CMSK Collision Mask File Specification V1.0

Revision 1 (28th May 2014)

Introduction

CMSK files are for storing bitmap masks for pixel perfect collision detection. The most likely usage is for accurate collision detection in games.

This file format is much simpler than standard image formats such as PNG, so that implementing a file reader in a new programming language is quick and easy. The loading time should also be fast as processing time can be critical in gaming applications.

A simple data compression scheme is used which allows fast decompression, but due to the simplicity of mask data, a comparable compression ratio to other image formats is achieved.

Overview

The following table outlines the layout of CMSK file:

Byte offset	Length (bytes)	Information	Description
0	10	Signature	The CMSK signature, which is the sequence of bytes: [139,67,77,83,75,13,10,26,10,0]
10	4	Version	32-bit big endian integer containing the version number
14	4	Mask Data Offset	32-bit big endian integer containing the offset in the file that the mask data starts
18	4	Mask Data Length	32-bit big endian integer containing the length of the mask data
22	4	Width	32-bit big endian integer containing the width of the mask
26	4	Height	32-bit big endian integer containing the height of the mask
-	-	-	Later versions may store data here
Read from Mask Data Offset	Read from Mask Data Length	Mask Data	The mask data
-	-	-	Later versions may store data here
Last 4 bytes	4	CRC-32	A CRC-32 checksum of the entire file to this point

Description

The different sections of a CMSK file are discussed below.

Signature

Offset: 0
Length: 10 bytes
The signature is a sequence of 10 bytes that identifies the file as a CMSK file.
The bytes should be as follows:

139	67	77	83	75	13	10	26	10	0
High bit set	C	M	S	K	Carriage Return	Newline	Ctrl-Z	Newline	Null

File writers should output these bytes as the first 10 bytes of the file.

File readers should check that the first 10 bytes of the file match these bytes exactly. If the signature does not match, then either the file is not a CMSK file, or the file is corrupt.

Version

Offset: 10
Length: 4 bytes
The version bytes hold a 32-bit integer in big endian format.
This value holds the internal CMSK version number that the file claims to be compatible with.
For V1.0 files this should contain 0x00000001 (1).
New versions will increase this value by one for each new version. The internal version will always be a single integer, not a [major.minor] number, for example.

File writers should always output 0x00000001 (1) for the version.

File readers should check that the version is equal to 0x00000001 (1) before processing the rest of the file. Future versions are not guaranteed to be backwards compatible or super-sets of current versions. A reader should abort if the version is not explicitly supported.

Mask Data Offset

Offset: 14
Length: 4 bytes
The mask data offset bytes hold a 32-bit integer in big endian format.
This value is the offset within the file of the first byte of the *Mask Data*.

File writers should output the offset of the first byte of the *Mask Data* for this value. For example, a valid value could be 0x0000001E (30), as this is the first byte available after the fixed position values.

File readers should read this value and use it as an offset in the file of where to start reading the *Mask Data* from. A reader should not hard-code or assume any default value for the *Mask Data Offset*, as writers are free to choose any offset they require.

Mask Data Length

Offset: 18
Length: 4 bytes
The mask data length bytes hold a 32-bit integer in big endian format.
This value is the length in bytes of the *Mask Data*.

File writers should output the length in bytes of the *Mask Data* for this value.

File readers should read this value and use it as the maximum length when parsing the *Mask Data*. A reader should not hard-code or assume any default value for the *Mask Data Length* as the data is compressed and so the length will vary.

Width

Offset: 22
Length: 4 bytes
The width bytes hold a 32-bit integer in big endian format.
This value is the width in pixels (currently also bytes) of the decompressed mask data.

File writers should output the width of the mask in pixels for this value.

File readers should read this value and store it as the width of the mask.

Height

Offset: 26
Length: 4 bytes
The height bytes hold a 32-bit integer in big endian format.
This value is the height in pixels (currently also bytes) of the decompressed mask data.

File writers should output the height of the mask in pixels for this value.

File readers should read this value and store it as the height of the mask.

Mask Data

Offset: Read from Mask Data Offset
Length: Read from Mask Data Length
The mask data bytes holds the Run Length Encoded (RLE) compressed mask data.

The data is compressed on a per-byte basis.

The RLE method employed does not use a specific escape byte value and instead uses a double byte to represent an escape. This way single bytes are encoded verbatim, while any repeated bytes are encoded as a double byte followed by a length. As the maximum value for a byte is 255, any runs longer than 255 bytes will be divided into multiple encodings of 255 bytes followed by the remainder.

For example, the following bytes:

0	2	5	5	5	5	5	5	3	8

would be encoded as follows:

0	2	5	5	6	3	8	-	-	-
Single Value	Single Value	Double Escape	Double Escape	Length of Run	Single Value	Single Value			

The uncompressed mask data is a byte-per-pixel bitmap representing which areas should be classed as collisions and which areas should not. Each byte contains a value from 0 to 255.

Typically 0 should represent an empty section of the mask that a collision should not occur in, while 255 should represent a solid section of the mask, that will cause a collision. It is recommended that values in-between (1-254) are subjected to a *threshold* value in the collision detection code that allows the user to choose the threshold at which a collision should occur.

The easiest way of creating a mask is to use the alpha channel from an image, as this will define the solid and empty areas of an object.

File writers should encode the mask data and output the compressed data for these bytes. The *Mask Data Length* value should be set to the length of the compressed data.

File readers should read these bytes and decompress the data. The length of the decompressed data will be exactly equal to the *Width* multiplied by the *Height* value.

CRC-32

Offset: Last 4 bytes of file
Length: 4 bytes
The CRC-32 bytes hold a 32-bit integer in big endian format.
This value is a checksum of the all the bytes in the file up to this point using the CRC-32 algorithm.

File writers should calculate and output a CRC-32 checksum for this value.

File readers **may choose** to calculate a CRC-32 checksum for the file (excluding the last 4 bytes) and verify that this value is correct. It is recommended that any readers that are not time critical take the time to verify the checksum so that corrupt files can be detected. Readers that are time critical (such as in games) are free to skip the CRC-32 verification in the name of speed.