OpenSAGE Documentation

Tim Jones, Stephan Vedder

Jul 15, 2020

Documentation

A place where we gather knowledge around the SAGE game engine. Feel free to contribute.

CHAPTER 1

File Formats

1.1 APT

1.1.1 Description

Apt files are flash based user interface files (based on the SWF file format created by Macromedia). Other than the SWF format the contents are splitted into multiple files, which are all in a single *.big* archive. Much of the information presented here was gathered from the apt2xml and xml2apt tools that are available for download here.

1.1.2 Constant Data

The constant variables that are used for the GUI are all stored in a `.const` file. The specification of that format is described here.

1.1.3 Apt Data

All parts of the GUI logic and display system are stored inside the `.apt` file. Before you are able to load the `.apt` file you are required to load the `.const` file to get the entry offset. The specification of that format can be found here.

1.2 BIG

1.2.1 Description

`.big` files are an archive format that was used in many game titles created by EA Studios.

1.2.2 Specification

Header

The header has a fixed size of 16 bytes, following the table below:

Offset	Bytes	Type	Name	Endianness
0	4	CHAR[]	FourCC	-
4	4	UINT32	Size	LE
8	4	UINT32	NumEntries	BE
12	4	UINT32	OffsetFirst	BE

- **FourCC**: Identifies the string as valid big archive. The string may either be "BIG4" or "BIGF", depending on the version.
- **Size**: The entire size of a big archive. Size of a single archive can not be greater than 2^32 bytes
- **NumEntries**: Number of files that were packed into this archive
- **OffsetFirst**: The offset inside the file to the first entry

List of entries

After the header the follows a list with **NumEntries** elements, each entry looks the following:

Offset	Bytes	Type	Name	Endianness
0	4	UINT32	EntryOffset	BE
4	4	UINT32	EntrySize	BE
8	4	CSTRING	EntryName	-

- **EntryOffset**: specified the start of this entry inside the file (in bytes)
- **EntrySize**: the size of the specified entry
- **EntryName**: the name of this entry, read as a nullterminated string. The maximum length is limited ny the Windows MAX_PATH (which is 260).

1.3 SAV

1.3.1 Description

`.sav` files, in the format described here, were used in C&C Generals and Zero Hour, and maybe later games too (to be investigated).

These files appear to store the complete state of the game engine at the time that the game was saved. This provides a valuable glimpse into the inner workings of the game engine.

Thanks to Omniblade for his help in figuring out this format.

1.3.2 Common Types

Strings in `.sav` files are one of two types:

- ASCII string: prefixed by the number of characters as a byte

- Unicode string: prefixed by the number of characters as a byte. Since each character is 2 bytes, the actual number of bytes to read is double this number

1.3.3 File Structure

A `.sav` file is a chunk-based binary format. The overall structure is:

- Variable number of ‘‘CHUNK‘‘s
- “SG_EOF” as an ASCII string (prefixed by length as a byte)

A chunk has this structure:

Type	Name
ASCII String	Chunk name
UINT32	Chunk size
BYTE	Version

The following are the chunks I have reverse-engineered so far.

CHUNK_GameState

Bytes	Type	Name
4	GAME_TYPE	Game type
n	ASCII String	Map path
2	UINT16	Year
2	UINT16	Month
2	UINT16	Day
2	UINT16	Day of week (Sunday = 0)
2	UINT16	Hour
2	UINT16	Minute
2	UINT16	Second
2	UINT16	Millisecond
n	ASCII String	Display name
n	ASCII String	Map file name
4	UINT32	Mission index

CHUNK_Campaign

Bytes	Type	Name
n	ASCII String	Side
n	ASCII String	Mission name
4	UINT32	?
4	UINT32	? maybe difficulty

If chunk version >= 5

Bytes	Type	Name
5	?	?

CHUNK_GameStateMap

Bytes	Type	Name
n	ASCII String	Map path 1
n	ASCII String	Map path 2
4	UINT32	? seems to be either 0 or 2
4	UINT32	length in bytes of map data
n	MAP	Byte-for-byte copy of current .map file
TODO		

CHUNK_TerrainLogic

TODO

CHUNK_TeamFactory

TODO

CHUNK_Players

TODO

CHUNK_GameLogic

TODO

CHUNK_ParticleSystem

TODO

CHUNK_Radar

TODO

CHUNK_ScriptEngine

TODO

CHUNK_SidesList

TODO

CHUNK_TacticalView

TODO

CHUNK_GameClient

TODO

CHUNK_InGameUI

TODO

CHUNK_Partition

TODO

CHUNK_TerrainVisual

TODO

CHUNK_GhostObject

TODO

1.3.4 Enumerations

GAME_TYPE

Value	Name
0	Skirmish
1	SinglePlayer

1.4 W3D

1.4.1 Description

`.w3d` files were used in the following games to store 3D models:

- C&C Generals
- Zero Hour
- BFME I
- BFME II

Later SAGE games (C&C3 and RA3) used an XML-based evolution of the w3d format called w3x.

1.4.2 References

- w3d2ply/w3d_file.h
- libw3d/types.hpp

1.4.3 Specification

Chunks

`.w3d` files can contain mesh data, animation data, bone transforms for placing meshes into a hierarchy and doing skinned animations, mesh bounding box information, and even particle emitter definitions. Skinned models are usually split over several files, with each animation chunk placed in a separate file. There is no technical need for this; it was presumably done for maintainability and / or runtime memory efficiency.

`.w3d` is a chunk-based binary format. A `.w3d` file can (and usually does) contain multiple chunks. Every chunk starts with a chunk header:

Offset	Bytes	Type	Name
0	4	UINT32	ChunkType
4	4	UINT32	ChunkSize

- **ChunkType**: A value from the *W3D_CHUNK_TYPE* enumeration.
- **ChunkSize**: MSB is 1 if the chunk contains sub-chunks, otherwise it's 0. Lower 31 bits are the chunk size, not including this chunk header.

