IDTF (Intermediate Data Text File) Format Description Version 100

8 March 2005

* Other names and brands may be claimed as the property of others.

Table of Contents

Basic Data Types

Following table contains description of all primitive data types, which can be used for IDTF format.

BOOL	Boolean value that's either "TRUE" or "FALSE"
FLOAT	Float number with decimal delimiter (point)
HEX	Unsigned hexadecimal number
INT	Unsigned integer number
INT2	2 unsigned integer numbers delimited by single space
INT3	3 unsigned integer numbers delimited by single space
COLOR3	3 float numbers delimited by single space in following order - R G B
COLOR4	4 float numbers delimited by single space in following order - R G B A
POINT3	3 float numbers divided by single space in following order - X Y Z
VECTOR4	4 float numbers delimited by single space in following order - X Y Z W
QUAT	Quaternion: 4 floats representing rotation
STRING	Maximum length of string is 256 chars with terminating null on one line. List of strings can be located at one line or different ones if length of string list is more than 256 chars

File Structure

The IDTF file contains text blocks which define different scene objects.

<FILE_HEADER>
<SCENE_DATA>
<FILE_REFERENCE>
<NODES>
<NODE_RESOURCES>
<SHADER_RESOURCES>
<MOTION_RESOURCES>
<MODIFIERS>

File header block represents information about format name and version. It should be only one header block.
Scene data block provides information which is common for entire scene. It should be only one scene data block.
Node blocks represent hierarchical scene graph. Type of nodes are: Group, Light, View and Model.
Node resources blocks provide resource data for nodes in a scene. Node resource types are: Light, View and Model.
Shader resource blocks represent shading resources which are applicable for models in a scene.
Motion resources block represents motions used by nodes.
Modifier blocks represent information about modifiers which can be added to modifier chain.

IDTF files should contain blocks in the order defined above.

Meta-Data

<BLOCK_TYPE>: META_DATA

There are a number of types of blocks defined in this format specification:
SCENE – defines common scene properties
NODE – defines node in the scene
RESOURCE – defines resource
MODIFIER – defines modifier

All these blocks can have meta-data. In case of resource block each resource in the resource list can have its own metadata.

The Meta-Data block contains a sequence of Key/Value pairs.

Description of <META_DATA> block:

```
META_DATA {
	META_DATA_COUNT <INT>: number_of_meta_data
	META_DATA 0 {
		META_DATA_ATTRIBUTE <STRING>: “BINARY” or “STRING”
		META_DATA_KEY <STRING>
		META_DATA_VALUE <STRING>
	}
	...
	META_DATA number_of_meta_data-1 {
		META_DATA_ATTRIBUTE <STRING>: “BINARY” or “STRING”
		META_DATA_KEY <STRING>
		META_DATA_VALUE <STRING>
	}
}
```

METADATA_VALUE example if METADATA_TYPE is “STRING” – “Some metadata value”.
METADATA_VALUE example if METADATA_TYPE is “BINARY” – “2C AD 02 74 2E 00 00 EC 00”.

File Header

File header defines file format extension in the form of string (“IDTF”) and file version in the form of integer number.
Example:

```
FILE_FORMAT “IDTF”
FILE_VERSION 100
```

Scene Data

At the beginning of file and after header there is some common scene information defined in SCENE block. It contains meta-data for entire scene (optional).

Scene block description:

```
SCENE {
	<META_DATA>
}
```

This block is optional.
Look at the meta-data description in the meta-data section.

File Reference

A File Reference block contains information for finding a single file that is associated with this file and is loaded with it. Multiple locations for the file may be specified. The File Reference block may also contain filters that load a portion of the file based on name, object type, or position.

```
FILE_REFERENCE {
	SCOPE_NAME <STRING>
	URL_COUNT <INT>
	URL_LIST {
		URL 0 <STRING>
		...
		URL url_count-1 <STRING>
	}
	FILTER_COUNT <INT>
	FILTER_LIST {
		FILTER 0 {
			<FILTER_DATA>
		}
		...
		FILTER filter_count-1 {
			<FILTER_DATA>
		}
```

```
        }
        NAME_COLLISION_POLICY <STRING>
        WORLD_ALIAS_NAME <STRING>
}
```

SCOPE_NAME
Used to identify the external file reference. Depending on the collision policy, the scope name may be used to modify the names of objects in the referenced file.

URL_LIST
List of Strings identifying the external file locations. Multiple locations can be specified for the external file. The player shall load the file from one of the locations. HTTP and FTP protocols will be recognized with absolute and relative addressing.

FILTER_COUNT
Number of filters to apply when loading the referenced file. If the filter count is zero, then all objects from the referenced file are loaded. If the filter count is greater than zero, then objects from the referenced file shall only be loaded if they match the specification of at least one of the filters. A modifier object shall be loaded if and only if the object it modifies is loaded.

FILTER
There are two types of filters.

<FILTER_DATA> block depends on the type of filter.

First is object name filter:
The <FILTER_DATA> for object name filter is:

TYPE “NAME”
OBJECT_NAME <STRING>

String used to filter objects by name. An object shall be loaded if its name matches Object Name Filter.

May contain the wildcard characters question mark ‘?’ and asterisk ‘*’. The question mark wildcard matches any one character at that position. The asterisk wildcard matches any zero or more characters at that position.

Second is object type filter:
The <FILTER_DATA> for object type filter is:

TYPE “TYPE”
OBJECT_TYPE <HEX>

Used to filter objects by type. An object shall be loaded if the block type of its declaration block matches Object Type Filter.

NAME_COLLISION_POLICY
A name collision occurs when the file being loaded contains an object with the same name as an object that already exists either loaded previously or created programmatically. Name Collision Policy indicates how name collisions are to be handled. Valid values are:

“REPLACE” – Replace existing object with the new object from external file.
“DISCARD”” – Discard the new object from external file.
“PREPEND_ALL” – Prepend scope name to object name for all objects from the external file
“PREPENDCOLLIDED” – Prepend scope name to new object name if there is a collision.
“POSTMANGLE” – Append instance number to new object name if there is a collision.

