The ZCB Format

Pierre Terdiman
Last revision : June, 18, 2001

The *ZCB* format is a very simple chunk-based format designed for tests purpose. Just in case you're wondering, ZCB stands for *Zappy's Custom Binary*, because « Zappy » was my nickname years ago.

Here it is:

Header

ZCB! format's signature (4 bytes)

udword file type:
ZCB_FILE_SCENE
ZCB_FILE_MOTION
All other values are now obsolete.

A scene file contains a complete MAX scene, including meshes, cameras, lights, etc. A motion file only contains motion data for Character Studio BIPEDs. A motion file only contains a MOVE chunk and one single motion.

Please note a ZCB_FILE_SCENE file can now also contain a MOVE chunk.

Data compression :
If the format's signature is ZCBP instead of ZCB!, this is a packed file. The header then contains extra information :

ZCBP format's signature (4 bytes)

udword Compression type
udword Original size
udword Compressed size
…followed by the compressed file. The compressed file is a ZCB file as well, which means the decompressed file will begin with a ZCB! signature, then follow the standard format.

Compression types are :
ZCB_COMPRESSION_ZLIB compressed with Zlib
ZCB_COMPRESSION_BZIP2 compressed with Bzip2

For more details, please refer to the provided source code.

MAIN Chunk

MAIN general chunk
udword general chunk version number (CHUNK_MAIN_VER)

// Time-related information
udword First frame
udword Last frame
udword Frame rate
udword Delta time

// Background color
float Background color red
float Background color green
float Background color blue

// Global ambient color
float Ambient color red
float Ambient color green
float Ambient color blue

// Scene info
string the user-defined scene info (a string of arbitrary length, null-terminated)

NB : this string is located in Summary Info in MAX.

// Scene statistics
udword Number of geomobjects (meshes, not including Character Studio skins)
udword Number of derived objects (=number of Character Studio skins)
udword Number of cameras
udword Number of lights
udword Number of shapes
udword Number of helpers
udword Number of controllers
udword Number of materials
udword Number of texmaps (textures)
udword Number of unknown nodes
udword Number of invalid nodes

MESH Chunk

MESH meshes chunk
udword meshes chunk version number (CHUNK_MESH_VER)

for each mesh
{
    BasicInfo basic info for current mesh
    ubyte 1 if the object has been collapsed, else 0

```
ubyte       1 if the object is a BIPED part, else 0
ubyte       1 if the object is an instance from another object, else 0
ubyte       1 for a target object, else 0
ubyte       1 if the object has been successfully converted, else 0
ubyte       1 if the mesh is a PHYSIQUE skin, else 0
ubyte       1 if the mesh can cast its shadow, else 0

if the mesh is a BIPED part or a skin
{
        udword      the owner character's ID
}

if the mesh is a BIPED part
{
        udword      the CSID
}

NB : if the mesh is an instance, the master mesh ID is the link ID from the
BasicInfo structure.

if the mesh is not an instance
{
        udword      Number of faces
        udword      Number of vertices
        udword      Number of texture-vertices
        udword      Number of vertex-colors
        udword      Flags
        ubyte       0/1 = mesh parity (CW / CCW)

        if (not a skin)
        {
                A list of vertices (quantized or not)
                NB : see at the end of the Mesh section how that list is exported.
        }
        else
        {
                if (Flags & ZCB_ONEBONEPERVERTEX)
                {
                        …
                        float       x,y,z       a list of offset vectors
                        …

                        …
                        udword      ID          a list of bone IDs
                        …
                }
                else
                {
                        …
                        udword      Nb          a number of bones /vertex
```

```
                …

                …
                udword      ID          a list of bone IDs
                …

                …
                float       weight      a list of weight values
                …

                …
                float       x,y,z       a list of offset vectors
                …
        }

        // Extra information about a compatible skeleton
        udword      Number of bones

        for each bone
        {
                udword      CSID        Bone's CSID
                udword      pCSID       Parent bone's CSID
        }
}

// Mapping coordinates
if (Flags & ZCB_UWV)
{
        A list of UV(W)s (quantized or not)

        NB : W not saved if (Flags & ZCB_WDISCARDED)
}

// Vertex colors
if( Flags & ZCB_VERTEXCOLORS)
{
        A list of vertex-colors (quantized or not)
}

// Topology : the way faces are saved has changed a lot.
Here's the old version :

for each face
{
        if (Flags & ZCB_VFACE)
        {
                udword      VRef0           vertex references
                udword      VRef1
                udword      VRef2
        }
```

```
        if (Flags & ZCB_TFACE)
        {
                udword      TRef0           texture-vertex refs
                udword      TRef1
                udword      TRef2
        }
        If (Flags & ZCB_CFACE)
        {
                udword      CRef0           vertex-color references
                udword      CRef1
                udword      CRef2
        }
        udword      MaterialID
        udword      Smoothing groups
        ubyte       Edge visibility code
}
```