Some chunks, depending on the value of **ChunkType** in the chunk header, have sub-chunks. If there are sub-chunks, then the Most Significant Bit (MSB) of the **ChunkSize** field will be `1`.

The following chunk type enumeration is not as complete as the references linked to above. This enumeration intentionally includes only the chunks that are actually used in SAGE games. Other chunks were used in previous W3D-based games, such as C&C Renegade. Note that the indentation denotes the chunk hierarchy, with nested chunk types appearing as sub-chunks of the parent chunk.

W3D_CHUNK_TYPE

Value	Name
0x0	*W3D_CHUNK_MESH*
0x2	*W3D_CHUNK_VERTICES*
0x3	*W3D_CHUNK_VERTEX_NORMALS*
0xC	*W3D_CHUNK_MESH_USER_TEXT*
0xE	*W3D_CHUNK_VERTEX_INFLUENCES*
0x1F	*W3D_CHUNK_MESH_HEADER3*
0x20	*W3D_CHUNK_TRIANGLES*
0x22	*W3D_CHUNK_VERTEX_SHADE_INDICES*
0x28	*W3D_CHUNK_MATERIAL_INFO*
0x29	*W3D_CHUNK_SHADERS*
0x2A	*W3D_CHUNK_VERTEX_MATERIALS*
0x2B	*W3D_CHUNK_VERTEX_MATERIAL*
0x2C	*W3D_CHUNK_VERTEX_MATERIAL_NAME*
0x2D	*W3D_CHUNK_VERTEX_MATERIAL_INFO*
0x2E	*W3D_CHUNK_VERTEX_MAPPER_ARGS0*
0x2F	*W3D_CHUNK_VERTEX_MAPPER_ARGS1*
0x30	*W3D_CHUNK_TEXTURES*
0x31	*W3D_CHUNK_TEXTURE*
0x32	*W3D_CHUNK_TEXTURE_NAME*
0x33	*W3D_CHUNK_TEXTURE_INFO*

Continued on next page

Table 1 – continued from previous page

Value	Name
0x38	*W3D_CHUNK_MATERIAL_PASS*
0x39	*W3D_CHUNK_VERTEX_MATERIAL_IDS*
0x3A	*W3D_CHUNK_SHADER_IDS*
0x3B	*W3D_CHUNK_DCG*
0x3C	*W3D_CHUNK_DIG*
0x3E	*W3D_CHUNK_SCG*
0x3F	*W3D_CHUNK_SHADER_MATERIAL_ID*
0x48	*W3D_CHUNK_TEXTURE_STAGE*
0x49	*W3D_CHUNK_TEXTURE_IDS*
0x4A	*W3D_CHUNK_STAGE_TEXCOORDS*
0x4B	*W3D_CHUNK_PER_FACE_TEXCOORD_IDS*
0x50	*W3D_CHUNK_SHADER_MATERIALS*
0x51	*W3D_CHUNK_SHADER_MATERIAL*
0x52	*W3D_CHUNK_SHADER_MATERIAL_HEADER*
0x53	*W3D_CHUNK_SHADER_MATERIAL_PROPERTY*
0x60	*W3D_CHUNK_TANGENTS*
0x61	*W3D_CHUNK_BITANGENTS*
0x80	*W3D_CHUNK_PS2_SHADERS*
0x90	*W3D_CHUNK_AABTREE*
0x91	*W3D_CHUNK_AABTREE_HEADER*
0x92	*W3D_CHUNK_AABTREE_POLYINDICES*
0x93	*W3D_CHUNK_AABTREE_NODES*
0x100	*W3D_CHUNK_HIERARCHY*
0x101	*W3D_CHUNK_HIERARCHY_HEADER*
0x102	*W3D_CHUNK_PIVOTS*
0x103	*W3D_CHUNK_PIVOT_FIXUPS*
0x200	*W3D_CHUNK_ANIMATION*
0x201	*W3D_CHUNK_ANIMATION_HEADER*
0x202	*W3D_CHUNK_ANIMATION_CHANNEL*
0x203	*W3D_CHUNK_BIT_CHANNEL*
0x280	*W3D_CHUNK_COMPRESSED_ANIMATION*
0x281	*W3D_CHUNK_COMPRESSED_ANIMATION_HEADER*
0x282	*W3D_CHUNK_COMPRESSED_ANIMATION_CHANNEL*
0x283	*W3D_CHUNK_COMPRESSED_BIT_CHANNEL*
0x284	*W3D_CHUNK_COMPRESSED_ANIMATION_MOTION_CHANNEL*
0x500	*W3D_CHUNK_EMITTER*
0x501	*W3D_CHUNK_EMITTER_HEADER*
0x502	*W3D_CHUNK_EMITTER_USER_DATA*
0x503	*W3D_CHUNK_EMITTER_INFO*
0x504	*W3D_CHUNK_EMITTER_INFOV2*
0x505	*W3D_CHUNK_EMITTER_PROPS*
0x509	*W3D_CHUNK_EMITTER_LINE_PROPERTIES*
0x510	*W3D_CHUNK_EMITTER_ROTATION_KEYFRAMES*
0x511	*W3D_CHUNK_EMITTER_FRAME_KEYFRAMES*
0x512	*W3D_CHUNK_EMITTER_BLUR_TIME_KEYFRAMES*
0x700	*W3D_CHUNK_HLOD*
0x701	*W3D_CHUNK_HLOD_HEADER*
0x702	*W3D_CHUNK_HLOD_LOD_ARRAY*
0x703	*W3D_CHUNK_HLOD_SUB_OBJECT_ARRAY_HEADER*

Continued on next page

Table 1 – continued from previous page

Value	Name
0x704	*W3D_CHUNK_HLOD_SUB_OBJECT*
0x705	*W3D_CHUNK_HLOD_AGGREGATE_ARRAY*
0x706	*W3D_CHUNK_HLOD_PROXY_ARRAY*
0x740	*W3D_CHUNK_BOX*
0xC00	*W3D_CHUNK_VERTICES_2*
0xC01	*W3D_CHUNK_VERTEX_NORMALS_2*