Prepending the scope name avoids collisions but does not prevent them in all cases. The new name with prepended scope name may still collide with an existing object. In this situation, the new object from the external file will replace that existing object.

When appending instance numbers, instance numbers shall be chosen to avoid collision with previously loaded objects.

WORLD_ALIAS_NAME
The world is the default node. The name of the default node is the empty string. Any references to the default node in the external file are replaced with a reference to the node named by World Alias Name.

Nodes

Nodes are the entities that populate the scene graph. Each node type contains a name, the number of parents it has, the name of each parent, and a transform for each parent specifying the position and orientation of the node relative to that parent. Nodes (except for the group node, see below) also have an associated node resource that is specified by ID. Also nodes can have an associated meta-data. To allow data sharing, multiple nodes may use the same resource.

Common node block description:

```
NODE <NODE_TYPE> {
      NODE_NAME <STRING>
      PARENT_LIST {
            PARENT_COUNT <INT>: n
            PARENT 0 {
                  PARENT_NAME “STRING”
                  PARENT_TM {
                        <TM_DATA>
                  }
            }
            ...
            PARENT n-1 {
                  PARENT_NAME “STRING”
                  PARENT_TM {
                        <TM_DATA>
                  }
            }
      }
      RESOURCE_NAME <STRING>
      <META_DATA>
}
```

Description of node block’s fields:

<NODE_TYPE>: the string without tabs and spaces, which allows to unique identify the type of node. A node type can be:

MODEL (can be mesh, line set and point set)
VIEW
LIGHT
GROUP

<NODE_NAME>: node’s name (e.g. “Sphere01”);

<PARENT_LIST>: list of names of node’s parents and TM relative to parents;

<TM_DATA>: transform matrix for each parent specifying the position and orientation of the node relative to that parent

```
      TM {
            <TM_COLUMN0>
            <TM_COLUMN1>
            <TM_COLUMN2>
            <TM_COLUMN3>
      }
```

<TM_COLUMN<N>>: Nth column of node’s TM. It must be 4 floats. TM has 4 rows and 4 columns of floats. (Node’s TM must be local object’s TM in parent space);

<RESOURCE_NAME>: allows to find node type's dependent data in the node resource list. The name here should exactly match the name of existing resource.

<METADATA>: meta-data specific to this node (optional).

Look at the specific node resource description below in the resource section.
Look at the meta-data description in the meta-data section.

Group

<NODE_TYPE>: GROUP

```
NODE "GROUP" {
      NODE_NAME <STRING>
      <PARENT_DATA>: see the common node block description
      <META_DATA>
}
```

There is no need for node resource for group node.

Light

<NODE_TYPE>: LIGHT

```
NODE "LIGHT" {
      NODE_NAME <STRING>
      <PARENT_DATA>: see the common node block description
      RESOURCE_NAME <STRING>
      <META_DATA>
}
```

View

<NODE_TYPE>: VIEW

```
NODE "VIEW" {
      NODE_NAME <STRING>
      <PARENT_DATA>: see the common node block description
      RESOURCE_NAME <STRING>
      VIEW_DATA {
            <VIEW_DATA>
      }
      <META_DATA>
}
```

Description of <VIEW_DATA>:

VIEW_ATTRIBUTE_SCREEN_UNIT <STRING>: "PIXEL" OR "PERCENT"
VIEW_TYPE <STRING>: (optional) "PERSPECTIVE" (default) OR "ORTHO"
VIEW_NEAR_CLIP <FLOAT>: near clipping distance (optional);
VIEW_FAR_CLIP <FLOAT>: far clipping distance (optional);
VIEW_PROJECTION <FLOAT>: field of view of the virtual camera in degrees for perspective or height of the orthographic view for orthographic;
<VIEW_PORT_DATA> (optional)
<BACKDROP DATA> (optional)
<OVERLAY_DATA> (optional)

Description of <VIEW_PORT_DATA>:

VIEW_PORT_WIDTH: width of the window in which the view will render
VIEW_PORT_HEIGHT: height of the window in which the view will render
VIEW_PORT_H_POSITION: horizontal position on the screen of the window in which the view will render
VIEW_PORT_V_POSITION: vertical position on the screen of the window in which the view will render

Description of backdrops and overlays:

```
BACKDROP_COUNT
BACKDROP_LIST {
      BACKDROP 0 {
            <VIEW_TEXTURE_DATA>
      }
      ...
      BACKDROP number_of_backdrops-1 {
            <VIEW_TEXTURE_DATA>
      }
}

OVERLAY_COUNT
OVERLAY_LIST {
      OVERLAY 0 {
            <VIEW_TEXTURE_DATA>
      }
      ...
      OVERLAY number_of_overlays-1 {
            <VIEW_TEXTURE_DATA>
      }
}
```

Description of <VIEW_TEXTURE_DATA>:

TEXTURE_NAME <STRING>: name of the texture resource to use with backdrop or overlay
TEXTURE_BLEND <FLOAT>: blend factor used with the backdrop's or overlay's texture
ROTATION <FLOAT>: texture used with the backdrop or overlay is rotated
LOCATION_X <FLOAT>: backdrop's or overlay's horizontal location
LOCATION_Y <FLOAT>: backdrop's or overlay's vertical location
REG_POINT_X <INT>: horizontal registration point
REG_POINT_Y <INT>: registration point
SCALE_X <FLOAT>: scale factor applied to the backdrop or overlay horizontally
SCALE_Y <FLOAT>: scale factor applied to the backdrop or overlay vertically

Model

<NODE_TYPE>: MODEL

```
NODE "MODEL" {
      NODE_NAME <STRING>
      <PARENT_DATA>: see the common node block description
      RESOURCE_NAME <STRING>
      MODEL_VISIBILITY <STRING>
      <META_DATA>
}
```

Model Visibility is used to indicate whether the front facing or back facing surface should be drawn. Possible values:

"NONE" – no polygons are drawn and the model is invisible.
"FRONT" – only polygons on the outer surface of the model are drawn, so that, if the camera were inside the model, the model wouldn't be seen.
"BACK" – only polygons on the inside of the object are drawn, so that if the camera were outside the model, the model wouldn't be seen.
"BOTH" – all polygons are drawn and the model is visible regardless of orientation.