And here's the new one :

```
Type = uword if (Flags & ZCB_WORDFACES)
Else Type = udword

if(Flags & ZCB_VFACE)
{
        for each face
        {
                Type   VRef0, VRef1, VRef2
        }
}

if((Flags & ZCB_TFACE) && (Flags & ZCB_UVW))
{
        for each face
        {
                Type   TRef0, TRef1, TRef2
        }
}

if((Flags & ZCB_CFACE) && (Flags & ZCB_VERTEXCOLORS))
{
        for each face
        {
                Type   CRef0, CRef1, CRef2
        }
}
```

NB : if (Flags & ZCB_COMPRESSED), references are also delta-encoded. At this point, please refer to the provided source code for details.

```
// Face parameters
for each face
{
	udword		MaterialID
}

for each face
{
	udword		Smoothing groups
}

if(Flags & ZCB_EDGEVIS)
{
	for each face
	{
		ubyte		Edge visibility code
	}
}
```

NB : hence, the way faces are exported has been turned upside-down, in order to please possible compression programs afterwards.

```
// Convex hull
if(Flags & ZCB_CONVEXHULL)
{
	udword		Number of vertices
	udword		Number of faces
	…
	float		x,y,z			a list of vertices
	…
	…
	udword		ref0, ref1, ref2	a list of faces
	…
}

// Bounding sphere
if(Flags & ZCB_BOUNDINGSPHERE)
{
	float		x,y,z			sphere center
	float		radius
}

// Volume integrals
if(Flags & ZCB_INERTIATENSOR)
{
	float		x,y,z			Center Of Mass (COM)
	float		Mass
	float		InertiaTensor[0][0]
	float		InertiaTensor[0][1]
	float		InertiaTensor[0][2]
```

```
        float       InertiaTensor[1][0]
        float       InertiaTensor[1][1]
        float       InertiaTensor[1][2]
        float       InertiaTensor[2][0]
        float       InertiaTensor[2][1]
        float       InertiaTensor[2][2]
        float       COMInertiaTensor[0][0]
        float       COMInertiaTensor[0][1]
        float       COMInertiaTensor [0][2]
        float       COMInertiaTensor [1][0]
        float       COMInertiaTensor [1][1]
        float       COMInertiaTensor [1][2]
        float       COMInertiaTensor [2][0]
        float       COMInertiaTensor [2][1]
        float       COMInertiaTensor [2][2]
}

// Consolidation
if(Flags & ZCB_CONSOLIDATION)
{
        // Topology
        uword       Total number of faces (TotalNbFaces)
        uword       Number of submeshes

        // Submeshes
        for each submesh
        {
                sdword      Material ID
                udword      Smoothing groups
                udword      Number of faces
                udword      Number of vertices
                udword      Number of strips (obsolete)
        }

        // Connectivity
        for each submesh
        {
                uword       Number of faces
                for each face
                {
                        uword       Ref0
                        uword       Ref1
                        uword       Ref2
                }
        }

        // Face normals
        if (Flags & ZCB_FACENORMALS)
        {
                for each face (0 to TotalNbFaces)
```

```
        {
                float   Nx,Ny,Nz                a face normal
        }
}

// Geometry
uword           Number of vertices in the original mesh
uword           Number of vertices in the final mesh (NbVerts)
uword           Number of texture-vertices in trhe original mesh

// Indexed geometry
for each vertex in the final mesh
{
        udword          a reference in the following vertex-pool
}

// Pool of vertices
udword          Number of vertices in the pool
for each vertex in the pool
{
        float   x,y,z           a position
}

if (Flags & ZCB_UVW)
{
        // Indexed Uvs
        for each vertex in the final mesh
        {
                udword          a reference in the following UV-pool
        }

        // Pool of UVWs
        udword          Number of UVWs in the pool
        for each texture-vertex in the pool
        {
                float   u,v     mapping coordinates
                if( !Flags & ZCB_WDISCARDED)            float   w
        }
}

// Vertex normals
if( Flags & ZCB_VERTEXNORMALS)
{
        udword          Number of normals (=NbVerts)
        for each normal
        {
                float   x,y,z           a vertex normal
        }
}
```

```
                        // Vertex colors (not indexed)
                        if( Flags & ZCB_VERTEXCOLORS)
                        {
                                udword          Number of vertex-colors (=NbVerts)
                                for each vertex-color
                                {
                                        float   r,g,b           a vertex color
                                }
                        }

                        // Normal info
                        if( Flags & ZCB_NORMALINFO)
                        {
                                udword          NormalInfo size (nisize)

                                then a list of nisize udwords
                                NB : see FlexporterSDK.h for more info about that.
                        }

                        // Materials
                        udword          Number of materials

                        for each material
                        {
                                sdword          Material ID
                                udword          Number of faces using this material
                                udword          Number of related vertices
                                udword          Number of related submeshes
                        }
                }
        }

        // Lighting values – even for instances since an instance has its own PRS,
        // hence its own lighting
        udword          Number of colors

        for each color
        {
                float   r,g,b
        }
}
```

