Direct-X File Format

Compiled by Paul Bourke
January 1999

DirectX 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.

This following sections deal with the content and syntax of the file format. The file format uses the extension .x when used with the DirectX Software Development Kit (SDK).

Reserved Words

The following words are reserved and must not be used:

- ARRAY
- BINARY
- BINARY_RESOURCE
- CHAR
- CSTRING
- DOUBLE
- DWORD
- FLOAT
- SDWORD
- STRING
- SWORD
- TEMPLATE
- UCHAR
- ULONGLONG
- 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 bytes	"txt "	Text File
			"bin "	Binary File
			"tzip"	MSZip Compressed Text File
			"bzip"	MSZip Compressed Binary File

Float size (required)	4 bytes 0064	64-bit floats
	0032	32-bit floats

Example

```
xof 0302txt 0064
```

Comments

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

```
# 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. This section discusses the following aspects of a template:

- Template form
- Template name
- UUID
- Members
- Restrictions

Example templates are presented in Examples.

Template form

A template has the following form.

```
template <template-name> {
<UUID>
<member 1>;
...
<member n>;
[restrictions]
}
```

Template name

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

UUID

A universally unique identifier (UUID) formatted to the Open Software Foundation's Distributed Computing Environment 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 as follows:

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 can 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>];
```

<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 for indicating an open template is three periods enclosed by square brackets.

```
[ ... ]
```

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 DirectX file format API. 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 bit 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 within 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] { [<UUID>]
<member 1>;
...
```

```
<member n>;
}
```

This section discusses the following parts of data objects:

- Identifier
- Name
- Members

Example templates are presented in Examples.

Identifier

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

Name

Name is optional. (See earlier for the syntax definition.)

Members

Data members can be one of the following: data object, data reference, integer list, float list, or string list.

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 for more information.

Data reference

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

```
{
  name |
  UUID |
  name UUID
}
```

Integer list

A semicolon-separated list of integers. For example:

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

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 topic is perhaps the most complex syntax issue in the file format, and it is very strict. Commas are used to separate array members; semicolons terminate each data item.

For example, if a template is defined as:

```
template mytemp {
DWORD myvar;
}
```

Then an instance of this template would look like:

```
mytemp dataTemp {
1;
}
```

A template containing another template is defined as:

```
template mytemp {
DWORD myvar;
DWORD myvar2;
}
template container {
FLOAT aFloat;
mytemp aTemp;
}
```

Then an instance of this template would look like:

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

Note that the second line that represents the *mytemp* inside *container* has two semicolons at the end of the line. The first semicolon indicates the end of the data item, *aTemp* (inside container), and the second semicolon indicates the end of the *container*.

If an array is defined as:

```
Template mytemp {
array DWORD myvar[3];
}
```

Then an instance of this would look like:

```
mytemp aTemp {
1, 2, 3;
}
```

In the array case, there is no need for the data items to be separated by semicolons because they are delineated by commas. The semicolon at the end marks the end of the array.

Consider a template that contains an array of data items defined by a template.