Chunks and sub-chunks appear in `.w3d` files in the following hierarchy:

- *W3D_CHUNK_MESH*
  - *W3D_CHUNK_VERTICES*
  - *W3D_CHUNK_VERTICES_2*
  - *W3D_CHUNK_VERTEX_NORMALS*
  - *W3D_CHUNK_VERTEX_NORMALS_2*
  - *W3D_CHUNK_MESH_USER_TEXT*
  - *W3D_CHUNK_VERTEX_INFLUENCES*
  - *W3D_CHUNK_MESH_HEADER3*
  - *W3D_CHUNK_TRIANGLES*
  - *W3D_CHUNK_VERTEX_SHADE_INDICES*
  - *W3D_CHUNK_MATERIAL_INFO*
  - *W3D_CHUNK_SHADERS*
  - *W3D_CHUNK_VERTEX_MATERIALS*
    - *W3D_CHUNK_VERTEX_MATERIAL*
      - *W3D_CHUNK_VERTEX_MATERIAL_NAME*
      - *W3D_CHUNK_VERTEX_MATERIAL_INFO*
      - *W3D_CHUNK_VERTEX_MAPPER_ARGS0*
      - *W3D_CHUNK_VERTEX_MAPPER_ARGS1*
  - *W3D_CHUNK_TEXTURES*
    - *W3D_CHUNK_TEXTURE*
      - *W3D_CHUNK_TEXTURE_NAME*
      - *W3D_CHUNK_TEXTURE_INFO*
  - *W3D_CHUNK_MATERIAL_PASS*
    - *W3D_CHUNK_VERTEX_MATERIAL_IDS*
    - *W3D_CHUNK_SHADER_IDS*
    - *W3D_CHUNK_DCG*
    - *W3D_CHUNK_DIG*
    - *W3D_CHUNK_SCG*
    - *W3D_CHUNK_SHADER_MATERIAL_ID*

W3D_CHUNK_MESH

This is the root mesh definition chunk. It is a container chunk that can contain these sub-chunks:

- W3D_CHUNK_MESH_HEADER3
- W3D_CHUNK_VERTICES
- W3D_CHUNK_VERTEX_NORMALS
- W3D_CHUNK_MESH_USER_TEXT
- W3D_CHUNK_VERTEX_INFLUENCES
- W3D_CHUNK_TRIANGLES
- W3D_CHUNK_VERTEX_SHADE_INDICES
- W3D_CHUNK_MATERIAL_INFO
- W3D_CHUNK_SHADERS
- W3D_CHUNK_VERTEX_MATERIALS
- W3D_CHUNK_TEXTURES
- W3D_CHUNK_MATERIAL_PASS
- W3D_CHUNK_TANGENTS
- W3D_CHUNK_BITANGENTS
- W3D_CHUNK_SHADER_MATERIALS
- W3D_CHUNK_PS2_SHADERS
- W3D_CHUNK_AABTREE
- W3D_CHUNK_VERTICES_2
- W3D_CHUNK_VERTEX_NORMALS_2

W3D_CHUNK_MESH_HEADER3

The mesh header contains general info about the mesh.

Offset	Bytes	Type	Name
0	4	UINT32	Version
4	4	UINT32	MeshFlags
8	16	CHAR[16]	MeshName
24	16	CHAR[16]	ContainerName
40	4	UINT32	NumTriangles
44	4	UINT32	NumVertices
48	4	UINT32	NumMaterials
52	4	UINT32	NumDamageStages
56	4	UINT32	SortLevel
60	4	UINT32	PrelitVersion
64	4	UINT32	FutureCounts
68	4	UINT32	VertexChannels
72	4	UINT32	FaceChannels
76	4	W3D_VECTOR3	BoundingBoxMin
88	4	W3D_VECTOR3	BoundingBoxMax
100	4	W3D_VECTOR3	BoundingSphereCenter
112	4	FLOAT32	BoundingSphereRadius

- **MeshFlags**: bitwise-or'd collection of *W3D_MESH_FLAGS* values.
- **VertexChannels**: bitwise-or'd collection of *W3D_VERTEX_CHANNELS* values.
- **FaceChannels**: bitwise-or'd collection of *W3D_FACE_CHANNELS* values.

W3D_VERTEX_CHANNELS

Value	Name	Description
0x1	Location	Object-space location of the vertex
0x2	Normal	Object-space normal for the vertex
0x4	TexCoord	Texture coordinate
0x8	Color	Vertex color
0x10	BoneId	Per-vertex bone id for skins

W3D_FACE_CHANNELS

Value	Name	Description
0x1	Face	Basic face info, W3dTriStruct...

W3D_CHUNK_VERTICES

Array of vertices.

Offset	Bytes	Type	Name
0	12 * N	W3D_VECTOR3[]	Vertices

N is the number of vertices specified in the *W3D_CHUNK_MESH_HEADER3* chunk.

W3D_CHUNK_VERTEX_NORMALS

Array of normals.

Offset	Bytes	Type	Name
0	12 * N	W3D_VECTOR3[]	Normals

N is the number of vertices specified in the *W3D_CHUNK_MESH_HEADER3* chunk.

W3D_CHUNK_MESH_USER_TEXT

Text from the MAX comment field (null-terminated string).

Offset	Bytes	Type	Name
0	ChunkSize	CHAR[]	UserText

W3D_CHUNK_VERTEX_INFLUENCES

Mesh deformation vertex connections.

Offset	Bytes	Type	Name
0	8 * N	W3D_VERTEX_INFLUENCE[]	VertexInfluences

N is the number of vertices specified in the *W3D_CHUNK_MESH_HEADER3* chunk.

W3D_VERTEX_INFLUENCE

Offset	Bytes	Type	Name
0	2	UINT16	BoneIndex
2	6	UINT8[]	[Padding]

TODO: Does BFME have a second bone index, and bone weights?

W3D_CHUNK_TRIANGLES

Offset	Bytes	Type	Name
0	32 * N	W3D_TRIANGLE	Triangles

N is the number of triangles specified in the *W3D_CHUNK_MESH_HEADER3* chunk.