List of vertices are saved in the following way :

```
If(ZCB_QUANTIZEDVERTICES)
{
        // Quantized vertices
        float   Dequantization coeff X
```

```
        float   Dequantization coeff Y
        [float  Dequantization coeff Z]

        for each vertex
        {
                sword           quantized x
                sword           quantized y
                [sword          quantized z]
        }
}
else
{
        // Non-compressed, just a list of floats as before
        for each vertex
        {
                float   x,y
                [float  z]
        }
}
```

The last value (z) can be dropped for UVs when (Flags & ZCB_WDISCARDED).

To dequantize, perform the following operations:
float x = float(QuantizedX) * DequantCoeffX;
float y = float(QuantizedY) * DequantCoeffY;
float z = float(QuantizedZ) * DequantCoeffZ;

CAMS Chunk

```
CAMS        cameras chunk
udword      cameras chunk version number (CHUNK_CAMS_VER)

for each camera
{
        BasicInfo       basic info for current camera
        ubyte           1 for orthographic cameras, else 0
        float           Field-Of-View in degrees or width for orthographic cameras
        float           Near clipping plane
        float           Far clipping plane
        float           Distance to target
        udword          Horizon line display
        float           Environment near range
        float           Environment far range
        udword          Environment display
        udword          Manual clip
}
```

LITE Chunk

LITE	lights chunk
udword	lights chunk version number (CHUNK_LITE_VER)

for each light:
{

BasicInfo	basic info for current light
udword	Light type (a LType enum)
ubyte	1 for spotlights, else 0
ubyte	1 for directional lights, else 0
float	Light color red
float	Light color green
float	Light color blue
float	Intensity
float	Contrast
float	Diffuse soft
ubyte	1 for used light, else 0
ubyte	Affect diffuse
ubyte	Affect specular
ubyte	UseAttenNear
ubyte	AttenNearDisplay
ubyte	UseAtten
ubyte	ShowAtten
float	Near attenuation start
float	Near attenuation end
float	Attenuation start
float	Attenuation end (a.k.a. range)
ubyte	Decay type
float	Hotspot
float	Fallsize
float	Aspect
udword	Spot shape (a SpotShp enum)
udword	Overshoot
ubyte	Cone display
float	Distance to target
udword	Shadow type
udword	Absolute map bias
float	Raytrace bias
float	Map bias
float	Map range
udword	Map size
ubyte	CastShadows
float	Shadow density
float	Shadow color red
float	Shadow color green
float	Shadow color blue
ubyte	Light affects shadow