```
template mytemp {
DWORD myvar;
DWORD myvar2;
}
template container {
DWORD count;
array mytemp tempArray[count];
}
```

Then an instance of this would look like:

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

DirectX File Templates

This appendix lists the templates used by Direct3D Retained Mode. A familiarity with Direct3D Retained Mode data types is assumed.

Template Name: Header

UUID

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

Member Name	Type	Optional Array Size	Optional Data Objects
major	WORD		None
minor	WORD		
flags	DWORD		

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: Vector

UUID

<3D82AB5E-62DA-11cf-AB39-0020AF71E433>

Member Name	Type	Optional Array Size	Optional Data Objects
x	FLOAT		None
y	FLOAT		
z	FLOAT		

Description

This template defines a vector.

Template Name: Coords2d

UUID

<F6F23F44-7686-11cf-8F52-0040333594A3>

Member Name	Type	Optional Array Size	Optional Data Objects
u	FLOAT		None
v	FLOAT		

Description

A two dimensional vector used to define a mesh's texture coordinates.

Template Name: Quaternion

UUID

<10DD46A3-775B-11cf-8F52-0040333594A3>

Member Name	Type	Optional Array Size	Optional Data Objects
s	FLOAT		None
v	Vector		

Description

Currently unused.

Template Name: Matrix4x4

UUID

<F6F23F45-7686-11cf-8F52-0040333594A3>

Member Name	Type	Optional Array Size	Optional Data Objects
matrix	array FLOAT	16	None

Description

This template defines a 4×4 matrix. This is used as a frame transformation matrix.

Template Name: ColorRGBA

UUID

<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: ColorRGB

UUID

<D3E16E81-7835-11cf-8F52-0040333594A3>

Member Name	Type	Optional Array Size	Optional Data Objects
red	FLOAT		None
green	FLOAT		
blue	FLOAT		

Description

This template defines the basic RGB color object.

Template Name: Indexed Color

UUID

<1630B820-7842-11cf-8F52-0040333594A3>

Member Name	Type	Optional Array Size

index	DWORD
indexColor	ColorRGBA

Description

This template consists of an index parameter and an RGBA color and is used for defining mesh vertex colors. The index defines the vertex to which the color is applied.

Template Name: Boolean

UUID

<4885AE61-78E8-11cf-8F52-0040333594A3>

Member Name	Type	Optional Array Size	Optional Data Objects
DWORD	truefalse		None

Description

Defines a simple Boolean type. This template should be set to 0 or 1.

Template Name: Boolean2d

UUID

<4885AE63-78E8-11cf-8F52-0040333594A3>

Member Name	Type	Optional Array Size	Optional Data Objects
u	Boolean		None
v	Boolean		

Description

This template defines a set of two Boolean values used in the MeshFaceWraps template to define the texture topology of an individual face.

Template Name: Material

UUID

<3D82AB4D-62DA-11cf-AB39-0020AF71E433>

Member Name	Type	Optional Array Size	Optional Data Objects
faceColor	ColorRGBA		Any
power	FLOAT		
specularColor	ColorRGB		
emissiveColor	ColorRGB		

Description

This template defines a basic material color that 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.

TextureFilename is an optional data object used by Direct3D Retained Mode. If this object is not present, the face is untextured.

Template Name: TextureFilename

UUID

<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 file name of a texture to apply to a mesh or a face. This template should appear within a material object.

Template Name: MeshFace

UUID

<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: MeshFaceWraps

UUID

<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. The value of the nFaceWrapValues member should be equal to the number of faces in a mesh.

Template Name: MeshTextureCoords

UUID

<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: MeshNormals

UUID

<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 is the normal vectors themselves, and the second array is an array of indexes specifying which normals should be applied to a given face. The value of the nFaceNormals member should be equal to the number of faces in a mesh.

Template Name: MeshVertexColors

UUID

<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, instead of applying a material per face or per mesh.

Template Name: MeshMaterialList

UUID

<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. The nMaterials member specifies how many materials are present, and materials specify which material to apply.

Template Name: Mesh

UUID

<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.

Optional Data Elements

The following optional data elements are used by Direct3D Retained Mode.

MeshFaceWraps	If this is not present, wrapping for both u and v defaults to false.
MeshTextureCoords	If this is not present, there are no texture coordinates.
MeshNormals	If this is not present, normals are generated using the GenerateNormals method of the API.
MeshVertexColors	If this is not present, the colors default to white.
MeshMaterialList	If this is not present, the material defaults to white.

Template Name: FrameTransformMatrix

UUID

<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: Frame

UUID

<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.

Optional Data Elements

The following optional data elements are used by Direct3D Retained Mode.

FrameTransformMatrix	If this element is not present, no local transform is applied to the frame.
Mesh	Any number of mesh objects that become children of the frame. These objects can be specified inline or by reference.

Template Name: FloatKeys

UUID

<10DD46A9-775B-11cf-8F52-0040333594A3>

Member Name	Type	Optional Array Size	Optional Data Objects
nValues	DWORD		None
values	array FLOAT	nValues	

Description

This template defines an array of floating-point numbers (floats) and the number of floats in that array. This is used for defining sets of animation keys.

Template Name: TimedFloatKeys

UUID

<F406B180-7B3B-11cf-8F52-0040333594A3>

Member Name	Type	Optional Array Size	Optional Data Objects
time	DWORD		None
tfkeys	FloatKeys		

Description

This template defines a set of floats and a positive time used in animations.

Template Name: AnimationKey

UUID

<10DD46A8-775B-11cf-8F52-0040333594A3>

Member Name	Type	Optional Array Size	Optional Data Objects
keyType	DWORD		None
nKeys	DWORD		

keys	array TimedFloatKeys	nKeys

Description

This template defines a set of animation keys. The keyType member specifies whether the keys are rotation, scale or position keys (using the integers 0, 1, or 2 respectively).

Template Name: AnimationOptions

UUID

<E2BF56C0-840F-11cf-8F52-0040333594A3>

Member Name	Type	Optional Array Size	Optional Data Objects
openclosed	DWORD		None
positionquality	DWORD		

Description

This template enables you to set the Direct3D Retained Mode Animation options. The openclosed member can be either 0 for a closed or 1 for an open animation. The positionquality member 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 closed.

Template Name: Animation

UUID

<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 also can contain an AnimationOptions data object.

Optional Data Elements

The following optional data elements are used by Direct3D Retained Mode.

AnimationKey	An animation is meaningless without AnimationKeys.
AnimationOptions	If this element is not present, an animation is closed.

Template Name: AnimationSet

UUID

<3D82AB50-62DA-11cf-AB39-0020AF71E433>

Member Name	Type	Optional Array Size	Optional Data Objects
none			Animation

Description

An AnimationSet template contains one or more Animation objects and is the equivalent to the Direct3D 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.

DirectX File Cube Example

This section describes a simple cube and then shows how to add textures, frames, and animations to it.

A Simple Cube

This file defines a simple cube that has four red sides 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;,
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 definitions
{GreenMaterial}         // 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

To add textures, use 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 as follows:

```
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

This section shows how to add frames and animations to a simple cube.

- Frames
- AnimationSets 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]
}
```

Place the previously defined cube mesh inside a frame with an identity transform. Then 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}        // You could have the mesh inline, but this                  // uses an object reference instead.
}
```

AnimationSets and Animations

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

```
Animation Animation0 {        // The name is chosen for convenience.
{ Frame that it applies to&em;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 ... } }

Now 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;;;
}
}
}
```