W3D_TRIANGLE

Offset	Bytes	Type	Name
0	4	UINT32	VertexIndex0
4	4	UINT32	VertexIndex1
8	4	UINT32	VertexIndex2
12	4	UINT32	SurfaceType
16	12	W3D_VECTOR3	Normal
28	4	FLOAT32	Distance

- **SurfaceType**: A value from the *W3D_SURFACE_TYPE* enumeration.

W3D_SURFACE_TYPE

Value	Name
0	LightMetal
1	HeavyMetal
2	Water
3	Sand
4	Dirt
5	Mud
6	Grass
7	Wood
8	Concrete
9	Flesh
10	Rock
11	Snow
12	Ice
13	Default
14	Glass
15	Cloth
16	TiberiumField
17	FoliagePermeable
18	GlassPermeable
19	IcePermeable
20	ClothPermeable
21	Electrical
22	Flammable
23	Steam
24	EletricalPermeable
25	FlammablePermeable
26	SteamPermeable
27	WaterPermeable
28	TiberiumWater
29	TiberiumWaterPermeable
30	UnderwaterDirt
31	UnderwaterTiberiumDirt

W3D_CHUNK_VERTEX_SHADE_INDICES

Shade indexes for each vertex.

The meaning of "shade indexes" is unknown.

Offset	Bytes	Type	Name
0	4 * N	UINT32	ShadeIndices

`N` is the number of vertices specified in the *W3D_CHUNK_MESH_HEADER3* chunk.

W3D_CHUNK_MATERIAL_INFO

Declares the number of material passes, vertex materials, shaders, and textures that will be found in subsequent chunks.

Offset	Bytes	Type	Name
0	4	UINT32	PassCount
4	4	UINT32	VertexMaterialCount
8	4	UINT32	ShaderCount
12	4	UINT32	TextureCount

- **PassCount**: How many material passes this render object uses
- **VertexMaterialCount**: How many vertex materials are used
- **ShaderCount**: How many shaders are used
- **TextureCount**: How many textures are used

W3D_CHUNK_SHADERS

Container chunk for an array of *W3D_SHADER* structures. The number of shaders is contained in the **ShaderCount** field in the *W3D_CHUNK_MATERIAL_INFO* chunk.

W3D_SHADER

Offset	Bytes	Type	Name
0	1	UINT8	DepthCompare
1	1	UINT8	DepthMask
2	1	UINT8	ColorMask
3	1	UINT8	DestBlend
4	1	UINT8	FogFunc
5	1	UINT8	PrimaryGradient
6	1	UINT8	SecondaryGradient
7	1	UINT8	SrcBlend
8	1	UINT8	Texturing
9	1	UINT8	DetailColorFunc
10	1	UINT8	DetailAlphaFunc
11	1	UINT8	ShaderPreset
12	1	UINT8	AlphaTest
13	1	UINT8	PostDetailColorFunc
14	1	UINT8	PostDetailAlphaFunc
15	1	UINT8	[Padding]

- **DepthCompare**: A value from the *W3D_SHADER_DEPTH_COMPARE* enumeration.
- **DepthMask**: A value from the *W3D_SHADER_DEPTH_MASK* enumeration.
- **ColorMask**: Now obsolete and ignored.
- **DestBlend**: A value from the *W3D_SHADER_DEST_BLEND_FUNC* enumeration.
- **FogFunc**: Now obsolete and ignored.
- **PrimaryGradient**: A value from the *W3D_SHADER_PRIMARY_GRADIENT* enumeration.
- **SecondaryGradient**: A value from the *W3D_SHADER_SECONDARY_GRADIENT* enumeration.
- **SrcBlend**: A value from the *W3D_SHADER_SRC_BLEND_FUNC* enumeration.
- **Texturing**: A value from the *W3D_SHADER_TEXTURING* enumeration.
- **DetailColorFunc**: A value from the *W3D_SHADER_DETAIL_COLOR_FUNC* enumeration.
- **DetailAlphaFunc**: A value from the *W3D_SHADER_DETAIL_ALPHA_FUNC* enumeration.
- **ShaderPreset**: Now obsolete and ignored.
- **AlphaTest**: A value from the *W3D_SHADER_ALPHA_TEST* enumeration.
- **PostDetailColorFunc**: A value from the *W3D_SHADER_DETAIL_COLOR_FUNC* enumeration.
- **PostDetailAlphaFunc**: A value from the *W3D_SHADER_DETAIL_ALPHA_FUNC* enumeration.

W3D_SHADER_DEPTH_COMPARE

Value	Name	Description
0	PassNever	Pass never (i.e. always fail depth comparison test)
1	PassLess	Pass if incoming less than stored
2	PassEqual	Pass if incoming equal to stored
3	PassLEqual	Pass if incoming less than or equal to stored (default)
4	PassGreater	Pass if incoming greater than stored
5	PassNotEqual	Pass if incoming not equal to stored
6	PassGEqual	Pass if incoming greater than or equal to stored
7	PassAlways	Pass always

W3D_SHADER_DEPTH_MASK

Value	Name	Description
0	WriteDisable	Disable depth buffer writes
1	WriteEnable	Enable depth buffer writes (default)

W3D_SHADER_DEST_BLEND_FUNC

Value	Name	Description
0	Zero	Destination pixel doesn't affect blending (default)
1	One	Destination pixel added unmodified
2	SrcColor	Destination pixel multiplied by fragment RGB components
3	OneMinusSrcColor	Destination pixel multiplied by one minus (i.e. inverse) fragment RGB components
4	SrcAlpha	Destination pixel multiplied by fragment alpha component
5	OneMinusSrcAlpha	Destination pixel multiplied by fragment inverse alpha
6	SrcColorPreFog	Destination pixel multiplied by fragment RGB components prior to fogging