}

SHAP Chunk

```
SHAP        shapes chunk
udword      shapes chunk version number (CHUNK_SHAP_VER)

for each shape
{
        BasicInfo       basic info for current shape

        udword          the number of poly-lines / piecewise linear curve
        udword          material ID

        for each curve
        {
                udword          number of vertices
                bool            true for closed curves, else false for open curves

                for each vertex
                {
                        float           x,y,z
                }
        }
}
```

NB : for closed curves, remember you must connect the first vertex of the curve to the last one.

HELP Chunk

```
HELP        helpers chunk
udword      helpers chunk version number (CHUNK_HELP_VER)

for each helper
{
        BasicInfo       basic info for current helper
        udword          Helper's type
        ubyte           1 for group head, else 0

        if(type==HTYPE_GIZMO_BOX)
        {
                float   Length
                float   Width
                float   Height
        }
        else if(type==HTYPE_GIZMO_SPHERE)
        {
                float   Radius
                float   Hemi
        }
```

```
        else if(type==HTYPE_GIZMO_CYLINDER)
        {
                float   Radius
                float   Height
        }
}
```

TEXM Chunk

```
TEXM        textures chunk
udword      textures chunk version number (CHUNK_TEXM_VER)

for each texture
{
        name        the texture's name (a string of arbitrary length, null-terminated)
        udword      the texture ID

        ubyte       Code
        Code = 0=> the texture bitmap is not included in the file
        Code = 1=> the texture bitmap is included in the file
        Code = 2=> the texture bitmap is included in the file and quantized

        if the texture is included
        {
                if not quantized
                {
                        udword      Width
                        udword      Height
                        ubyte       1 if texture has an alpha channel, else 0

                        followed by Width*Height pixels.
                        Each pixel is :
                        -   a 32bits RGBA value if texture has an alpha channel
                        -   a 24bits RGB value otherwise.
                }
                else
                {
                        udword      Width
                        udword      Height
                        ubyte       1 if texture has an alpha channel, else 0

                        followed by a RGB palette (768 bytes)
                        followed by Width*Height color incides.
                        Each color index is a byte.
                }
        }
        Then :

        // Cropping values
```

```
        float       Offset U
        float       Offset V
        float       Scale U
        float       Scale V

        // The 4*3 MAX texture matrix
        float       m[0][0]
        float       m[0][1]
        float       m[0][2]
        float       m[1][0]
        float       m[1][1]
        float       m[1][2]
        float       m[2][0]
        float       m[2][1]
        float       m[2][2]
        float       m[3[0]
        float       m[3][1]
        float       m[3][2]
}
```

MATL Chunk

```
MATL        materials chunk
udword      materials chunk version number (CHUNK_MATL_VER)

for each material
{
        name        the material's name (a string of arbitrary length, null-terminated)
        udword      the material ID

        // Texture IDs
        udword      Ambient map ID
        udword      Diffuse map ID
        udword      Specular map ID
        udword      Shininess map ID
        udword      Shining Strength map ID
        udword      Self-Illumination map ID
        udword      Opacity map ID
        udword      Filter map ID
        udword      Bump map ID
        udword      Reflexion map ID
        udword      Refraction map ID
        udword      Displacement map ID

        // Blending coeffs
        float       Ambient coeff
        float       Diffuse coeff
        float       Specular coeff
        float       Shininess coeff
```

float	Shining Strength coeff
float	Self-Illumination coeff
float	Opacity coeff
float	Filter coeff
float	Bump coeff
float	Reflexion coeff
float	Refraction coeff
float	Displacement coeff

