWraps	Is this is not present, wrapping for both u and v defaults to false.		
MeshTextureCoords	Is this is not present, there are no texture coordinates.		
MeshNormals	Is this is not present, normals are generated using the GenerateNormals() member of the API.		
MeshVertexColors	Is this is not present, the colors default to white.		
MeshMaterialList	Is this is not present, the material defaults to white.		

	Template Name	***UUID***	
	FrameTransformMatrix	<F6F23F41-7686-11cf-8F52-0040333594A3>	
Member Name	***Type***	***Optional Array Size***	***Optional Data Objects***
frameMatrix	Matrix4x4		None
Description			
This template defines a local transform for a frame (and all its child objects).			

	Template Name	***UUID***	
	Frame	<3D82AB46-62DA-11cf-AB39-0020AF71E433>	
Member Name	***Type***	***Optional Array Size***	***Optional Data Objects***
none			Any
Description			
This template defines a frame. Currently the frame can contain objects of the type Mesh and a FrameTransformMatrix.			
Option Data Objects used by Direct3DRM			
FrameTransformMatrix	Is this is not present, no local transform is applied to the frame.		
Mesh	Any number of mesh objects that become children of the frame. These can be specified inlined or by reference.		

	Template Name	***UUID***	
	FloatKeys	<10DD46A9-775B-11cf-8F52-0040333594A3>	
Member Name	***Type***	***Optional Array Size***	***Optional Data Objects***
nValues values	DWORD array FLOAT	 nValues	None
Description			
This template defines an array of floats and the number of floats in that array. This is used for defining sets of animation keys.			

	Template Name	***UUID***	
	TimedFloatKeys	<F406B180-7B3B-11cf-8F52-0040333594A3>	
Member Name	***Type***	***Optional Array Size***	***Optional Data Objects***

time tfkeys	DWORD FloatKeys		None
Description			
This template defines a set of floats and a positive time used in animations.			

	Template Name	***UUID***	
	AnimationKey	<10DD46A8-775B-11cf-8F52-0040333594A3>	
Member Name	***Type***	***Optional Array Size***	***Optional Data Objects***
keyType nKeys keys	DWORD DWORD array TimedFloatKeys	 nKeys	None
Description			
This template defines a set of animation keys. The keyType parameter specifies whether the keys are rotation, scale or position keys (using the integers 0, 1 or 2 respectively).			

	Template Name	***UUID***	
	AnimationOptions	<E2BF56C0-840F-11cf-8F52-0040333594A3>	
Member Name	***Type***	***Optional Array Size***	***Optional Data Objects***
openclosed positionquality	DWORD DWORD		None
Description			
This template allows you to set the D3DRM Animation options. The openclosed parameter can be either 0 for a closed or 1 for an open animation. The positionquality parameter is used to set the position quality for any position keys specified and can			

either be 0 for spline positions or 1 for linear positions. By default an animation is open and uses linear position keys.

	Template Name	***UUID***	
	Animation	<3D82AB4F-62DA-11cf-AB39-0020AF71E433>	
Member Name	***Type***	***Optional Array Size***	***Optional Data Objects***
none			any
Description			
This template contains animations referencing a previous frame. It should contain one reference to a frame and at least one set of AnimationKeys. It can also contain an AnimationOptions data object.			
Option Data Objects used by Direct3DRM			
AnimationKey	An animation is meaningless without AnimationKeys.		
AnimationOptions	If this is not present, then an animation is open and uses linear position keys.		

	Template Name	***UUID***	
	AnimationSet	<3D82AB50-62DA-11cf-AB39-0020AF71E433>	
Member Name	***Type***	***Optional Array Size***	***Optional Data Objects***
none			Animation
Description			
An AnimationSet contains one or more Animation objects and is the equivalent to the D3D Retained Mode concept of Animation Sets. This means each animation within an animation set has the same time at any given point. Increasing the animation set's time will increase the time for all the animations it contains.			

Appendix B

In this appendix we will describe two cubes – one simple and nontextured, and the other textured.