W3D_SHADER_PRIMARY_GRADIENT

Value	Name	Description
0	Disable	Disable primary gradient (same as OpenGL 'decal' texture blend)
1	Modulate	Modulate fragment ARGB by gradient ARGB (default)
2	Add	Add gradient RGB to fragment RGB, copy gradient A to fragment A
3	BumpEnvMap	Environment-mapped bump mapping

W3D_SHADER_SECONDARY_GRADIENT

Value	Name	Description
0	Disable	Don't draw secondary gradient (default)
1	Enable	Add secondary gradient RGB to fragment RGB

W3D_SHADER_SRC_BLEND_FUNC

Value	Name	Description
0	Zero	Fragment not added to color buffer
1	One	Fragment added unmodified to color buffer (default)
2	SrcAlpha	Fragment RGB components multiplied by fragment A
3	OneMinusSrcAlpha	Fragment RGB components multiplied by fragment inverse (one minus) A

W3D_SHADER_TEXTURING

Value	Name	Description
0	Disable	No texturing (treat fragment initial color as 1,1,1,1) (default)
1	Enable	Enable texturing

W3D_SHADER_DETAIL_COLOR_FUNC

Value	Name	Description
0	Disable	Local (default)
1	Detail	Other
2	Scale	Local * Other
3	InvScale	~(~Local * ~Other) = Local + (1-Local)*Other
4	Add	Local + Other
5	Sub	Local - Other
6	SubR	Other - Local
7	Blend	(LocalAlpha)*Local + (~LocalAlpha)*Other
8	DetailBlend	(OtherAlpha)*Local + (~OtherAlpha)*Other

W3D_SHADER_DETAIL_ALPHA_FUNC

Value	Name	Description
0	Disable	Local (default)
1	Detail	Other
2	Scale	Local * Other
3	InvScale	~(~Local * ~Other) = Local + (1-Local)*Other

W3D_SHADER_ALPHA_TEST

Value	Name	Description
0	Disable	Disable alpha testing (default)
1	Enable	Enable alpha testing

W3D_CHUNK_VERTEX_MATERIALS

Wraps the vertex materials.

W3D_CHUNK_VERTEX_MATERIAL

Vertex material wrapper.

W3D_CHUNK_VERTEX_MATERIAL_NAME

Vertex material name (null-terminated string)

W3D_CHUNK_VERTEX_MATERIAL_INFO

Vertex material info.

W3D_CHUNK_VERTEX_MAPPER_ARGS0

Arguments for the first stage mapping (null-terminated, line-break-separated string).

W3D_CHUNK_VERTEX_MAPPER_ARGS1

Arguments for the second stage mapping (null-terminated, line-break-separated string).

W3D_CHUNK_TEXTURES

Wraps all of the texture info.

W3D_CHUNK_TEXTURE

Wraps a texture definition.

W3D_CHUNK_TEXTURE_NAME

Texture filename (null-terminated string).

W3D_CHUNK_TEXTURE_INFO

Optional texture info.

W3D_CHUNK_MATERIAL_PASS

Wraps the information for a single material pass.

W3D_CHUNK_VERTEX_MATERIAL_IDS

Single or per-vertex array of UINT32 vertex material indices.

W3D_CHUNK_SHADER_IDS

Single or per-triangle array of UINT32 shader indices.

W3D_CHUNK_DCG

Per-vertex diffuse color values.

W3D_CHUNK_DIG

Per-vertex diffuse illumination values.

W3D_CHUNK_SCG

Per-vertex specular color values.

W3D_CHUNK_SHADER_MATERIAL_ID

Index into the array of shader materials in the **'W3D_SHADER_MATERIALS'_** chunk.

W3D_CHUNK_TEXTURE_STAGE

Wrapper around a texture stage.

W3D_CHUNK_TEXTURE_IDS

Single or per-triangle array of UINT32 texture indices.

W3D_CHUNK_STAGE_TEXCOORDS

Per-vertex texture coordinates.

W3D_CHUNK_PER_FACE_TEXCOORD_IDS

Indices to *W3D_CHUNK_STAGE_TEXCOORDS*.

W3D_CHUNK_SHADER_MATERIALS

Wrapper around an array of shader materials.

W3D_CHUNK_SHADER_MATERIAL

Stores material properties for use in conjunction with a specific pixel shader.

W3D_CHUNK_SHADER_MATERIAL_HEADER

Stores the shader filename.

W3D_CHUNK_SHADER_MATERIAL_PROPERTY

A single shader material property with a type and value.

W3D_CHUNK_TANGENTS

Array of tangent vectors.

W3D_CHUNK_BITANGENTS

Array of bitangent vectors.

W3D_CHUNK_PS2_SHADERS

Shader info specific to the PlayStation 2.

W3D_CHUNK_AABTREE

Axis-Aligned Box Tree for hierarchical polygon culling

W3D_CHUNK_AABTREE_HEADER

Catalog of the contents of the AABTree.

W3D_CHUNK_AABTREE_POLYINDICES

Array of UINT32 polygon indices with count=mesh.PolyCount.

W3D_CHUNK_AABTREE_NODES

Array of W3dMeshAABTreeNode's with count=aabheader.NodeCount.

W3D_CHUNK_HIERARCHY

Hierarchy tree definition.

W3D_CHUNK_HIERARCHY_HEADER

W3D_CHUNK_PIVOTS

W3D_CHUNK_PIVOT_FIXUPS

Only needed by the exporter. Doesn't seem to be used at runtime.

W3D_CHUNK_ANIMATION

Hierarchy animation data.

W3D_CHUNK_ANIMATION_HEADER

W3D_CHUNK_ANIMATION_CHANNEL

Channel of vectors.

W3D_CHUNK_BIT_CHANNEL

Channel of boolean values (e.g. visibility).

W3D_CHUNK_COMPRESSED_ANIMATION

Compressed hierarchy animation data.

W3D_CHUNK_COMPRESSED_ANIMATION_HEADER

Describes playback rate, number of frames, and type of compression.

W3D_CHUNK_COMPRESSED_ANIMATION_CHANNEL

Compressed channel, format dependent on type of compression.

