CBF

- Specifications –

1. INTRODUCTION

CBF, or the Cubeflix Binary Format, is a dynamic and extensible file format specification designed for large binary files. CBF functions as a key-value data format, with support for multiple data types and raw binary blobs. CBF is designed to be easy to implement and fast, with a low memory footprint.

This document outlines the specifications for CBF version ‘A’ (1.0.0).

2. SPECIFICATIONS

A CBF file is split into three sections, the *header*, *dataset section* and *binary section*. The header contains metadata about the CBF file. The dataset section contains the primary key-value data associated with the file. The binary section contains arbitrary unstructured binary data that is referenced by the dataset section. These three sections directly follow each other within the CBF file. If a file does not contain any binary data, the binary section may be omitted.

The general layout of a CBF file is as follows:

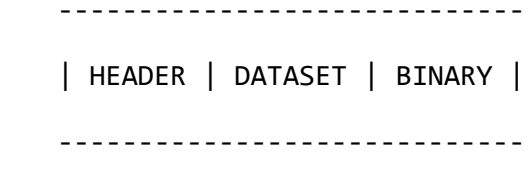

Figure 2.1: The layout of a CBF file.

2.1. HEADER SECTION

The header consists of the *magic number*, followed by the *version*. The magic number consists of two bytes: the ASCII-encoded string ‘CB’, represented by the hexadecimal values 0x43 and 0x42. The version is a single byte representing the file’s CBF version in ASCII form (currently ‘A’, or the hexadecimal value 0x41).

The layout of the header is as follows:

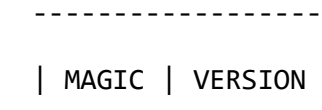

Figure 2.2: The layout of the CBF header.

An example of a valid CBF header is (in hexadecimal bytes): 0x43 0x42 0x41. This would imply that the CBF version for the file is version ‘A’. All CBF headers are exactly 3 bytes long.

2.2. DATASET SECTION

The dataset section contains the key-value data for the CBF file. In a CBF file, a *key* is a unique string which points to a *value*, a piece of data that can take various forms. The dataset section consists of one *dataset block*, a set of unordered key-value pairs that defines the dataset.

2.3. DATASET BLOCK

A dataset block consists of a list of key-value pairs. A block begins with the number of key-value pairs in the block, encoded as a 64-bit little-endian unsigned integer. This is directly followed by each of the key-value pairs that make up the dataset block, each directly following the last.

The layout of a dataset block is as follows:

Figure 2.3: The layout of a dataset block.

2.4. KEY-VALUE PAIR

A key-value pair consists of three parts: the key, data type, and value. These three sections directly follow each other.

The layout of a key-value pair is as follows:

Figure 2.4: The layout of a key-value pair.

2.4.1. KEY

A key is a unique string (within the dataset block) that is paired with

a value in a key-value pair. A key consists of the length of the key string, encoded as a 16-bit little-endian unsigned integer, followed by the key itself, encoded in ASCII. Keys are not null-terminated.

Figure 2.5: The layout of a key.

2.4.2. DATA TYPE

The data type is a single byte representing the data type of the value in a key-value pair. A list of data types and their associated values are available in section 3.

2.4.3. VALUE

A value in a key-value pair can take many forms, depending on the data type. The length of the value in bytes is not guaranteed, and implementations of the CBF format should read the value based on the data type. See section 3 for more information.

2.5. BINARY SECTION

The binary section contains raw, unstructured binary data. The binary section is generally used in conjunction with the “BLOB” data type (see section 3.2) for storing large amounts of data that may be read at a later time. Implementations should load the dataset first, before reading the binary section, as it can be very large.

In files that do not use the “BLOB” data type, the binary section is not required and may be omitted.

3. DATA TYPES

The following is a list of all acceptable data types and their associated values.

3.1. NONE

The “NONE” data type has a data type value of 0x00. The associated value should be empty. The “NONE” data type represents a null or unspecified value. In JavaScript implementations of the CBF format, this should deserialize to “null”, not “undefined”.

3.2. BLOB

The “BLOB” data type has a data type value of 0x01. The associated value should be a 64-bit pointer to the location of the blob data within the file, followed by a 64-bit little-endian unsigned integer containing the length of the data. The pointer should point to a valid location within the file, in the binary section. The “BLOB” data type represents a blob of data within the binary section of the CBF file.

3.3. DATASET

The “DATASET” data type has a data type value of 0x02. The associated value should be a full dataset block (see section 2.3). The new dataset block may contain keys already used in the parent block. The “DATASET” data type represents an embedded key-value dataset block.

3.4. STRING

The “STRING” data type has a data type value of 0x03. The associated value should be the length of the UTF-8 encoded string, encoded as a 64-bit little-endian unsigned integer, followed directly by the string itself, encoded in UTF-8. The string should not be null-terminated. The “STRING” data type represents a UTF-8 encoded string.

3.5. INT

The “INT” data type has a data type value of 0x04. The associated value should be a 64-bit little-endian signed integer. The “INT” data type represents a signed 64-bit integer.

3.6. UINT

The “UINT” data type has a data type value of 0x05. The associated value should be a 64-bit little-endian unsigned integer. The “UINT” data type represents an unsigned 64-bit integer.

3.7. FLOAT

The “FLOAT” data type has a data type value of 0x06. The associated value should be a 64-bit double-percision IEEE 754 floating point number. The “FLOAT” data type represents a double-percision floating point number.

3.8. BYTES

The “BYTES” data type has a data type value of 0x07. The associated value should be the length of the byte data, encoded as a 64-bit little-endian unsigned integer, followed by the byte data itself. The “BYTES” data type represents raw byte data, or a bytearray.

3.9. BOOL

The “BOOL” data type has a data type value of 0x08. The associated value should be a single byte, either 0x00 or 0xFF. Values of 0x00 represent a boolean value of “false”, while values of 0xFF represent a boolean value of “true”. The “BOOL” data type represents a boolean value.

4. NOTES ON IMPLEMENTATION

4.1. FILE EXTENSIONS

It is recommended that CBF files use the “.cbf” file extension, although this is not required. Compressed CBF files should NOT use the “.cbf” file extension, but may use a variation of the “.cbf” extension, such as “.cbf.gz” or “.gcbf” for gunzip-compreseed CBF files.