Dungeon Keeper 2 File Format Guide

George Gensure

September 9, 2015

Contents

Contents **i**

1 Introduction **1**
1.1 Copyright 1
1.2 License 1

2 Type Reference **2**
2.1 Primitive Types 2
2.2 Structures 2
2.3 Enumerations 3
2.4 Byte Expansions 3

3 Text Assets **4**
3.1 BFMU - MultiByte to Unicode 4
3.2 BFUM - Unicode to MultiByte 5
3.3 BFMB - Multi Byte 6
3.4 BFST - Strings 6
3.5 BF4F - Fonts 7

4 WAD World Assets Directory **10**

5 KMF Keeper Mesh Format **11**
5.1 IFF Hierarchy 11
5.2 Types Reference 12
5.3 KMF Structures 12

6 KCS Keeper Camera Sweep **23**
6.1 KCS Structures 23

Chapter 1

Introduction

This document describes the format of binary files associated with Dungeon Keeper 2.

1.1 Copyright

©2015 George Gensure

1.2 License

This work is licensed under the Creative Commons Attribution-ShareAlike 4.0 International License. To view a copy of this license, visit http://creativecommons.org/licenses/by-sa/4.0/.

Chapter 2

Type Reference

2.1 Primitive Types

Type	Name
[u]int8_t	1 byte [un]signed integer
[u]int16_t	2 byte [un]signed integer
[u]int32_t	4 byte [un]signed integer
float	32 bit IEEE 724 floating point number
char[n]	n byte fixed size ASCII nul terminated string
char[]	variable byte ASCII nul terminated string
uchar[n]	n * 2 byte fixed size UNICODE nul terminated string
uchar[]	variable byte UNICODE nul terminated string

Table 2.1: Primitive Types

All multibyte integers and floating point numbers stored in Dungeon Keeper 2 files have been observed to be little-endian. The types in Table 2.1 will be used throughout this document to describe structure layouts as consistently as possible, all of which can be assumed to be little-endian.

2.2 Structures

Offset	Name	Type	Description
0x00	worldTick	uint32_t	World loop iteration count

Table 2.2: Structure Example

Structure layouts for combinatorial types will appear as Table 2.2.

Offset is be a byte count in hexadecimal from the start of a structure with 0 padding for N hex digits such that: N is not less than 2, the highest offset

present is less than 16^N, and the highest order hex digit of the highest offset present represented in N hex digits is less than c.

For example, if a structure's highest offset present is 0xc40, N is 4, since 0xc40 is less than 16^3 but since the highest order hex digit of 0xc40 (c) is not less than a, N must be incremented.

Offset will also contain mathematical expressions as much as possible to describe variable offsets, referencing fields in the current or related structures. When an offset is unnecessarily complicated to describe via an expression, it will be marked as variable.

Structure layouts will contain rows which are contiguous. The data in one row will be immediately followed by the data in the next.

Type will identify one of the reference types in Table 1.1, or a related structure type. It also may appear with brackets (i.e. [4]), indicating that it is an array of N contiguous base types, for N contained within the brackets.

2.3 Enumerations

Value	Meaning
2	Animated Mesh - ANIM Subtype

Table 2.3: Enumeration Example

Enumerations will appear similar to Table 2.3, being related to a particular field in a structure.

2.4 Byte Expansions

Bits	Meaning
0	Enabled
1:11	10-bit signed integer

Table 2.4: Byte Expansions Example

Use of one or more bits as addressable fields will be expanded into a table similar to Table 2.4.

The Bits value 0 refers the the least significant bit of the associated fields binary representation. The associated field types size will determine the number of bits available. Bits which are unspecified are assumed to be unused and behavior undetermined.

Chapter 3

Text Assets

Text files are FourCC-identified blocks of data which correspond to character encodings, conversions, strings, and fonts.