W3D_CHUNK_COMPRESSED_BIT_CHANNEL

Compressed bit stream channel, format dependent on type of compression.

W3D_CHUNK_COMPRESSED_ANIMATION_MOTION_CHANNEL

Added in BFME II.

W3D_CHUNK_EMITTER

Description of a particle emitter.

W3D_CHUNK_EMITTER_HEADER

General information such as name and version.

W3D_CHUNK_EMITTER_USER_DATA

User-defined data that specific loaders can switch on.

W3D_CHUNK_EMITTER_INFO

Generic particle emitter definition.

W3D_CHUNK_EMITTER_INFOV2

General particle emitter definition (version 2.0).

W3D_CHUNK_EMITTER_PROPS

Key-frameable properties.

W3D_CHUNK_EMITTER_LINE_PROPERTIES

Line properties, used by line rendering mode.

W3D_CHUNK_EMITTER_ROTATION_KEYFRAMES

Rotation keys for the particles.

W3D_CHUNK_EMITTER_FRAME_KEYFRAMES

Frame keys (u-v based frame animation).

W3D_CHUNK_EMITTER_BLUR_TIME_KEYFRAMES

Length of tail for line groups.

W3D_CHUNK_HLOD

Description of an HLOD object.

W3D_CHUNK_HLOD_HEADER

General information such as name and version.

W3D_CHUNK_HLOD_LOD_ARRAY

Wrapper around the array of objects for each level of detail.

W3D_CHUNK_HLOD_SUB_OBJECT_ARRAY_HEADER

Info on the objects in this level of detail array.

W3D_CHUNK_HLOD_SUB_OBJECT

An object in this level of detail array.

W3D_CHUNK_HLOD_AGGREGATE_ARRAY

Array of aggregates, contains W3D_CHUNK_SUB_OBJECT_ARRAY_HEADER and W3D_CHUNK_SUB_OBJECT_ARRAY.

W3D_CHUNK_HLOD_PROXY_ARRAY

Array of proxies, used for application-defined purposes, provides a name and a bone.

W3D_CHUNK_BOX

Defines an collision box render object (W3dBoxStruct).

W3D_CHUNK_VERTICES_2

Unknown purpose - added in BFME. Contained vertices are very similar to, but not the same as, the vertices in W3D_CHUNK_VERTICES.

W3D_CHUNK_VERTEX_NORMALS_2

Unknown purpose - added in BFME. Contained normals are very similar to, but not the same as, the normals in W3D_CHUNK_NORMALS.

1.4.4 Structures

W3D_VECTOR3

Offset	Bytes	Type	Name
0	4	FLOAT32	X
4	4	FLOAT32	Y
8	4	FLOAT32	Z

1.4.5 Enumerations

W3D_MESH_FLAGS

Value	Name	Description
0x0	None	Plain old normal mesh
0x1	CollisionBox	(obsolete as of 4.1) Mesh is a collision box (should be 8 verts, should be hidden, etc
0x2	Skin	(obsolete as of 4.1) Skin mesh
0x4	Shadow	(obsolete as of 4.1) Intended to be projected as a shadow
0x8	Aligned	(obsolete as of 4.1) Always aligns with camera
0xFF0	CollisionTypeMask	Mask for the collision type bits
0x10	CollisionTypePhysical	Physical collisions
0x20	CollisionTypeProjectile	Projectiles (rays) collide with this
0x40	CollisionTypeVis	Vis rays collide with this mesh

Table 3 – continued from previous page

Value	Name	Description
0x80	CollisionTypeCamera	Camera rays/boxes collide with this mesh
0x100	CollisionTypeVehicle	Vehicles collide with this mesh (and with physical collision meshes)
0x1000	Hidden	This mesh is hidden by default
0x2000	TwoSided	Render both sides of this mesh
0x4000	ObsoleteLightMapped	Obsolete lightmapped mesh
0x8000	CastShadow	This mesh casts shadows. Retained for backwards compatibility - use W3D_MESH
0xFF0000	GeometryTypeMask	(introduced with 4.1)
0x0	GeometryTypeNormal	(4.1+) Normal mesh geometry
0x10000	GeometryTypeCameraAligned	(4.1+) camera aligned mesh
0x20000	GeometryTypeSkin	(4.1+) skin mesh
0x30000	ObsoleteGeometryTypeShadow	(4.1+) shadow mesh OBSOLETE!
0x40000	GeometryTypeAAbox	(4.1+) aabox OBSOLETE!
0x50000	GeometryTypeOBBox	(4.1+) obbox OBSOLETE!
0x60000	GeometryTypeCameraOriented	(4.1+) Camera oriented mesh (points *towards* camera)
0xF000000	PrelitMask	(4.2+)
0x1000000	PrelitUnlit	Mesh contains an unlit material chunk wrapper
0x2000000	PrelitVertex	Mesh contains a precalculated vertex-lit material chunk wrapper
0x4000000	PrelitLightMapMultiPass	Mesh contains a precalculated multi-pass lightmapped material chunk wrapper
0x8000000	PrelitLightMapMultiTexture	Mesh contains a precalculated multi-texture lightmapped material chunk wrapper
0x10000000	Shatterable	This mesh is shatterable
0x20000000	NPatchable	It is okay to NPatch this mesh

1.5 WAK

1.5.1 Description

`.wak` files were used in C&C Generals and Zero Hour to store ocean and pond wave definitions. In later SAGE games, this data became part of the `.map` format.

Not all maps have accompanying `.wak` files, but those that require wave movement do.

1.5.2 File Structure

The file has a variable number of entries. The number of entries is stored at the end of the file.

Offset	Bytes	Type	Name
0	NumEntries * 20	*WAVE_ENTRY*[NumEntries]	Entries
NumEntries * 20	4	UINT32	NumEntries

- **Entries**: See *WAVE_ENTRY* structure.
- **NumEntries**: Number of entries. Don't ask me why this is at the end of the file, not the beginning... Thankfully WAVE_ENTRY is a fixed-length structure.