Resources

Resource lists are used for resource sharing.

Common resource list block format:

```
RESOURCE_LIST <RESOURCE_TYPE> {
	RESOURCE_COUNT <INT>: number_of_resources in the list
	RESOURCE 0 {
		RESOURCE_NAME <STRING>
		<RESOURCE_DATA>: resource specific data
		<META_DATA>
	}
	...
	RESOURCE number_of_resources-1 {
		RESOURCE_NAME <STRING>
		<RESOURCE_DATA>: resource specific data
		<META_DATA>
	}
}
```

<RESOURCE_TYPE>: the string which allows to identify type of resource data.

Possible `RESOURCE_LIST` types:

VIEW
LIGHT
MODEL
SHADER
MOTION

IDTF file can have only one list of each resource list type. They should be located after node's blocks.

Light Resource

<RESOURCE_TYPE>: LIGHT

Description of `<RESOURCE_DATA>` for Light:

`LIGHT_TYPE <STRING>`: light type;
`LIGHT_COLOR <COLOR4>`: diffuse color;
`LIGHT_ATTENUATION <POINT3>`: attenuation (constant, linear, quadratic factors);
`LIGHT_SPOT_ANGLE <FLOAT>`: spot angle, used only for spot light;
`LIGHT_INTENSITY <FLOAT>`: intensity

Possible values of `LIGHT_TYPE`:

"AMBIENT" - Light provides uniform non-directional light to the scene.
"DIRECTIONAL" - Light provides uniform directional light to the scene.
"POINT" - Light is emitted from a specific point in the scene.
"SPOT" - Like point light, but constrained to specific directions.

View Resource

<RESOURCE_TYPE>: VIEW

Description of `<RESOURCE_DATA>` for View:

```
VIEW_PASS_COUNT <INT>: the number of passes that are used when rendering this view
VIEW_ROOT_NODE_LIST {
	ROOT_NODE 0 {
	ROOT_NODE_NAME <STRING>:  the name of a node the view will render
	}
	...
	ROOT_NODE n-1 {
		ROOT_NODE_NAME <STRING>
	}
```

```
}
```

Model Resource

<RESOURCE_TYPE>: MODEL

Model Type

```
MODEL_TYPE <STRING>
```

Model type defines the type of model resource and can be three types – point set, line set and mesh. So there are three types of block:
MESH
POINT_SET
LINE_SET

Common lists for all model types:

Shading Description List

```
MODEL_SHADING_DESCRIPTION_LIST {
      SHADING_DESCRIPTION 0 {
            TEXTURE_LAYER_COUNT <INT>: number_of_texture_layers used by this shader list
            TEXTURE_COORD_DIMENSION_LIST {
                  TEXTURE_LAYER 0 DIMENSION: m
                  ...
                  TEXTURE_LAYER number_of_texture_layers-1 DIMENSION: k
            }
            SHADER_ID <INT>
      }
      ...
      SHADING_DESCRIPTION number_of_shaders-1 {
            TEXTURE_LAYER_COUNT <INT>: number_of_texture_layers used by this shader list
            TEXTURE_COORD_DIMENSION_LIST {
                  TEXTURE_LAYER 0 DIMENSION: m
                  ...
                  TEXTURE_LAYER number_of_texture_layers-1 DIMENSION: k
            }
            SHADER_ID <INT>
      }
}
```

This list represents shading descriptions used in the mesh. Each shading description corresponds to one shader in the shader resource list.
If shading description defines the number of texture layers to be zero only material properties from shader is used.

<SHADER_ID> - shading index for this shading description. Each shading description corresponds to one shader list in the shading group of corresponding shading modifier.

`m, k` - number of dimensions in the texture coordinate vector, can be 1, 2, 3 or 4.

Model Skeleton

```
MODEL_SKELETON {
      BONE 0 {
            BONE_NAME <STRING>
            PARENT_BONE_NAME <STRING>
            BONE_LENGTH <FLOAT>
            BONE_DISPLACEMENT <POINT3>
            BONE_ORIENTATION <VECTOR4>
            BONE_LINK_COUNT <INT>
```

```
            BONE_LINK_LENGTH <VECTOR4>
            BONE_START_JOINT <VECTOR4>
            BONE_END_JOINT <VECTOR4>
            BONE_ROTATION_CONSTRAINTS_MAX <POINT3>
            BONE_ROTATION_CONSTRAINTS_MIN <POINT3>
      }
      ...
      BONE number_of_bones-1 {
      }
}
```

This list represents bone structure information.

Mesh

Description of `<RESOURCE_DATA>` for mesh:

```
MESH {
      FACE_COUNT <INT>: number of faces in the face list
      MODEL_POSITION_COUNT <INT>: number of positions in the base position list
      MODEL_BASE_POSITION_COUNT <INT>: number of base positions in the base position list
      MODEL_NORMAL_COUNT <INT>: number of mesh (face + position) normals in the normal list
      MODEL_DIFFUSE_COLOR_COUNT <INT>: number of diffuse vertex colors
      MODEL_SPECULAR_COLOR_COUNT <INT>: number of specular vertex colors
      MODEL_TEXTURE_COORD_COUNT <INT>: number of texture coordinates
      MODEL_BONE_COUNT <INT>: number of bone structures
      MODEL_SHADING_COUNT <INT>: number of shading descriptions used in the mesh

      MODEL_SHADING_DESCRIPTION_LIST {...}
      MESH_FACE_POSITION_LIST {...}
      MESH_FACE_NORMAL_LIST {...}
      MESH_FACE_SHADER_LIST {...}
      MESH_FACE_TEXTURE_LAYER_LIST {...}
      MESH_FACE_DIFFUSE_COLOR_LIST {...}
      MESH_FACE_SPECULAR_COLOR_LIST {...}
      MODEL_POSITION_LIST {...}
      MODEL_NORMAL_LIST {...}
      MODEL_DIFFUSE_COLOR_LIST {...}
      MODEL_SPECULAR_COLOR_LIST {...}
      MODEL_TEXTURE_COORD_LIST {...}
      MODEL_SKELETON {...}
}
```

Descriptions of several mesh lists:

Face Position List

```
MESH_FACE_POSITION_LIST {
      <INT3>
      ...
      <INT3>
}
```

This list represents indices of faces into the position list. The size of this list is the number of faces.