A Simple Cube

This file defines a simple cube that has four red and two green sides. Notice in this file that optional information is being used to add information to the data object defined by the Mesh template.

```
Material RedMaterial {
1.000000;0.000000;0.000000;1.000000;;   // R = 1.0, G = 0.0, B = 0.0
0.000000;
0.000000;0.000000;0.000000;;
0.000000;0.000000;0.000000;;
}
Material GreenMaterial {
0.000000;1.000000;0.000000;1.000000;;   // R = 0.0, G = 1.0, B = 0.0
0.000000;
0.000000;0.000000;0.000000;;
0.000000;0.000000;0.000000;;
}
// Define a mesh with 8 vertices and 12 faces (triangles). Use
// optional data objects in the mesh to specify materials, normals
// and texture coordinates.
Mesh CubeMesh {
8;          // 8 vertices
1.000000;1.000000;-1.000000;,           // vertex 0
-1.000000;1.000000;-1.000000;,    // vertex 1
-1.000000;1.000000;1.000000;,           // etc…
1.000000;1.000000;1.000000;,
1.000000;-1.000000;-1.000000;,
-1.000000;-1.000000;-1.000000;,
-1.000000;-1.000000;1.000000;,
1.000000;-1.000000;1.000000;;

12;             // 12 faces
3;0,1,2;,                   // face 0 has 3 vertices
3;0,2,3;,                   // etc…
3;0,4,5;,
3;0,5,1;,
3;1,5,6;,
3;1,6,2;,
3;2,6,7;,
3;2,7,3;,
3;3,7,4;,
3;3,4,0;,
```

B
cube

```
3;4,7,6;,
3;4,6,5;;

// All required data has been defined. Now define optional data
// using the hierarchical nature of the file format.
MeshMaterialList {
   2;                 // Number of materials used
12;         // A material for each face
0,       // face 0 uses the first
0,       // material
0,
0,
0,
0,
0,
0,
1,       // face 8 uses the second
1,       // material
1,
1;;
{RedMaterial}              // References to the defini-
{GreenMaterial}            // tions of material 0 and 1
}
MeshNormals {
8;       // define 8 normals
0.333333;0.666667;-0.666667;,
-0.816497;0.408248;-0.408248;,
-0.333333;0.666667;0.666667;,
0.816497;0.408248;0.408248;,
0.666667;-0.666667;-0.333333;,
-0.408248;-0.408248;-0.816497;,
-0.666667;-0.666667;0.333333;,
0.408248;-0.408248;0.816497;;
12;         // For the 12 faces,
3;0,1,2;,            // define the normals
3;0,2,3;,
3;0,4,5;,
3;0,5,1;,
3;1,5,6;,
3;1,6,2;,
3;2,6,7;,
3;2,7,3;,
3;3,7,4;,
3;3,4,0;,
3;4,7,6;,
3;4,6,5;;
}
MeshTextureCoords {
8;       // Define texture coords
0.000000;1.000000;         // for each of the vertices
1.000000;1.000000;
0.000000;1.000000;
1.000000;1.000000;
```

```
0.000000;0.000000;
1.000000;0.000000;
0.000000;0.000000;
1.000000;0.000000;;
}
}
```

Adding Textures

In order to add textures, we make use of the hierarchical nature of the file format and add an optional **TextureFilename** data object to the **Material** data objects. So, the Material objects now read

```
Material RedMaterial {
1.000000;0.000000;0.000000;1.000000;;   // R = 1.0, G = 0.0, B = 0.0
0.000000;
0.000000;0.000000;0.000000;;
0.000000;0.000000;0.000000;;
TextureFilename {
"tex1.ppm";
}
}
Material GreenMaterial {
0.000000;1.000000;0.000000;1.000000;;   // R = 0.0, G = 1.0, B = 0.0
0.000000;
0.000000;0.000000;0.000000;;
0.000000;0.000000;0.000000;;
TextureFilename {
"win95.ppm";
}
}
```

Frames and Animations

Frames

A frame is expected to take the following structure

```
Frame Aframe {        // The frame name is chosen for convenience.
FrameTransformMatrix {
…transform data…
}
[ Meshes ] and/or [ More frames]
}
```

---

textures

frames

So, what we're going to do is place the cube mesh we defined earlier inside a frame with an identity transform. We're then going to apply an animation to this frame.

```
Frame CubeFrame {
FrameTransformMatrix {
1.000000, 0.000000, 0.000000, 0.000000,
0.000000, 1.000000, 0.000000, 0.000000,
0.000000, 0.000000, 1.000000, 0.000000,
0.000000, 0.000000, 0.000000, 1.000000;;
}
{CubeMesh}      // We could have the mesh inline, but we'll
      // use an object reference instead.
}
```