3.1 BFMU - MultiByte to Unicode

BFMU files contain conversions from MultiByte to Unicode characters

BFMU

Offset	Name	Type	Description
0x00	id	uint32_t	'BFMU' - 0x554d4642
0x04	single_count	uint8_t	count of single byte characters
0x05	unknown	uint8_t	unused
0x06	x06	uint8_t	used in character count calculation
0x07	x07	uint8_t	used in character count calculation
0x08	chars	uchar[]	unicode characters by multibyte index

Table 3.1: BFMU

Threshold

The multibyte threshold for converting values is computed as follows:

$$threshold = 255 - single_count$$

Count

Character counts are calculated through fields in the BFMU structure. This does not indicate how many uchars are present, it is used only as a limiting factor during conversion:

$$count = (((x06 - 1) + 257) * 255) - threshold * 254 + x07$$

count relies on uint16_t truncation

This behavior is believed to be a bug, resulting in a count of 64515 for a file containing 250 unicode character conversions, with 0, 250, and 0 in the x04, x06, and x07 fields, respectively. Further research into multibyte conversion sample files is required for representations of non-zero single_count and x07.

Conversion

Conversions occur by using these fields on multi-byte characters:

$$mbb = *mb++ - 1$$

$$index = mbb < threshold?mbb : ((mbb * 255) - threshold * 254 + *mb++)$$

The count bounds these, preventing any index which exceeds the count, though this does not necessarily properly limit to the size of the chars array.

$$c = index < count?chars[index] : 0$$

When processing a stream of multibyte characters, the above pointer increments should be used, as passing the multibyte threshold indicates whether 1 or 2 bytes should be processed for a character.

3.2 BFUM - Unicode to MultiByte

BFUM files contain conversions from Unicode to MultiByte characters

BFUM

Offset	Name	Type	Description
0x00	id	uint32_t	'BFUM' - 0x4d554642
0x04	tree	umnode[]	unicode keyed binary tree of multibyte characters

Table 3.2: BFUM

UnicodeToMultiByteNode

Offset	Name	Type	Description
0x00	min_char	uchar	minimum unicode character represented
0x02	left_offset	uint16_t	offset in bytes from current node to left child
0x04	max_char	uchar	maximum unicode character represented
0x06	right_offset	uint16_t	offset in bytes from current node to right child
0x08	mbchars	uint16_t[]	multibyte character values, indexed by difference of unicode value and the node minimum

Table 3.3: UnicodeToMultiByteNode

Conversion

Conversion to a unicode character is conducted through a binary tree of the unicode characters to find. Each node of the tree contains a range of contiguous multibyte representations of unicode indexes. Tree traversal should be used to locate or iterate over the character set.

3.3 BFMB - Multi Byte

No representative files appear with BFMB header for English language assets.

3.4 BFST - Strings

BFST

Offset	Name	Type	Description
0x00	id	uint32_t	'BFST' - 0x54534642
0x04	key	uint32_t	strings list key from index
0x08	count	uint32_t	string count
0x0c	string_offsets	StringOffsets	

Table 3.4: BFST

StringOffsets

Offset	Name	Type	Description
0x00	offsets	uint32_t[./BFST.count]	array of offsets in bytes to strings

Table 3.5: StringOffsets

An offset refers to the position of a String entry in bytes relative to the position of the StringOffset.

String

Offset	Name	Type	Description
0x00	header	StringHeader	String entry Header
0x04	mbchars	uint8_t[header.count]	MultiByte String
variable	nul	uint32_t	String entry nul terminator

Table 3.6: String

Bits	Meaning
0:7	present - must be 1 to be valid string
8:31	character count

Table 3.7: StringHeader

All valid String entries will have a 1 in the present field, as well as a String offset-based 4-byte aligned uint32_t following the MultiByte array containing the value 0.

3.5 BF4F - Fonts

BF4F

Font characters are 4-bit grayscale images with variable encodings. The images are organized with top-left origins with tightly packed rows proceeding towards bottom, and may have a half-byte padding at the end of the last row (for images with odd total pixels).

Offset	Name	Type	Description
0x00	id	uint32_t	'BF4F' - 0x42463446
0x04	max_width	uint8_t	max width for any character in the font
0x05	max_height	uint8_t	max height for any character in the font
0x06	count	uchar	maxiumum character index represented in font
0x10	offsets	uint32_t[./count]	offsets in bytes to character font entries