Face Normal List

```
MESH_FACE_NORMAL_LIST {
      <INT3>
      ...
      <INT3>
```

```
}
```

This list represents indices of faces into the normal list. The size of this list is the number of faces.

Face Shading List

```
MESH_FACE_SHADING_LIST {
      <INT>
      ...
      <INT>
}
```

This list represents array of indices to the shading descriptions in the shading description list used by mesh faces. The size of this list is the number of faces. Shading ID is used to identify shading description used by each face in the next list.

Face Texture Coordinate List

```
MESH_FACE_TEXTURE_COORD_LIST {
      FACE 0 {
            TEXTURE_LAYER 0 TEX_COORD: <INT3>
            ...
            TEXTURE_LAYER n-1 TEX_COORD: <INT3>
      }
      FACE number_of_faces-1 {
            TEXTURE_LAYER 0 TEX_COORD: <INT3>
            ...
            TEXTURE_LAYER m-1 TEX_COORD: <INT3>
      }
}
```

This list represents indices of texture coordinates in the texture coordinate array for faces. The size of this list is the number of faces. n and m are numbers of texture layers for faces.
This list is optional if there are no defined texture layers for faces in this mesh.
Face element can be omitted if there are no texture layers for this face.

Face Diffuse Color List

```
MESH_FACE_DIFFUSE_COLOR_LIST {
      <INT3>
      ...
      <INT3>
}
```

This list represents indices of faces into the vertex diffuse color array. The size of this list is the number of faces.

Face Specular List

```
MESH_FACE_SPECULAR_COLOR_LIST {
      <INT3>
      ...
      <INT3>
}
```

This list represents indices of faces into the vertex specular color array. The size of this list is the number of faces.

Position List

```
MODEL_POSITION_LIST {
      <POINT3>,
```

```
      ...
      <POINT3>
}
```

This list represents coordinates of every vertex.

Normal List

```
MODEL_NORMAL_LIST {
      <POINT3>
      ...
      <POINT3>
}
```

This list represents normals. One normal for every face position, 3 normals per face. The size of this list is the numbers of faces multiply 3.

Diffuse Color List

```
MODEL_DIFFUSE_COLOR_LIST {
      <COLOR4>,
      ...
      <COLOR4>
}
```

This list represents vertices diffuse colors.

Specular Color List

```
MODEL_SPECULAR_COLOR_LIST {
      <COLOR4>,
      ...
      <COLOR4>
}
```

This list represents vertices specular colors.

Texture Coordinate List

```
MODEL_TEXTURE_COORD_LIST {
      <VECTOR4>
      ...
      <VECTOR4>
}
```

This list represents texture coordinates.

Base Position List

```
MESH_BASE_POSITION_LIST {
      <INT>
      ...
      <INT>
}
```

This list represents indices of base positions into the position list. The size of this list is the number of base positions.

Line Set

<RESOURCE_TYPE>: “LINE_SET”

Description of <RESOURCE_DATA> for line set:

```
LINE_SET {
      LINE_COUNT <INT>; number of line segments in the line set
      MODEL_POSITION_COUNT <INT>: number of positions in the position list
      MODEL_NORMAL_COUNT <INT>: number of normals in the normal list
      MODEL_DIFFUSE_COLOR_COUNT <INT>: number of diffuse colors
      MODEL_SPECULAR_COLOR_COUNT <INT>: number of specular colors
      MODEL_TEXTURE_COORD_COUNT <INT>: number of texture coordinates

      MODEL_SHADING_DESCRIPTION_LIST {...}
      LINE_POSITION_LIST {...}
      LINE_NORMAL_LIST {...}
      LINE_SHADING_LIST {...}
      LINE_TEXTURE_COORD_LIST {...}
      LINE_DIFFUSE_COLOR_LIST {...}
      LINE_SPECULAR_COLOR_LIST {...}
      MODEL_POSITION_LIST {...}
      MODEL_NORMAL_LIST {...}
      MODEL_DIFFUSE_COLOR_LIST {...}
      MODEL_SPECULAR_COLOR_LIST {...}
      MODEL_TEXTURE_COORD_LIST {...}
}
```

Descriptions of several line set lists:

Line Position List

```
LINE_POSITION_LIST {
      <INT2>
      ...
      <INT2>
}
```

This list represents indices of lines into the position list. The size of this list is the number of lines.

Line Normal List

```
LINE_NORMAL_LIST {
      <INT2>
      ...
      <INT2>
}
```

This list represents indices of lines into the normal list. The size of this list is the number of lines.

Line Shading List

```
LINE_SHADING_LIST {
      <INT>
      ...
      <INT>
}
```

This list represents array of indices to the shading descriptions in the shading description list used by model lines. The size of this list is the number of lines. Shading ID is used to identify shading description used by each line in the next list.

Line Texture Coordinate List

```
LINE_TEXTURE_COORD_LIST {
      LINE 0 {
```

```
            TEXTURE_LAYER 0 TEX_COORD: <INT2>
            ...
            TEXTURE_LAYER n-1 TEX_COORD: <INT2>
      }
      LINE number_of_faces-1 {
            TEXTURE_LAYER 0 TEX_COORD: <INT2>
            ...
            TEXTURE_LAYER m-1 TEX_COORD: <INT2>
      }
}
```

This list represents indices of texture coordinates in the texture coordinate array for line. The size of this list is the number of lines. n and m are numbers of texture layers for lines.
This list is optional if there are no defined texture layers for lines in this line set.
Line element can be omitted if there are no texture layers for this line.

Line Diffuse Color List

```
LINE_DIFFUSE_COLOR_LIST {
      <INT2>
      ...
      <INT2>
}
```

This list represents indices of lines into the vertex diffuse color array. The size of this list is the number of lines.

Line Specular Color List

```
LINE_SPECULAR_COLOR_LIST {
      <INT2>
      ...
      <INT2>
}
```

This list represents indices of lines into the vertex specular color array. The size of this list is the number of lines.