AnimationSets and Animations

Animations and AnimationSets in the file format map directly to Direct3D Retained Mode's animation concepts.

```
Animation Animation0 {            // The name is chosen for convenience.
{ Frame that it applies to - normally a reference }
AnimationKey {
…animation key data…
}
{ …more animation keys… }
}
```

Animations are then grouped into AnimationSets:

```
AnimationSet AnimationSet0 {// The name is chosen for convenience.

 { an animation - could be inline or a reference }

 { … more animations … }

}
```

So, what we'll do now is take the cube through an animation

```
AnimationSet AnimationSet0 {
Animation Animation0 {
{CubeFrame} // Use the frame containing the cube
AnimationKey {
2;    // Position keys
9;    // 9 keys
10; 3; -100.000000, 0.000000, 0.000000;;,
20; 3; -75.000000, 0.000000, 0.000000;;,
30; 3; -50.000000, 0.000000, 0.000000;;,
40; 3; -25.500000, 0.000000, 0.000000;;,
50; 3; 0.000000, 0.000000, 0.000000;;,
60; 3; 25.500000, 0.000000, 0.000000;;,
```

```
70; 3; 50.000000, 0.000000, 0.000000;;,
80; 3; 75.500000, 0.000000, 0.000000;;,
90; 3; 100.000000, 0.000000, 0.000000;;;
}
}
}
```

Appendix C

Binary Format Specification

This section details the binary version of the DirectX File Format as introduced with the release of DirectX 3. This appendix should be read in conjunction with the section entitled File Format Architecture above.

The binary format is a tokenized representation of the text format. Tokens may be stand-alone or accompanied by primitive data records. Stand-alone tokens give grammatical structure and record-bearing tokens supply the necessary data.

Note that all data is stored in little endian format.

A valid binary data stream consists of a header followed by templates and/or data objects.

Header

The following definitions should be used when reading and writing the binary header directly. Note that compressed data streams are not currently supported and are therefore not detailed here.

```
#define XOFFILE_FORMAT_MAGIC \
  ((long)'x' + ((long)'o' << 8) + ((long)'f' << 16) + ((long)' ' << 24))

#define XOFFILE_FORMAT_VERSION \
  ((long)'0' + ((long)'3' << 8) + ((long)'0' << 16) + ((long)'2' << 24))

#define XOFFILE_FORMAT_BINARY \
  ((long)'b' + ((long)'i' << 8) + ((long)'n' << 16) + ((long)' ' << 24))

#define XOFFILE_FORMAT_TEXT   \
  ((long)'t' + ((long)'x' << 8) + ((long)'t' << 16) + ((long)' ' << 24))

#define XOFFILE_FORMAT_COMPRESSED \
  ((long)'c' + ((long)'m' << 8) + ((long)'p' << 16) + ((long)' ' << 24))

#define XOFFILE_FORMAT_FLOAT_BITS_32 \
```

C
binary
head

```
  ((long)'0' + ((long)'0' << 8) + ((long)'3' << 16) + ((long)'2' << 24))

#define XOFFILE_FORMAT_FLOAT_BITS_64 \
  ((long)'0' + ((long)'0' << 8) + ((long)'6' << 16) + ((long)'4' << 24))
```

Templates

A template has the following syntax definition

```
template              : TOKEN_TEMPLATE name TOKEN_OBRACE
                                              class_id
                                              template_parts
                                            TOKEN_CBRACE

template_parts        : template_members_part TOKEN_OBRACKET
                                                template_option_info
                                              TOKEN_CBRACKET
                      | template_members_list

template_members_part : /* Empty */
                      | template_members_list

template_option_info  : ellipsis
                      | template_option_list

template_members_list :    template_members
                      | template_members_list template_members

template_members      : primitive
                      | array
                      | template_reference

primitive             : primitive_type optional_name TOKEN_SEMICOLON

array                 : TOKEN_ARRAY array_data_type name dimension_list
                        TOKEN_SEMICOLON

template_reference    : name optional_name YT_SEMICOLON

primitive_type        : TOKEN_WORD
                      | TOKEN_DWORD
                      | TOKEN_FLOAT
                      | TOKEN_DOUBLE
                      | TOKEN_CHAR
                      | TOKEN_UCHAR
                      | TOKEN_SWORD
                      | TOKEN_SDWORD
                      | TOKEN_LPSTR
                      | TOKEN_UNICODE
                      | TOKEN_CSTRING
```

temp

```
array_data_type       : primitive_type
                      | name

dimension_list        : dimension
                      | dimension_list dimension

dimension             : TOKEN_OBRACKET dimension_size TOKEN_CBRACKET

dimension_size        : TOKEN_INTEGER
                      | name

template_option_list  : template_option_part
                      | template_option_list template_option_part

template_option_part  : name optional_class_id

name                  : TOKEN_NAME

optional_name         : /* Empty */
                      | name

class_id              : TOKEN_GUID

optional_class_id     : /* Empty */
                      | class_id

ellipsis              : TOKEN_DOT TOKEN_DOT TOKEN_DOT
```