WAVE_ENTRY

Offset	Bytes	Type	Name
0	4	UINT32	StartX
4	4	UINT32	StartY
8	4	UINT32	EndX
12	4	UINT32	EndY
16	4	*WAVE_TYPE*	WaveType

- **StartX** and **StartY**: Specify the start position of the wave.
- **EndX** and **EndY**: Specify the end position of the wave.
- **WaveType**: Type of the wave. See *WAVE_TYPE* enumeration.

1.5.3 Enumerations

WAVE_TYPE

Value	Name
0	Pond
1	Ocean
2	CloseOcean
3	CloseOceanDouble
4	Radial

CHAPTER 2

Systems

2.1 Scripting

Note: This document describes how the scripting system works in Command & Conquer: Generals - Zero Hour. Some details can be different in newer SAGE titles, and some features are not covered at all.

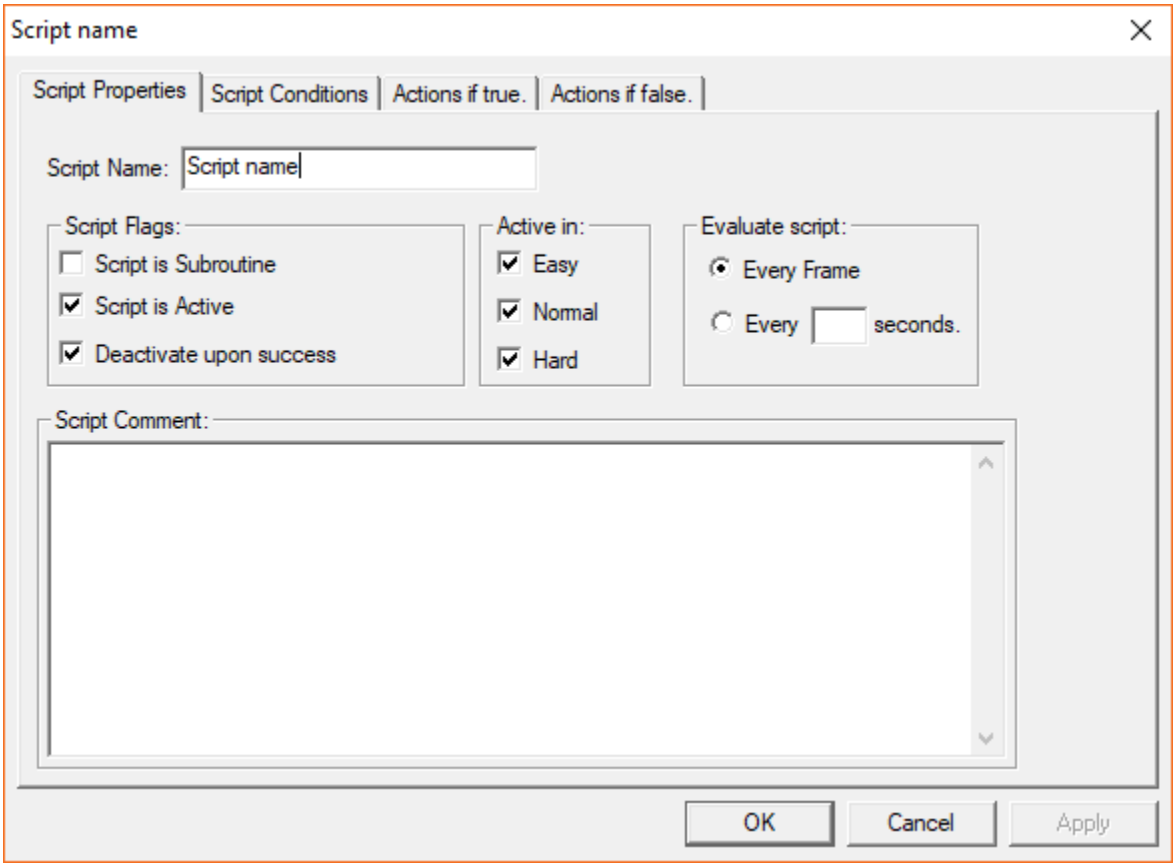
SAGE has a built-in scripting system, which is used for scripted sequences, cinematics and AI. Scripts are usually embedded within `.map` files, but they can also be stored in `.scb` files.

The scripting language is fairly high level, but at the same time it's rather limited in terms of features since it has been specialised for RTS game scripting. It is written using the graphical editor included in WorldBuilder.

The following information is based on black box testing done with World Builder and Zero Hour, along with the official manuals for various versions of WorldBuilder.

2.1.1 Basics

The game expects the scripting system to run at about 30hz at all times. Some options and actions refer to *frames*, which means game update cycles and not actual rendered frames. (This is because - unfortunately enough - SAGE has a 30fps frame limit by default, and the game speed is tied to the framerate.)



A script consists of the following parts:

- Properties
  - Name
  - Is active
    - A boolean flag which enables or disables the script.
  - Deactivate upon success
    - If the script is executed succesfully and this flag is set, the script will disable itself.
  - Is subroutine
    - Changes how the script is triggered. See *Subroutines*.
  - Active in `<difficulty level>`
  - Execution frequency ("Evaluate script" in WorldBuilder)
    - Every frame
    - Every `<number>` seconds
      - The script will run on the first frame it can run, and then every `<number>` seconds.
      - Disabling and re-enabling the script will not reset the interval timer.
- Conditions
  - Conditions are predicate functions which query the game state and return a boolean value. For example, they can check flags and counters, which units or technologies each player has, and so on.

  - They are combined into a single value with the AND operation - in other words, the value of the condition is true only if every subcondition is true. OR operations can be added manually.

- Actions
  - Actions are commands which modify the game state. They can do a lot of different things, like spawn units, modify variables and run *Subroutines* and *Sequential scripts*.
  - Every script has an *if true* and an *if false* section, which is a list of actions. Which section gets run depends on the value of the condition variable.
  - Actions *that take longer than a frame* will not block other actions from starting. They are started in a sequential order during the same frame, and executed to completion in parallel.

