Fairytale File Format

Draft v1.2 – Márcio Pais, 2018

The Fairytale file format is designed to be flexible and extensible. It makes heavy use of an efficient byte-aligned variable length integer encoding strategy (henceforth VLI) and unless stated otherwise, uses little-endian encoding for non-VLI data types. See the appendix for more information on VLI.

A Fairytale archive is composed of the following structures in linear order:

File header	*Compressed data*	*Codec definitions*	*Block segmentation*	*Directory tree*	*File list*

Aside from the *file header* and the *compressed data* itself, all structures are composed of:

Field name	Data type
Size	VLI
Data	variable
Checksum (CRC32)	4 bytes

The *size* field is specified in bytes and represents the length of the *data* for the structure. The *checksum* field takes into account the *size* field along with the *data* of the structure.

Metadata

All metadata related to files and directories is optional, and uses a tag list representation, where each tag is composed of:

Field name	Data type
Tag id	VLI
Tag length	VLI
Tag content	variable

The decoder should ignore unknown tags. The *tag length* field represents the size, in bytes, of the *tag content* only. The decoder should read the tags until it finds a *termination* tag, i.e., with its *tag id* set to 0 (zero). This special tag doesn't require any further fields.

File header

The *file header* is composed of:

Field name	Data type
Magic signature	3 bytes
Version	1 byte
Flags	1 byte
Data size	8 bytes (signed integer)

The *magic signature* is formed by the ASCII characters "*FTL*" (0x46, 0x54, 0x4C).

The *version* byte is currently defined as the ASCII digit "*0*" (0x30) and should be incremented if subsequent backward-incompatible changes are introduced.

The *flags* byte specifies global options that apply to the whole archive. For more information, see the appendix.

The *data size* field specifies the size, in bytes, of the compressed block data, and as such is used to determine the offset to the *codec definitions* structure.

Codec definitions

The *codec definitions* structure describes what codecs were used in the compression of the chunks, and their parameters. It is composed of:

Field name	Data type
Number of sequence entries	VLI
Codec sequence entries	Codec sequence list

The *number of sequence entries* specifies how many codec configuration sequences are used in the archive, and is followed by a list with these many elements of *codec sequence entries*, defined as such:

Field name	Data type
Number of codecs	VLI
Codec entries	Codec list

The *number of codecs field* specifies how many codecs are used in the sequence, and is followed by a list with these many elements of *codec entries*, defined as such:

Field name	Data type
Codec id	VLI
Parameters	variable

The *codec id* field identifies the codec used and the *parameters* field contains the required codec parameters to decompress the data, if any are needed.

Block segmentation

The *block segmentation* structure describes the blocks that are encoded in chunks in the archive, along with their relative relationship. It is composed of:

Field name	Data type
Chunk info sequences	Chunk info sequence list
Block node sequences	Block node sequence list

Each *chunk info sequence* is composed of:

Field name	Data type
Chunk size	VLI
Checksum (CRC32)	4 bytes
Flags	1 byte
Codec sequence id	VLI
Block count	VLI
Block type	VLI
Block info entries	Block info entry list

The *chunk size* field specifies the size, in bytes, of the compressed chunk. The decoder should keep reading until an entry of 0 (zero) *chunk size* is seen (this entry doesn't require any further fields).

The *checksum* field applies only to the uncompressed data encoded in the chunk.

The *flags* field specifies options that apply to this chunk only. For more information, check the appendix.

The *codec sequence id* field specifies the *codec sequence entry* to use to decompress this chunk.

The *block count* field specifies how many *block info entries* of this *block type* are encoded in the chunk.

In non-solid compression mode, each chunk encodes only one block. The decoder should read all *chunk info sequences* and verify if the total size matches that specified in the *data size* field of the file header.

Each *block info entry* is composed of:

Field name	Data type
Size	VLI
Checksum	2 bytes
Info	variable

The *block info entries* state the *size*, in bytes, of the uncompressed data representing each block, in linear order. The relative offsets into the uncompressed chunk data for each block can thus be derived by accumulating the sizes of previous blocks.

The *checksum* field is optional and depends on the *flags* field of this *chunk info sequence* having the option "use checksum-per-block" active and the *block count* being higher than 1 (one). The *checksum* value is comprised of the 2 most significant bytes of the CRC32 checksum of the uncompressed data for this block.

The *info* field is optional and depends on the *block type* of this *chunk info sequence*.

Upon reading each entry, the block is assigned an *id*, derived incrementally and starting at 0 (zero).

Chunk info sequences are thus required to match the linear order of the compressed chunks of blocks as they are stored in the archive.

Each *block node sequence* is composed of:

Field name	Data type
Block count	VLI
Block type	VLI
Block node entries	Block node entry list

The *block count* field specifies how many blocks of this *block type* will be defined by the following *block node entries*. The decoder should keep reading until an entry of 0 (zero) *block count* is seen (this entry doesn't require any further fields).

The *block node entries* are themselves lists that describe blocks composed of other child blocks, which as such weren't defined by the *block info entries*. Each *block node entry* is defined as such:

Field name	Data type
Number of child blocks	VLI
Checksum	2 bytes
Info	variable
Block ids	VLI list

The *number of child blocks* field specifies how many child blocks compose the current block. As before, upon reading each entry, the block is assigned an *id*, derived incrementally and starting from the lowest available *id*.

The *checksum* field is optional and depends on the *flags* field of the *file header* having the option "use checksum-per-parent-block" active. The *checksum* value is comprised of the 2 most significant bytes of the CRC32 checksum of the uncompressed data for this block.