[#] Data

A data object has the following syntax definition

```
object                : identifier optional_name TOKEN_OBRACE
                                                   optional_class_id
                                                   data_parts_list
                                                 TOKEN_CBRACE
data_parts_list       : data_part
                      | data_parts_list data_part

data_part             : data_reference
                      | object
                      | number_list
                      | float_list
                      | string_list

number_list           : TOKEN_INTEGER_LIST

float_list            : TOKEN_FLOAT_LIST
```

[#] dat

```
string_list           : string_list_1 list_separator

string_list_1         : string
                      | string_list_1 list_separator string

list_separator        : comma
                      | semicolon

string                : TOKEN_STRING

identifier            : name
                      | primitive_type

data_reference        : TOKEN_OBRACE name optional_class_id TOKEN_CBRACE
```

Tokens

Tokens are written as little endian DWORDs. A list of token values follows. The list is divided into record-bearing and stand-alone tokens.

Record-bearing

```
#define TOKEN_NAME 1
#define TOKEN_STRING 2
#define TOKEN_INTEGER 3
#define TOKEN_GUID 5
#define TOKEN_INTEGER_LIST 6
#define TOKEN_REALNUM_LIST 7
```

Stand-alone

```
#define TOKEN_OBRACE 10
#define TOKEN_CBRACE 11
#define TOKEN_OPAREN 12
#define TOKEN_CPAREN 13
#define TOKEN_OBRACKET 14
#define TOKEN_CBRACKET 15
#define TOKEN_OANGLE 16
#define TOKEN_CANGLE 17
#define TOKEN_DOT 18
#define TOKEN_COMMA 19
#define TOKEN_SEMICOLON 20
#define TOKEN_TEMPLATE 31
#define TOKEN_WORD 40
#define TOKEN_DWORD 41
#define TOKEN_FLOAT 42
#define TOKEN_DOUBLE 43
#define TOKEN_CHAR 44
#define TOKEN_UCHAR 45
#define TOKEN_SWORD 46
#define TOKEN_SDWORD 47
#define TOKEN_VOID 48
```

tokens

```
#define TOKEN_LPSTR 49
#define TOKEN_UNICODE 50
#define TOKEN_CSTRING 51
#define TOKEN_ARRAY 52
```

Token Records

This section describes the format of the records for each of the record-bearing tokens.

TOKEN_NAME

Field	Type	Size (bytes)	Contents
token	DWORD	4	TOKEN_NAME
count	DWORD	4	Length of *name* field in bytes
name	BYTE array	*count*	ASCII name

TOKEN_NAME is a variable length record. The token is followed by a *count* value which specifies the number of bytes which follow in the *name* field. An ASCII name of length *count* completes the record.

TOKEN_STRING

Field	*Type*	*Size (bytes)*	*Contents*
token	DWORD	4	TOKEN_STRING
count	DWORD	4	Length of *string* field in bytes
string	BYTE array	count	ASCII string
terminator	DWORD	4	TOKEN_SEMICOLON or TOKEN_COMMA

TOKEN_STRING is a variable length record. The token is followed by a *count* value which specifies the number of bytes which follow in the *string* field. An ASCII string of length *count* continues the record which is completed by a terminating token. The choice of terminator is determined by syntax issues discussed elsewhere.

TOKEN_INTEGER

Field	*Type*	*Size (bytes)*	*Contents*

records

token	DWORD	4	TOKEN_INTEGER
value	DWORD	4	Single integer

TOKEN_INTEGER is a fixed length record. The token is followed by the integer value required.

TOKEN_GUID

Field	*Type*	*Size (bytes)*	*Contents*
token	DWORD	4	TOKEN_GUID
data1	DWORD	4	uuid data field 1
data2	WORD	2	uuid data field 2
data3	WORD	2	uuid data field 3
data4	BYTE array	8	uuid data field 4

TOKEN_GUID is a fixed length record. The token is followed by the four data fields as defined by the OSF DCE standard.