Table 3.8: BF4F

Members of the offsets array are relative to the position of the BF4F structure

Offset	Name	Type	Description
0x00	char	uchar	unicode character represented
0x02	x02	uint16_t	unknown
0x04	data_size	int32_t	size of image payload
0x08	total_size	uint32_t	bytes in image payload after decoding
0x0c	flags	uint8_t	font flags
0x0d	x0d	uint8_t	unknown
0x0e	x0e	uint8_t	unknown
0x0f	x0f	uint8_t	unknown
0x10	width	uint16_t	width in pixels
0x12	height	uint16_t	height in pixels
0x14	offsetx	int8_t	offset from left
0x15	offsety	int8_t	offset from top
0x16	outer_width	int16_t	width with padding
0x18	image	uint8_t[./data_size]	character image

Table 3.9: FontEntry

Bits	Meaning
0	RLE4Data
1	unknown

Table 3.10: 'flags' Field

All image data, encoded or not, is organized as 4-bit values in high→low where in the following byte sequence:

$$0x440x030xa8$$

Values are to be processed in the following order:

$$4403a8$$

Row termination and decoding all assume a stream format - no padding snap should be implied when interpreting the data. An encoded source stream 4-bit may move from high to low depending on the position of the decoding output stream.

RLE4Data indicates that data is a run length encoded structure represented by high→low 4-bit control sequences. The following table illustrates the control interpretations.

'0'	→	'0'	:=	end		
	→	n	→	X	:=	$output([X_0, X_1, X_2, ...X_{n-1}])$
X	:=	output(X)				

unknown indicates that data is further/differently encoded. This format requires further investigation, and has no exemplar font assets in observed text languages.

With neither RLE4Data nor unknown set, font image data requires no decoding to be processed.

Image data is represented by high→low 4-bit grayscale representations. 0 values are considered transparent color keys.

Chapter 4

WAD World Assets Directory

WAD files are indexed key → value stores which are used to represent directories in a single file.

Chapter 5

KMF Keeper Mesh Format

Keeper Model Format (KMF) files follow the Interchange File Format (IFF) and only appear in WAD files.

5.1 IFF Hierarchy

Their FourCC codes are laid out in the following hierarchy

```
KMSH
        HEAD
        MATL
                MAT2
        MESH*
                HEAD
                CTRL
                SPRS
                        SPHD**
                        SPRS**
                GEOM
        ANIM*
                HEAD
                CTRL
                SPRS
                        SPHD**
                        SPRS**
                                POLY
                                VERT
                        ITAB
                        GEOM
                        VGEO
        GROP*
                HEAD
                ELEM***
```

* Only one of these sections will be present, depending upon KMSH/HEAD/format
** An array of KMSH/MESH,ANIM/HEAD/nsprs of these will be present
*** An array of KMSH/GROP/HEAD/nelems of these will be present

5.2 Types Reference

Triangle

Offset	Name	Type	Description
0x00	vertIndex	uint8_t[3]	Index into vertex definition list

Table 5.1: Triangle

UV

Offset	Name	Type	Description
0x00	uv	uint16_t[2]	Texture coordinate in uint16_t space. Divide by 32768 to normalize.

Table 5.2: UV

Vector

Offset	Name	Type	Description
0x00	xyz	float[3]	Coordinate, Normal, etc.

Table 5.3: Vector

5.3 KMF Structures

These tables are descriptions of the structures of the contents of the IFF chunks. Due to the hierarchical nature of the structures, type references to related structures will use a shorthand with . for the current structure path, and .. for the parent of the current structure path.

KMSH

The mesh toplevel container chunk. The only observed 'version' field value is 17. It contains:

A header common to all mesh types. If ./HEAD/type indicates a ./MESH or ./ANIM data type

A material table indexed by ./MESH,ANIM/SPRS/SPRS/material.

The type specific structure data which will be one of ./MESH, ./ANIM, or ./GROP types as indicated by ./HEAD/type

Offset	Name	Type	Description
0x00	version	uint32_t	Version number (only 17 observed)
0x04	head	./HEAD	Mesh Header
0x14	materials	./MATL	Material subsection (not present when ./HEAD/type == 3)
variable	data	./MESH or ./ANIM or ./GROP	Mesh data, pertaining to one of the indicated types

Table 5.4: KMSH

KMSH/HEAD

A header to indicate the type of the mesh.

Offset	Name	Type	Description
0x00	type	uint32_t	Mesh type
0x04	unknown	uint32_t	Value ignored (only 1 observed)

Table 5.5: KMSH/HEAD

Value	Meaning
1	Static Mesh - MESH Subtype
2	Animated Mesh - ANIM Subtype
3	Mesh Group - GROP Subtype

Table 5.6: 'type' enumerations

KMSH/MATL

A table of materials used by the mesh indexed by ../(ANIM,MESH)/SPRS/SPRS/material.