Position List

```
MODEL_POSITION_LIST {
      <POINT3>,
      ...
      <POINT3>
}
```

This list represents coordinates of every line.

Normal List

```
MODEL_NORMAL_LIST {
      <POINT3>
      ...
      <POINT3>
}
```

This list represents normals. One normal for every line position, 2 normals per line. The size of this list is the numbers of lines multiply 2.

Diffuse Color List

```
MODEL_DIFFUSE_COLORS_LIST {
```

```
      <COLOR4>,
      ...
      <COLOR4>
}
```

This list represents vertices diffuse colors.

Specular Color List

```
MODEL_SPECULAR_COLORS_LIST {
      <COLOR4>,
      ...
      <COLOR4>
}
```

This list represents vertices specular colors.

Texture Coordinate List

```
MODEL_TEXTURE_COORD_LIST {
      <VECTOR4>
      ...
      <VECTOR4>
}
```

This list represents texture coordinates.

Point Set

<RESOURCE_TYPE>: “POINT_SET”

Description of `<RESOURCE_DATA>` for point set:

```
POINT_SET {
      POINT_COUNT <INT>; number of points in the point set
      MODEL_POSITION_COUNT <INT>: number of positions in the position list
      MODEL_NORMAL_COUNT <INT>: number of normals in the normal list
      MODEL_DIFFUSE_COLOR_COUNT <INT>: number of diffuse colors
      MODEL_SPECULAR_COLOR_COUNT <INT>: number of specular colors
      MODEL_TEXTURE_COORD_COUNT <INT>: number of texture coordinates

      MODEL_SHADING_DESCRIPTION_LIST {...}
      POINT_POSITION_LIST {...}
      POINT_NORMAL_LIST {...}
      POINT_SHADING_LIST {...}
      POINT_TEXTURE_COORD_LIST {...}
      POINT_DIFFUSE_COLOR_LIST {...}
      POINT_SPECULAR_COLOR_LIST {...}
      MODEL_POSITION_LIST {...}
      MODEL_NORMAL_LIST {...}
      MODEL_DIFFUSE_COLOR_LIST {...}
      MODEL_SPECULAR_COLOR_LIST {...}
      MODEL_TEXTURE_COORD_LIST {...}
}
```

Descriptions of several point set lists:

Point Position List

```
POINT_POSITION_LIST {
      <INT>
      ...
```

```
      <INT>
}
```

This list represents indices of points into the position list. The size of this list is the number of points.

Point Normal List

```
POINT_NORMAL_LIST {
      <INT>
      ...
      <INT>
}
```

This list represents indices of points into the normal list. The size of this list is the number of points.

Point Shading List

```
POINT_SHADING_LIST {
      <INT>
      ...
      <INT>
}
```

This list represents array of indices to the shading descriptions in the shading description list used by model points. The size of this list is the number of points. Shading ID is used to identify shading description used by each point in the next list.

Point Texture Coordinate List

```
POINT_TEXTURE_COORD_LIST {
      POINT 0 {
            TEXTURE_LAYER 0 TEX_COORD: <INT>
            ...
            TEXTURE_LAYER n-1 TEX_COORD: <INT>
      }
      POINT number_of_faces-1 {
            TEXTURE_LAYER 0 TEX_COORD: <INT>
            ...
            TEXTURE_LAYER m-1 TEX_COORD: <INT>
      }
}
```

This list represents indices of texture coordinates in the texture coordinate array for point. The size of this list is the number of point. `n` and `m` are numbers of texture layers for points.
This list is optional if there are no defined texture layers for points in this point set.
Point element can be omitted if there are no texture layers for this point.

Point Diffuse Color List

```
POINT_DIFFUSE_COLOR_LIST {
      <INT>
      ...
      <INT>
}
```

This list represents indices of points into the vertex diffuse color array. The size of this list is the number of points.

Point Specular Color List

```
POINT_SPECULAR_COLOR_LIST {
      <INT>
```

```
      ...
      <INT>
}
```

This list represents indices of points into the vertex specular color array. The size of this list is the number of points.

Position List

```
MODEL_POSITION_LIST {
      <POINT3>,
      ...
      <POINT3>
}
```

This list represents coordinates of every point.

Normal List

```
MODEL_NORMAL_LIST {
      <POINT3>
      ...
      <POINT3>
}
```

This list represents normals. One normal per point. The size of this list is the numbers of points.

Diffuse Color List

```
MODEL_DIFFUSE_COLORS_LIST {
      <COLOR4>,
      ...
      <COLOR4>
}
```

This list represents vertices diffuse colors.

Specular Color List

```
MODEL_SPECULAR_COLORS_LIST {
      <COLOR4>,
      ...
      <COLOR4>
}
```

This list represents vertices specular colors.

Texture Coordinate List

```
MODEL_TEXTURE_COORD_LIST {
      <VECTOR4>
      ...
      <VECTOR4>
}
```

This list represents texture coordinates.

Lit Texture Shader Resource

<RESOURCE_TYPE>: SHADER

The Shader contains information needed to determine the appearance of a surface during rendering. The Shader includes references to Material Resources and Texture Resources and how to combine those resources when rendering.