TOKEN_INTEGER_LIST

Field	*Type*	*Size (bytes)*	*Contents*
token	DWORD	4	TOKEN_INTEGER_LIST
count	DWORD	4	Number of integers in *list* field
list	DWORD array	*4 x count*	Integer list

TOKEN_INTEGER_LIST is a variable length record. The token is followed by a count value which specifies the number of integers which follow in the *list* field. For efficiency, consecutive integer lists should be compounded into a single list.

TOKEN_REALNUM_LIST

Field	*Type*	*Size (bytes)*	*Contents*
token	DWORD	4	TOKEN_REALNUM_LIST
count	DWORD	4	Number of floats or doubles in *list* field
list	float/double	*4 or 8 x count*	Float or double list

array

TOKEN_REALNUM_LIST is a variable length record. The token is followed by a count value which specifies the number of floats or doubles which follow in the *list* field. The size of the floating point value (float or double) is determined by the value of *float size* specified in the file header discussed elsewhere. For efficiency, consecutive realnum lists should be compounded into a single list.

Example Templates

Two example binary template definitions are given below. Note that data is stored in little endian format, which is not shown in these illustrative examples.

The closed template *RGB* is identified by the uuid {55b6d780-37ec-11d0-ab39-0020af71e433} and has three members *r*, *g*, and *b* each of type *float*

```
TOKEN_TEMPLATE, TOKEN_NAME, 3, 'R', 'G', 'B', TOKEN_OBRACE,
TOKEN_GUID, 55b6d780, 37ec, 11d0, ab, 39, 00, 20, af, 71, e4, 33,
TOKEN_FLOAT, TOKEN_NAME, 1, 'r', TOKEN_SEMICOLON,
TOKEN_FLOAT, TOKEN_NAME, 1, 'g', TOKEN_SEMICOLON,
TOKEN_FLOAT, TOKEN_NAME, 1, 'b', TOKEN_SEMICOLON,
TOKEN_CBRACE
```

The closed template *Matrix4x4* is identified by the uuid {55b6d781-37ec-11d0-ab39-0020af71e433} and has one member, a two-dimensional array named *matrix* of type *float*

```
TOKEN_TEMPLATE, TOKEN_NAME, 9, 'M', 'a', 't', 'r', 'i', 'x', '4', 'x', '4',
TOKEN_OBRACE,
TOKEN_GUID, 55b6d781, 37ec, 11d0, ab, 39, 00, 20, af, 71, e4, 33,
TOKEN_ARRAY, TOKEN_FLOAT, TOKEN_NAME, 6, 'm', 'a', 't', 'r', 'i', 'x',
TOKEN_OBRACKET, TOKEN_INTEGER, 4, TOKEN_CBRACKET,
TOKEN_OBRACKET, TOKEN_INTEGER, 4, TOKEN_CBRACKET,
TOKEN_CBRACE
```

Example Data

The binary data object below shows an instance of the *RGB* template defined above. The example object is named *blue* and its three members *r, g,* and *b* have the values 0.0, 0.0 and 1.0 respectively. Note that data is stored in little endian format which is not shown in this illustrative example.

```
TOKEN_NAME, 3, 'R', 'G', 'B', TOKEN_NAME, 4, 'b', 'l', 'u', 'e', TOKEN_OBRACE,
TOKEN_FLOAT_LIST, 3, 0.0, 0.0, 1.0, TOKEN_CBRACE
```

[#] Microsoft does not make any representation or warranty regarding this specification or any product or item developed based on this specification. Microsoft disclaims all express and implied warranties, including but not limited to the implied warranties of merchantability, fitness

[#] disclaimer

for a particular purpose and freedom from infringement. Without limiting the generality of the foregoing, Microsoft does not make any warranty of any kind that any item developed based on this specification, or any portion of it, will not infringe any copyright, patent, trade secret or other intellectual property right of any person or entity in any country. It is your responsibility to seek licenses for such intellectual property rights where appropriate. Microsoft shall not be liable for any damages arising out of or in connection with the use of this specification, including liability for lost profit, business interruption, or any other damages whatsoever. Some states do not allow the exclusion or limitation of liability for consequential or incidental damages; the above limitation may not apply to you.