Offset	Name	Type	Description
0x00	count	uint32_t	Material Count
0x04	mat2	./MAT2[count]	Materials

Table 5.7: KMSH/MATL

KMSH/MATL/MAT2

An entry in the material table. It contains:
A list of EngineTexture names
The flags controlling the material's application
Brightness (pre-lighting color scale factor)
Gamma (post-lighting color scale factor)
An environment map used for reflective rendering

Offset	Name	Type	Description
0x00	name	char[]	Material Name
variable	count	uint32_t	EngineTexture Name Count
variable	textures	char[][count]	EngineTexture Names
variable	flags	uint32_t	Material Flags
variable	brightness	float	Material Brightness (pre-lighting)
variable	gamma	float	Gamma Correction (post-lighting)
variable	envmap	char[]	Environment Map EngineTexture

Table 5.8: KMSH/MAT2

Bits	Meaning
0	unknown
2	unknown
3	unknown
4	unknown
5	unknown
6	EnvironmentMapped

Table 5.9: 'flags' Field

KMSH/MESH

The static mesh type container. It contains:
A header describing mesh properties
Control Data with unknown function
Sprite Sections which describe mesh sections with a single material and limited to 255 vertices.
The Geometry for the mesh, indexed by ./SPRS/SPRS/vertices/geomIndex

Offset	Name	Type	Description
0x00	header	./HEAD	Mesh Header
variable	control	./CTRL	Mesh Control Data
variable	sprs	./SPRS	Mesh Sprite Sections
variable	geometry	./GEOM	Mesh Geometry

Table 5.10: KMSH/MESH

KMSH/MESH/HEAD

A header describing the static mesh properties.

Offset	Name	Type	Description
0x00	name	char[]	Mesh Name
variable	nsprs	uint32_t	Number of SPRS in file
variable	ngeom	uint32_t	Count of GEOM members
variable	pos	Vector	Basis Position of Mesh
variable	scale	float	Unit cube scale for culling
variable	nlods	uint32_t	progressive Levels Of Detail (LOD) used in file

Table 5.11: KMSH/MESH/HEAD

KMSH/MESH/CTRL

Offset	Name	Type	Description
0x00	count	uint32_t	Control Field Count
0x04	controls	mesh_ctrl[count]	Control Fields

Table 5.12: KMSH/MESH/CTRL

Offset	Name	Type	Description
0x00	unknown	uint32_t	Unknown
0x04	unknown	uint32_t	Unknown

Table 5.13: mesh_ctrl

KMSH/MESH/SPRS

Container for the sprite section list headers and elements.

Offset	Name	Type	Description
0x00	header	./SPHD[../HEAD/nsprs]	Sprite Section Headers
variable	sprs	./SPRS[../HEAD/nsprs]	Sprite Section Definitions

Table 5.14: KMSH/MESH/SPRS

KMSH/(MESH—ANIM)/SPRS/SPHD

A header item for each Sprite Section present in this file. The per-lod triangle counts are used in the Sprite Section to determine how many triangle index sets appear contiguously in the associated ../SPRS/tris.

Offset	Name	Type	Description
0x00	triCounts	uint32_t[../../HEAD/nlods]	Triangle count for each lod
variable	nvert	uint32_t	Number of Vertices
variable	mmFactor	float	Inverse factor in mipmap level decision (falloff?)

Table 5.15: KMSH/(MESH—ANIM)/SPRS/SPHD

KMSH/MESH/SPRS/SPRS

A data item for each Sprite Section present in this file. The tris member contains each lod's count of triangles, as indicated in ../SPHD/triCounts, organized contiguously.

Offset	Name	Type	Description
0x00	material	uint32_t	MATL Index
0x04	tris	Triangle[$\sum_{i=0}^{../../HEAD/nlods} ../SPHD/triCounts_i$]	Triangles
variable	vertices	mesh_vertex[../SPHD/nvert]	Vertices

Table 5.16: KMSH/MESH/SPRS/SPRS

Offset	Name	Type	Description
0x00	geomIndex	uint16_t	GEOM Index
0x02	uv	UV	Texture Coordinate
0x06	normal	Vector	Normal

Table 5.17: mesh_vertex

KMSH/MESH/GEOM

The static mesh geometry.