Description of <RESOURCE_DATA> for Shader:

```
ATTRIBUTE_LIGHTING_ENABLED <BOOL>: (optional)
ATTRIBUTE_ALPHA_TEST_ENABLED <BOOL>: (optional)
ATTRIBUTE_USE_VERTEX_COLOR <BOOL>: (optional)
ATTRIBUTE_USE_FAST_SPECULAR <BOOL>: (optional)
SHADER_ALPHA_TEST_REFERENCE <FLOAT>: (optional)
SHADER_ALPHA_TEST_FUNCTION <STRING>: (optional)
SHADER_COLOR_BLEND_FUNCTION <STRING>: (optional)
SHADER_MATERIAL_NAME <STRING>
SHADER_ACTIVE_TEXTURE_COUNT <INT>
SHADER_TEXTURE_LAYER_LIST {
      TEXTURE_LAYER m < max_number_of_texture_layers {
            TEXTURE_LAYER_INTENSITY <FLOAT>: (optional, 1.0 by default)
            TEXTURE_LAYER_BLEND_FUNCTION <STRING>: (opt., MULTIPLY by default)
            TEXTURE_LAYER_BLEND_SOURCE <STRING>: (opt., CONSTANT by default)
            TEXTURE_LAYER_BLEND_CONSTANT <FLOAT>: (opt., 0.5 by default )
            TEXTURE_LAYER_MODE <STRING>: (optional, TM_NONE by default)
            TEXTURE_LAYER_ALPHA_ENABLED <BOOL>: (optional, FALSE by default)
            TEXTURE_NAME <STRING>
      }
      ...
      TEXTURE_LAYER n < max_number_of_texture_layers {
            TEXTURE_LAYER_INTENSITY <FLOAT>: (optional, 1.0 by default)
            TEXTURE_LAYER_BLEND_FUNCTION <STRING>: (opt., MULTIPLY by default)
            TEXTURE_LAYER_BLEND_SOURCE <STRING>: (opt., CONSTANT by default)
            TEXTURE_LAYER_BLEND_CONSTANT <FLOAT>: (opt., 0.5 by default )
            TEXTURE_LAYER_MODE <STRING>: (optional, TM_NONE by default)
            TEXTURE_LAYER_ALPHA_ENABLED <BOOL>: (optional, FALSE by default)
            TEXTURE_NAME <STRING>
      }
}
```

SHADER_ACTIVE_TEXTURE_COUNT - number of texture layers used by this shader, up to 8.

You can skip specific items in the Shader Texture Layer List. For example you can describe only 2 and 5 texture layers if you want only 2 and 5 texture layers to be active (enabled).

TEXTURE_ LAYER_INTENSITY - a scale factor applied to the color components of the texture.

TEXTURE_ LAYER_BLEND_FUNCTION - determines how the current texture layer is combined with the result from previous layers. Can be “ADD”, “MULTIPLY”, “REPLACE”, “BLEND”.

TEXTURE_ LAYER_BLEND_SOURCE - indicates whether the blending operation combines the current layer with the result from previous layers using a blending constant or the alpha value of each pixel. Can be “ALPHA”, “CONSTANT”.

0 – Alpha value of each pixel
1 – Blending constant.

TEXTURE_ LAYER_BLEND_CONSTANT - used when combining the results of texture layers.

A shader uses a Material to determine how surfaces will appear when rendered. A material associated with this shader determines how the shader appears when lit. The Material contains information defining how a material interacts with light in a scene.

Multi-material is supported. Nested sub-materials are not allowed.
Standard bitmaps and 2D procedural maps are supported.

Material Resource

<RESOURCE_TYPE>: MATERIAL

Description of <RESOURCE_DATA> for Material:

```
MATERIAL {
      ATTRIBUTE_AMBIENT_ENABLED <BOOL>: (optional)
      ATTRIBUTE_DIFFUSE_ENABLED <BOOL>: (optional)
      ATTRIBUTE_SPECULAR_ENABLED <BOOL>: (optional)
      ATTRIBUTE_EMISSIVE_ENABLED <BOOL>: (optional)
      ATTRIBUTE_REFLECTIVITY_ENABLED <BOOL>: (optional)
      ATTRIBUTE_OPACITY_ENABLED <BOOL>: (optional)
      MATERIAL_AMBIENT <COLOR4>: ambient color;
      MATERIAL_DIFFUSE <COLOR4>: diffuse color;
      MATERIAL_SPECULAR <COLOR4>: specular color;
      MATERIAL_EMISSIVE <COLOR4>: emissive (self illumination) color;
      MATERIAL_REFLECTIVITY <FLOAT>: reflectivity (shininess) parameter;
      MATERIAL_OPACITY <FLOAT>: opacity;
}
```

All attributes set to “TRUE” by default. If you want to switch off one of them you should specify it as “FALSE”

Texture Resource

<RESOURCE_TYPE>: TEXTURE

Description of <RESOURCE_DATA> for Texture:

```
IDTF_TEXTURE_HEIGHT <INT>: (optional)
IDTF_TEXTURE_WIDTH <INT>: (optional)
TEXTURE_IMAGE_TYPE <STRING>: (optional)
TEXTURE_IMAGE_COUNT <INT>: (optional)
IMAGE_FORMAT_LIST {: (optional)
      IMAGE_FORMAT 0 {
            <TEXTURE_IMAGE_FORMAT>
      }
      ...
      IMAGE_FORMAT number_of_image-1 {
            <TEXTURE_IMAGE_FORMAT>
      }
}
TEXTURE_PATH <STRING>: texture file name
```

TEXTURE_IMAGE_TYPE can be used to define the image type of texture being written to U3D file.

Possible values of this field are follows:
“ALPHA”
“RGB”
“RGBA”
“LUMINANCE”
“LUMINANCE_AND_ALPHA”

If this field is not defined the following default settings are used:

“RGB” for 3 channel texture

Description of <TEXTURE_IMAGE_FORMAT>:

```
COMPRESSION_TYPE <STRING>
ALPHA_CHANNEL <BOOL>
BLUE_CHANNEL <BOOL>
```

```
GREEN_CHANNEL <BOOL>
RED_CHANNEL <BOOL>
LUMINANCE <BOOL>
EXTERNAL_REFERENCE <BOOL>
URL_COUNT <INT>
URL_LIST {
      URL 0 <STRING>
      ...
      URL url_count-1 <STRING>
}
```

Channel attributes (alpha, blue, red, green channels and luminance) can be used only for channels which are composed in this image.
If channel attribute is not defined it is considered as “FALSE”.

If TEXTURE_IMAGE_TYPE is not defined the default settings are used for IMAGE_FORMAT (see below).

Possible values of COMPRESSION_TYPE:

“JPEG24” - color
“JPEG8” - greyscale
“PNG”

Default settings for “RGB”:

```
TEXTURE_IMAGE_COUNT 1
IMAGE_FORMAT_LIST {
      IMAGE_FORMAT 0 {
            COMPRESSION_TYPE “JPEG24”
            BLUE_CHANNEL “TRUE”
            GREEN_CHANNEL “TRUE”
            RED_CHANNEL “TRUE”
      }
}
```

If URL_COUNT and URL_LIST presents in any image format block then TEXTURE_PATH should not present and texture is considered as external.