// Colors

float	Ambient color red
float	Ambient color green
float	Ambient color blue
float	Diffuse color red
float	Diffuse color green
float	Diffuse color blue
float	Specular color red
float	Specular color green
float	Specular color blue
float	Filter color red
float	Filter color green
float	Filter color blue

udword	Shading
ubyte	Soften
ubyte	Face Map
ubyte	Two Sided
ubyte	Wire
ubyte	Wire Units
ubyte	FalloffOut
udwod	Transparency type

float	Shininess
float	Shining strength
float	Self-illumination
float	Opacity
float	Opacity falloff
float	Wiresize
float	IOR

float	Bounce
float	Static friction
float	Sliding friction

// Cropping values

float	Offset U
float	Offset V
float	Scale U
float	Scale V

```
        // The 4*3 MAX texture matrix
        float       m[0][0]
        float       m[0][1]
        float       m[0][2]
        float       m[1][0]
        float       m[1][1]
        float       m[1][2]
        float       m[2][0]
        float       m[2][1]
        float       m[2][2]
        float       m[3][0]
        float       m[3][1]
        float       m[3][2]
}
```

CTRL Chunk

```
CTRL        controllers chunk
udword      controllers chunk version number (CHUNK_CTRL_VER)

for each controller
{
        String      controlled field (a string of arbitrary length, null-terminated)
        udword      controller ID
        udword      owner ID
        udword      owner type
        udword      controller type
        udword      controller mode

        if(controller mode==ZCB_CTRL_SAMPLES)
        {
                if(controller type==ZCB_CTRL_VERTEXCLOUD)
                {
                        udword      Number of vertices
                }

                udword      Number of samples
                udword      Sampling rate

                for each sample
                {
                        Sample data
                }
        }
        // NB : keyframes not recoded yet
}

Sample data can be :
A float if controller type == ZCB_CTRL_FLOAT
A vector if controller type == ZCB_CTRL_VECTOR
```

A quaternion if controller type == ZCB_CTRL_QUAT
A PR structure if controller type == ZCB_CTRL_PR
A PRS structure if controller type == ZCB_CTRL_PRS
A list of Nb vectors if controller type == ZCB_CTRL_VERTEXCLOUD

Controller mode can be :
ZCB_CTRL_SAMPLES
ZCB_CTRL_KEYFRAMES (Not recoded yet)

OwnerID links the controller to the controlled node (which can be a mesh, a camera, a light, etc). OwnerID maps the object's ID in the BasicInfo structure.

Possible controlled fields :

PRS	the PRS data
VISTRACK	the visibility track
VERTICES	the morph controller
FOVTRACK	the FOV controller
NearClip	the near clipping plane
FarClip	the far clipping plane
TDist	the distance to target
NearEnvRange	environment near range
FarEnvRange	environment far range
Color	a color
Intensity	color multiplier
Diffuse Soften	diffuse soften
Near Atten Start	attenuation near start
Near Atten End	attenuation near end
Far Atten Start	attenuation far start
Far Atten End	attenuation far end
Shadow Color	shadow color
Shadow Multiplier	shadow multiplier

(…more to come…)

MOVE Chunk

MOVE	motions chunk
udword	motions chunk version number (CHUNK_MOVE_VER)

for each motion
{

udword	owner character's ID
udword	Number of bones involved
udword	Number of virtual bones involved (obsolete)
ubyte	1 for local PRS, else 0
string	the motion's name (null-terminated string)
string	the motion's type (null-terminated string)

for each bone

```
        {
                udword      CSID (Character Studio ID)
                udword      Number of recorded frames

                for each frame
                {
                        float   x,y,z       translation vector
                        float   x,y,z,q     rotation quaternion
                }
        }
}
```

The BasicInfo structure

BasicInfo contains:

string	the object's name (a string of arbitrary length, null-terminated)
udword	the object's ID
udword	the parent's ID
udword	a possible link ID (ex : target nodes for cameras)
ubyte	1 if the object belongs to a group (in MAX), else 0
x,y,z	position in local space (relative to the parent's one) or absolute
x,y,z,w	rotation (quaternion) in local space or absolute
x,y,z	scale in local space or absolute
udword	a color for wireframe mode
ubyte	1 if the PRS is local, else 0
ubyte	1 if it has been converted to D3D, else 0
string	the user-defined properties (a string of arbitrary length, null-terminated)
x,y,z	pivot position
x,y,z,w	pivot rotation (quaternion)