Offset	Name	Type	Description
0x00	geometry	Vector[../HEAD/ngeom]	Mesh Geometry

Table 5.18: KMSH/MESH/GEOM

KMSH/ANIM

The animated mesh type container. It contains:
A header describing animated mesh properties
Control Data with unknown function
Sprite Sections which describe animated mesh sections with a single material and limited to 255 vertices.

Offset	Name	Type	Description
0x00	header	./HEAD	Anim Header
variable	control	./CTRL	Anim Control Data
variable	sprs	./SPRS	Anim Sprite Sections

Table 5.19: KMSH/ANIM

KMSH/ANIM/HEAD

A header describing the animated mesh properties.

Offset	Name	Type	Description
0x00	name	char[]	Anim Name
variable	nsprs	uint32_t	Number of SPRS
variable	nfram	uint32_t	Frame Count
variable	nindx	uint32_t	Index Count
variable	ngeom	uint32_t	Count of GEOM members
variable	flags	uint32_t	Flag
variable	pos	Vector	Basis Position of Anim
variable	cubeScale	float	Unit cube scale for culling
variable	scale	float	GEOM/coord Scale Factor
variable	nlods	uint32_t	progressive Levels Of Detail (LOD) used in file

Table 5.20: KMSH/ANIM/HEAD

Bits	Meaning
0	Frame Factor Function: 0 : Clamp, 1 : Wrap

Table 5.21: 'flags' Field

KMSH/ANIM/CTRL

Offset	Name	Type	Description
0x00	count	uint32_t	Control Field Count (only 0 observed)
0x04	controls	anim_ctrl[count]	Control Fields

Table 5.22: KMSH/ANIM/CTRL

Offset	Name	Type	Description
0x00	unknown	uint16_t	Unknown
0x02	unknown	uint16_t	Unknown
0x04	unknown	uint32_t	Unknown

Table 5.23: anim_ctrl

KMSH/ANIM/SPRS

Offset	Name	Type	Description
0x00	header	./SPHD[../HEAD/nsprs]	Sprite Section Headers
variable	sprs	./SPRS[../HEAD/nsprs]	Sprite Section Definitions
variable	itab	./ITAB	Index Table
variable	geom	./GEOM	Geometry
variable	vgeo	./VGEO	Variable Geometry

Table 5.24: KMSH/ANIM/SPRS

KMSH/ANIM/SPRS/SPRS

Offset	Name	Type	Description
0x00	material	uint32_t	MATL Index
0x04	poly	./POLY	Triangles
variable	vertices	./VERT	Vertices

Table 5.25: KMSH/ANIM/SPRS/SPRS

KMSH/ANIM/SPRS/SPRS/POLY

Offset	Name	Type	Description
0x00	tris	Triangle[$\sum_{i=0}^{../../HEAD/nlods} ../SPHD/triCounts_i$]	Triangles

Table 5.26: KMSH/ANIM/SPRS/SPRS/POLY

KMSH/ANIM/SPRS/SPRS/VERT

A list of animated mesh vertices. The itabIndex member is a component of an index (the other being the current frame) into the ITAB table.

Offset	Name	Type	Description
0x00	vertices	anim_vertex[../../SPHD/nvert]	Vertices

Table 5.27: KMSH/ANIM/SPRS/SPRS/VERT

Offset	Name	Type	Description
0x00	uv	UV	Texture Coordinate
0x04	normal	Vector	Normal
0x10	itabIndex	uint16_t	ITAB Index

Table 5.28: anim_vertex

KMSH/ANIM/SPRS/ITAB

A table of indices into ../GEOM/geom. These indices are organized in ../../HEAD/nindx sized chunks for each 128 frame block.

Calculcation of a base geometry index for a given frame and itabIndex is as follows:

$$\text{geomBase} = \text{geomIndex}[frame >> 7][\text{itabIndex}]$$

The count of geomIndices per ../../HEAD/nindx is given as

$$\text{indxCount} = \frac{../../HEAD/nfram}{128} + 1$$

Offset	Name	Type	Description
0x00	geomIndex	uint32_t[nindx][indxCount]	GEOM Base Index

Table 5.29: KMSH/ANIM/SPRS/ITAB

KMSH/ANIM/SPRS/GEOM

A table of mesh geometry. The anim_vector structure encodes a fixed-point 3 dimensional coordinate in 10 bits per dimension. This coordinate vector is then scaled via ../../HEAD/scale and added to the ../../HEAD/pos basis to transform into world space.