Motion Resource

<RESOURCE_TYPE>: MOTION

Description of <RESOURCE_DATA> for Motion:

```
MOTION_TRACK_COUNT <INT>: number_of_motion_tracks
MOTION_TRACK 0 {
      MOTION_TRACK_NAME <STRING>: name of motion track
      MOTION_TRACK_TIME_COUNT <INT>: number_of_time_samples for this motion track
      KEY_FRAME 0 {
            KEY_FRAME_TIME <FLOAT>; time in seconds from beginning of animation;
            KEY_FRAME_DISPLACEMENT <POINT3>
            KEY_FRAME_ROTATION <QUAT>
            KEY_FRAME_SCALE <POINT3>
      }
      ...
      KEY_FRAME number_of_time_samples-1 {
            KEY_FRAME_TIME <FLOAT>: time in seconds from beginning of animation
            KEY_FRAME_DISPLACEMENT <POINT3>
            KEY_FRAME_ROTATION <QUAT>
            KEY_FRAME_SCALE <POINT3>
      }
}
```

```
...
MOTION_TRACK number_of_motion_tracks-1 {
      <MOTION_TRACK_KEY_FRAMES>
}
```

The motion track has one Key Frame for each time sample.

KEY_FRAME_DISPLACEMENT - translation of the start of the bone from the end of its parent bone. For a root bone or for a node, Displacement is the translation from the origin of the local coordinate space.

KEY_FRAME_ROTATION - the change in orientation of the bone relative to the parent bone. Rotation is expressed as a quaternion with the real part first.

KEY_FRAME_SCALE - the scaling component of the transformation of the bone relative to its parent bone.

Modifiers

Modifier blocks contain the information necessary to create certain modifiers that can be added to a modifier chain. It is assumed that every modifier is attached at the end of modifier chain.

```
MODIFIER <MODIFIER_TYPE> {
      MODIFIER_NAME <STRING>
      MODIFIER_CHAIN_TYPE <STRING>
      <MODIFIER_DATA>
      <META_DATA>
}
```

MODIFIER_TYPE is a string specified type of modifier. Possible values:
“ANIMATION”
“SHADING”
“BONE_WEIGHT”
“CLOD”
“SUBDIV”
“GLYPH”

MODIFIER_CHAIN_TYPE is a string specified type of modifier chain. Possible values:
“NODE”
“MODEL”

Shading Modifier

The Shading Modifier block describes the shading group that is used in the drawing of a renderable group. The shading modifier replaces the shading group associated with a renderable group.

Description of <MODIFIER_DATA> for shading modifier:

```
SHADER_LIST_COUNT <INT>: number of shader lists in the shading group
SHADING_GROUP {
      SHADER_LIST 0 {
            SHADER_COUNT <INT>: number of shaders in the shader list.
            SHADER_NAME_LIST {
                  SHADER 0 NAME: <STRING>: refers to a shader in the shader resource palette
                  ...
                  SHADER shader_count-1 NAME: <STRING>
            }
      }
      ...
      SHADER_LIST shader_list_count-1 {
            SHADER_COUNT <INT>
            SHADER_NAME_LIST {
                  SHADER 0 NAME: <STRING>
                  ...
                  SHADER shader_count-1 NAME: <STRING>
```

```
            }
      }
}
```

Animation Modifier

The Animation Modifier block describes parameters for animating a node or a renderable group. These parameters indicate which motion resources should be used and how they should be applied. When animating a node, the animation modifier changes the transforms for the node. When animating a renderable group, the animation modifier uses a skeleton defined by the generator and bone weights defined by a bone weight modifier to change the positions and normals in the renderable group

Description of <MODIFIER_DATA> for shading modifier:

```
ATTRIBUTE_ANIMATION_PLAYING <BOOL>
ATTRIBUTE_ROOT_BONE_LOCKED <BOOL>
ATTRIBUTE_SINGLE_TRACK <BOOL>
ATTRIBUTE_AUTO_BLEND <BOOL>
TIME_SCALE <FLOAT>
BLEND_TIME <FLOAT>
MOTION_COUNT <INT>
MOTION_INFO_LIST {
      MOTION_INFO 0 {
            MOTION_NAME <STRING>
            ATTRIBUTE_LOOP <BOOL>
            ATTRIBUTE_SYNC <BOOL>
            TIME_OFFSET <FLOAT>
            TIME_SCALE <FLOAT>
      }
      ...
      MOTION_INFO motion_count-1 {
            MOTION_NAME <STRING>
            ATTRIBUTE_LOOP <BOOL>
            ATTRIBUTE_SYNC <BOOL>
            TIME_OFFSET <FLOAT>
            TIME_SCALE <FLOAT>
      }
}
```

`ATTRIBUTE_ANIMATION_PLAYING` - Animation should start when possible.

`ATTRIBUTE_ROOT_BONE_LOCKED` - The node's root bone's transform does not change as a result of the animation.

`ATTRIBUTE_SINGLE_TRACK` - Playing a single track.

`ATTRIBUTE_AUTO_BLEND` - The bones' transforms should transition smoothly from one motion to the next during the animation.

`TIME_SCALE` - Time Scale is a scaling value for the times of the motions.

`MOTION_COUNT` - Number of motion resources referenced by this modifier. If the Motion Count is zero, the Animation Modifier will use the default motion.

`MOTION_NAME` - String that identifies a motion resource.

`ATTRIBUTE_LOOP` - Determines whether this motion repeats.

`ATTRIBUTE_SYNC` - Determines if all of the motion resources playing concurrently should end at the same time.

`TIME_OFFSET` - Number of milliseconds to offset the start time of the motion.

`TIME_SCALE` - Scaling factor for the time of this motion resource for this animation modifier.