The *info* field is optional and depends on the *block type* of this *block node sequence*.

The *block ids* list the child blocks of the current block in linear order. Should an entry specify a *block id* which has still not been assigned, the decoder must assume the archive is corrupted.

See the appendix for a visual description of this *block segmentation* structure.

Directory tree

The *directory tree* is a structure describing the relative relationship of the directories present in the archive. Should no directories exist, the *size* field of the structure must be set to 0 (zero), and the *checksum* must follow it.

The *data* field of this structure is composed of a linear list of *directory entries*, defined as such:

Field name	Data type
Parent id	VLI
Length	VLI
Name	UTF8 string
Metadata	Tag list

The *parent id* field states the parent directory of the current entry. If set to 0 (zero), then it belongs to the root directory. Upon reading each entry, it is itself assigned an *id*, derived incrementally and starting at 1 (one). Should an entry specify a *parent id* which has still not been assigned, the decoder must assume the archive is corrupted.

The *length* field specifies the size, in bytes, used by the *name* field. It is up to the decoder to check the validity of the *name* string and provide any necessary conversions.

File list

The *file list* is a structure that contains information about all files present in the archive and cannot have 0 (zero) *size*. It is composed of a linear list of *file entries*, defined as such:

Field name	Data type
Directory id	VLI
Length	VLI
Name	UTF8 string
Metadata	Tag list
Number of blocks	VLI
Block ids	VLI list

The *directory id* field states to what directory this file belongs to.

The *length* field specifies the size, in bytes, used by the *name* field. It is up to the decoder to check the validity of the *name* string and provide any necessary conversions.

The *number of blocks* field specifies how many blocks compose this file, and is followed by a list with these many elements of VLI encoded *block ids*, in linear order. The decoder should validate these *ids* against the *block segmentation* specified.

Appendix

Block segmentation example, full solid-mode

Section	Structure	Field	Value
Chunk info sequences	Chunk info sequence 0	Chunk Size	-
		Checksum	-
		Flags	-
		Codec sequence id	-
		Block Count	4
		Block Type	*Default*
	Block info entry 0	Size	-
	...	...	...
	Block info entry 3	Size	-
	Chunk info sequence 1	Chunk Size	-
		Checksum	-
		Flags	-
		Codec sequence id	-
		Block Count	1
		Block Type	*Image*
	Block info entry 0	Size	-
		Info	*Image info structure*
	Chunk info sequence 2	Chunk Size	0
Block node sequences	Block node sequence 0	Block count	2
		Block Type	*DEFLATE*
	Block node entry 0	Number of child blocks	1
		Info	*DEFLATE info structure*
		Block id 0	2
	Block node entry 1	Number of child blocks	1
		Info	*DEFLATE info structure*
		Block id 0	4
	Block node sequence 1	Block count	1
		Block Type	Base64
	Block node entry 0	Number of child blocks	1
		Info	*Base64 info structure*
		Block id 0	6
	Block node sequence 2	Block count	0

In diagram form, this *block segmentation* corresponds to:

Default, id: 0	*Default, id: 1*	*DEFLATE, id: 5*	*Default, id: 3*	*Base64, id: 7*
		Default, id: 2		*DEFLATE, id: 6*
				Image, id: 4

If *non-solid* compression mode was used, the *block segmentation* would look like this:

Section	Structure	Field	Value
Chunk info sequences	Chunk info sequence 0	Chunk Size	-
		Checksum	-
		Flags	-
		Codec sequence id	-
		Block Count	1
		Block Type	*Default*
	Block info entry 0	Size	-
	...	...	...
	Chunk info sequence 3	Chunk Size	-
		Checksum	-
		Flags	-
		Codec sequence id	-
		Block Count	1
		Block Type	*Default*
	Block info entry 0	Size	-
	Chunk info sequence 4	Chunk Size	-
		Checksum	-
		Flags	-
		Codec sequence id	-
		Block Count	1
		Block Type	*Image*
	Block info entry 0	Size	-
		Info	*Image info structure*
	Chunk info sequence 5	Chunk Size	0
Block node sequences	Block node sequence 0	Block count	2
		Block Type	*DEFLATE*
	Block node entry 0	Number of child blocks	1
		Info	*DEFLATE info structure*
		Block id 0	2
	Block node entry 1	Number of child blocks	1
		Info	*DEFLATE info structure*
		Block id 0	4
	Block node sequence 1	Block count	1
		Block Type	Base64
	Block node entry 0	Number of child blocks	1
		Info	*Base64 info structure*
		Block id 0	6
	Block node sequence 2	Block count	0

Variable length integer encoding

Unsigned integers are encoded byte-aligned in 7 bit chunks with a 1 bit flag to specify end-of-encoding, assigned to the most significant bit (msb) of each byte of the representation. Each iteration consumes the 7 least significant bits (lsb) of the remaining number, activating the flag bit if any more non-zero bits remain.

This scheme allows encoding of all non-negative numbers representable by a 64 bit signed integer in up to 9 bytes.

File header flags

Name	Value	Meaning
Use checksum-per-block	0x01	Store 16-bit checksum for every encoded block, for faster deduplication when adding new files to an existing archive

Chunk info sequence flags

Name	Value	Meaning
Use checksum-per-parent-block	0x01	Store 16-bit checksum for every parent block, for faster deduplication when adding new files to an existing archive

File metadata tags

Name	Id	Length	Content
Original file size	1	8	Stores the original uncompressed file size