An extra anim_vertex entry is added for each (../../HEAD/nfram / 128) + 1 block per index, allowing geomIndex + 1 to be assumed to be present for interpolation purposes, even for the last frame in the animation.

Offset	Name	Type	Description
0x00	geom	anim_geom[../HEAD/ngeom]	Geometry

Table 5.30: KMSH/ANIM/SPRS/GEOM

Offset	Name	Type	Description
0x00	coord	anim_vector	Coordinate
0x04	frameBase	uint8_t	Frame Base Modulus

Table 5.31: anim_geom

Bits	Meaning
0:9	Z
10:19	Y
20:29	X

Table 5.32: anim_vector

KMSH/ANIM/SPRS/VGEO

A table of offsets from geomBase calculated from ITAB. These offsets are organized into ../../HEAD/nfram sized chunks for each ../../HEAD/nindx. Calculation for index in GEOM items from geomBase for a given itabIndex and frame is as follows:

$$geomOffset = offsets[itabIndex][frame]$$
$$geomIndex = geomBase + geomOffset$$

Simple interpolation must be used to discover the progression from the most recent keyframe. The frameBase field of GEOM is used in this interpolation to determine percentage progressed for a given frame, and the .next. geom element is always geomIndex + 1. The offset value resets at each 128 frame chunk. Calculation of the the interpolation geomFactor for a given geomIndex is as follows:

$$frameBase = GEOM.geom[geomIndex].frameBase$$
$$nextFrameBase = GEOM.geom[geomIndex + 1].frameBase$$
$$geomFactor = ((frame \& 0x7f) - frameBase)/(nextFrameBase - frameBase)$$

This makes calculation of the interpolated position at a given geomIndex as follows:

$$coord = GEOM.geom[geomIndex].coord$$
$$nextCoord = GEOM.geom[geomIndex + 1].coord$$
$$interpCoord = (nextCoord - coord) * geomFactor + coord$$

Offset	Name	Type	Description
0x00	offsets	uint8_t[../HEAD/nfram][../HEAD/nindx]	GEOM Offset for Frame

Table 5.33: KMSH/ANIM/SPRS/VGEO

KMSH/GROP

A group of meshes.

Offset	Name	Type	Description
0x00	header	./HEAD	Group Header
0x0c	elements	./ELEM[./HEAD/count]	Group Element

Table 5.34: KMSH/GROP

KMSH/GROP/HEAD

Offset	Name	Type	Description
0x00	count	uint32_t	Element Count

Table 5.35: KMSH/GROP/HEAD

KMSH/GROP/ELEM

An element of a mesh group. name identifies the mesh entry without .kmf extension in Meshes asset. pos specifies the translation of the element from the group origin.

Offset	Name	Type	Description
0x00	name	char[]	Element Name
variable	pos	Vector	Offset Position

Table 5.36: KMSH/GROP/ELEM

Chapter 6

KCS Keeper Camera Sweep

Keeper Camera Sweep (KCS) files represent Camera paths. A single KCS structure makes up its layout.

6.1 KCS Structures

KCS

Offset	Name	Type	Description
0x00	count	uint32_t	Count of following frames.
0x04	frames	CameraSweepFrame[count]	Camera data for each frame.

Table 6.1: KCS

CameraSweepFrame

A CameraSweepFrame identifies the position and orientation of the camera.

Offset	Name	Type	Description
0x00	position	Vector	Position
0x0c	forward	Vector	Forward Orientation Vector
0x18	left	Vector	Left Orientation Vector
0x24	up	Vector	Up Orientation Vector
0x28	fovy	float	Field of view (radians)
0x2c	near	float	Near clipping distance in fixed units

Table 6.2: CameraSweepFrame

The axis vectors are placed into a 3x3 rotation matrix with the following positioning for multiplication with the scene.

$$\begin{pmatrix} forward[0] & -left[0] & -up[0] \\ -forward[1] & left[1] & up[1] \\ -forward[2] & left[2] & up[2] \end{pmatrix}$$