Bone Weight Modifier

The Bone Weight Modifier block describes a set of bone weights that can be added to a modifier chain. The animation modifier uses the bone weights in combination with the skeleton to animate the positions in a renderable group (mesh, point set, or line set). The normals are also changed by the animation modifier.

```
MODIFIER_ATTRIBUTES <STRING>: “MESH” or “LINE_SET” or “POINT_SET”
INVERSE_QUANT <FLOAT>
POSITION_COUNT <INT>
POSITION_BONE_WEIGHT_LIST {
      BONE_WEIHT_LIST 0 {
            BONE_WEIGHT_COUNT <INT>: number_of_bones which have influence at this position
            BONE_INDEX_LIST {
                  0 <INT>
                  ...
                  bone_weight_count-1 <INT>
            }
            BONE_WEIGHT_LIST {
                  0 <INT>
                  ...
                  bone_weight_count-2 <INT>
            }
      }
      ...
      BONE_WEIHT_LIST number_of_positions-1 {
            ...
      }
}
```

Position weights specify how strongly each vertex is influenced by each bone in the skeleton.

Position Bone Weight List indicates which bones have a non-zero influence at this position. The reconstructed bone weights at this position should sum to +1.0. The bone weights cannot be negative.

Bone Index List is a list of the indices of the bone in the skeleton that has influence at this position. Bone Index is present only if Bone Weight Count is greater than zero.

Bone Weight List is a list of the quantized bone weight values. Quantized Bone Weight is present only if Bone Weight Count is greater than one.

For other than the last bone weight value, the reconstructed bone weight value is calculated as:

(reconstructed bone weight) = (Quantized Bone Weight) * (Bone Weight Inverse Quant)

The last bone weight value is reconstructed by subtracting the sum of all the other reconstructed bone weight values from +1.0. The sum of all the bone weights at this position will be +1.0.

CLOD Modifier

The CLOD Modifier adjusts the level of detail in the renderable meshes in the data packet. The CLOD Modifier block contains parameters for how the level of detail should be adjusted.

Description of <MODIFIER_DATA> for CLOD modifier:

```
ATTRIBUTE_AUTO_LOD_CONTROL <BOOL>
LOD_BIAS <FLOAT>
CLOD_LEVEL <FLOAT>
```

`LOD_BIAS` - Describes screen space metric calculation.

`CLOD_LEVEL` - The range for CLOD Modifier Level is 0.0 to 1.0.

The CLOD Modifier adjusts the resolution of the renderable meshes.

The target resolution is determined by multiplying the CLOD Modifier Level by the maximum resolution of the author mesh. If the target resolution is less than the minimum resolution, then the resolution will be adjusted to the minimum resolution.

If the automatic LOD control is enabled, then the automatic LOD control overrides the CLOD Modifier Level specified in this block.

Subdivision Modifier

The Subdivision Modifier increases the resolution of a shape by dividing polygons into smaller polygons. The Subdivision Modifier block contains parameters that control the performance and appearance of the output of the subdivision algorithm.

Description of <MODIFIER_DATA> for Subdivision modifier:

`ATTRIBUTE_ENABLED <BOOL>`: subdivision modifier is enabled or not
`ATTRIBUTE_ADAPTIVE <BOOL>`: subdivision modifier should use adaptive subdivision or not
`DEPTH <INT>`: maximum number of levels of subdivision
`TENSION <FLOAT>`: tension value used for adaptive subdivision
`ERROR <FLOAT>`: value of the screen space error metric.

Glyph Modifier

The Glyph Modifier contains information used to create a 2D shape. The shape is defined by a number of control points and parameters that define how to connect the points. The shape consists of a sequence of individual glyphs called a glyph string. Each glyph in the glyph string is defined by a sequence of drawing commands.

Description of <MODIFIER_DATA> for Glyph modifier:

```
ATTRIBUTE_BILLBOARD <BOOL>
GLYPH_COMMAND_COUNT <INT>
GLYPH_COMMAND_LIST {
      GLYPH_COMMAND 0 {
            TYPE <STRING>
            <COMMAND_PARAMETERS>
      }
      ...
      GLYPH_COMMAND command_count-1 {
            TYPE <STRING>
            <COMMAND_PARAMETERS>
      }
}
GLYPH_TM {
      <TM_DATA>
}
```

Possible command's type which can be defined by TYPE:

START_GLYPH_STRING	Start a sequence of glyph symbols.
END_GLYPH_STRING	End a sequence of glyph symbols.
START_GLYPH	Start a glyph.
END_GLYPH	End the current glyph definition.
START_PATH	Start a new path to be drawn.
END_PATH	End the current path.
MOVE_TO	Move the current drawing position.
LINE_TO	Draw a line from the current drawing position to the new position.
CURVE_TO	Draw a curve from the current drawing position to the new position. The curve shape is determined by two control points.

END_GLYPH, MOVE_TO, LINE_TO and CURVE_TO have additional parameters. Other glyph commands do not have any parameters.

END_GLYPH parameters describe the horizontal and vertical offsets between the starting point for this glyph and the starting point for next glyph.

```
END_GLYPH_OFFSET_X <FLOAT>
END_GLYPH_OFFSET_Y <FLOAT>
```

MOVE_TO parameters describe the new horizontal and vertical positions of the active point.

```
MOVE_TO_X <FLOAT>
MOVE_TO_Y <FLOAT>
```

LINE_TO parameters describe the horizontal and vertical positions of the end point of the line.

```
LINE_TO_X <FLOAT>
LINE_TO_Y <FLOAT>
```

CURVE_TO parameters describe the horizontal and vertical positions of the first, the second control and the end points of the curve.

```
CONTROL1_X <FLOAT>
CONTROL1_Y <FLOAT>
CONTROL2_X <FLOAT>
CONTROL2_Y <FLOAT>
ENDPOINT_X <FLOAT>
ENDPOINT_Y <FLOAT>
```

<TM_DATA>: transform matrix which make up the transform that is applied to the glyph modifier after drawing to place it in the 3D world. This is a relative transform when used as a modifier attached to a node.

```
GLYPH_TM {
      <TM_COLUMN0>
      <TM_COLUMN1>
      <TM_COLUMN2>
      <TM_COLUMN3>
}
```

<TM_COLUMN<N>>: Nth column of TM. It must be 4 floats. TM has 4 rows and 4 columns of floats.