Scripts are executed in the order defined in WorldBuilder. Conditions are only checked once per script per frame. In practise this means that if a script has a set of conditions which are fulfilled during the same frame by an another script, the script will only be executed if it's positioned after the other script.

If a script is disabled during execution (using the "Disable script" -action), it will still run to completion.

See the graph below for a compact description of how the script evaluation cycle works.

It should be noted that *Sequential scripts* work very differently from normal scripts, and a lot of this information doesn't apply to them.

2.1.2 Flags, counters and timers

The scripting system supports three kinds of variables:

- Flags
  - Flags are boolean variables, which can be toggled on and off.
  - Generals and Zero Hour have about two dozen global flags, which are only used to make the shell map react to the main menu.
- Counters
  - Counters are 32-bit signed integer variables, which can be modified and compared in various ways.
  - They overflow and underflow.
- Timers
  - Timers are automatically decrementing counters with a configurable starting value.
  - Timers tick down towards zero, and stay at `-1` when they expire.
  - All counter actions can be used on timers and vice versa.
  - The scripting system has separate actions for creating second timers and frame timers, but internally there's no difference.
  - The remaining time is stored as the number of remaining frames.
    - This means that the smallest unit of time that can be measured with a timer is one frame (~33 milliseconds).

2.1.3 Subroutines

Like most other programming languages, map scripts support subroutines. They are created just like any other script, except they are explicitly marked as subroutines with the "Is subroutine" -checkbox.

Unlike normal scripts, subroutines are never executed automatically. Instead, they are started from other scripts with the "Run subroutine script" action. Conditions are checked on each call. They cannot take arguments, so data must be passed via global variables.

Testing has shown that if a subroutine's execution frequency is set to anything other than "Every frame", calling the subroutine will not do anything. There's no simple logical explanation for this, so it could be a bug in the engine.

2.1.4 Sequential scripts

On the surface sequential scripts seem very similar to normal scripts - in fact, they are just normal scripts and they don't even have a specific configuration property like subroutines (until later SAGE games). However, sequential scripts are executed very differently from other scripts:

- As the name implies, actions are run sequentially from top to bottom, during multiple frames if necessary. If an action takes multiple frames to execute, actions below it will wait until it's done.
- Like subroutines, sequential scripts have to be started manually. A sequential script is not necessarily started immediately, but instead queued for later execution.
- A sequential script is always associated with a *team*. This is presumably because they were primarily intended for AI programming.
  - Every team has an *execution queue*, which contains all scripted and player-issued commands for the team. The queue is shared by all units in the team, so if any unit is performing a task (such as moving and guarding) no scripted actions will be taken from the queue.
  - If the team has no units or buildings, the script will not be queued and therefore will not execute even if some units are added to the team later.
  - The team is bound to an implicit variable, named `<this team>` in WorldBuilder, which can be used in actions that take a team.
    - If `<this team>` is used outside of a sequential script, it will be effectively `null` and actions that try to use it do nothing.
    - The variable is **not** automatically propagated in subroutine calls.
  - A team can only run one sequential script at a time.
  - A sequential script can be run by multiple teams at the same time.
- Sequential scripts can be run in a loop.
- Sequential scripts can be stopped with an action.
  - If a sequential script stops itself, it will stop executing further actions immediately.
  - If a sequential script is stopped by an another script, it will finish the currently executing action and then stop.
  - Both cases also clear the team's execution queue of all *scripted* commands. Player-issued commands are not removed.
- "Is active" has no effect.
- "Is subroutine" has no effect.

- "Deactivate upon success" does nothing.
- Conditions are never checked.

Sequential scripts seem to have the same execution frequency bug as subroutines do. These seems to be a sequential script specific engine bug as well: If a sequential script is queued for a team that has no units, it normally does nothing. However, if that team is the player's default team (named `teamplayer001`, `teamplayer002`, ...) it will execute the first action of the script and then do nothing until the team gets some units.

2.1.5 Edge cases and trivia

- If a sequential script is performing a "move to waypoint" -action and the player can control the unit(s), they can change the target location without interrupting the rest of the actions. The script continues normally after the unit(s) stop moving.
  - If the player queues multiple movement targets (with Alt+Click), the actions will continue when the unit(s) have gone through every target.

CHAPTER 3

Behaviors

3.1 SlowDeathBehavior Module

3.1.1 ProbabilityModifier

`ProbabilityModifier` is used to choose between several ‘‘SlowDeathBehavior‘‘s when more than one is used on the same object.

3.1.2 DeathFlags

These flags are added to both model condition and object status.

3.1.3 Flinging

When an object dies, and `FlingForce` is greater than zero, a force is added to the object's `PhysicsBehavior` based on the following properties:

- `FlingForce`
- `FlingForceVariance`
- `FlingPitch`
- `FlingPitchVariance`

3.1.4 Sinking

When an object dies, it first waits until `SinkDelay`, plus or minus a random duration based on `SinkDelayVariance`, has elapsed.

Then it starts to sink at the rate specified by `SinkRate`.

3.1.5 Fading

When an object dies, it first waits until `FadeDelay` has elapsed.

Then it starts to fade out at the rate specified by `FadeTime`.

3.1.6 Decaying

After `DecayBeginTime`, the `DECAY` model condition flag is added to the object.

3.1.7 Destruction

When an object dies, it enters the `INITIAL` phase of destruction.

A destruction duration is calculated, based on `DestructionDelay` plus or minus a random duration based on `DestructionDelayVariance`.

- `INITIAL`: Any `FX`, `OCL`, or `Weapon` specified for the `INITIAL` phase will be triggered. After the first half of the calculated destruction delay has elapsed, the phase changes to `MIDPOINT`.
- `MIDPOINT`: Any `FX`, `OCL`, or `Weapon` specified for the `MIDPOINT` phase will be triggered. After the second half of the calculated destruction delay has elapsed, the phase changes to `FINAL`.
- `FINAL`: Any `FX`, `OCL`, or `Weapon` specified for the `FINAL` phase will be triggered. Then the object will be destroyed.