The Generic Tagged Array File Format

Changes

- **2019-08-26** Compression is now Deprecated.
- **2013-02-06** Fixed typos.
- **2010-08-17** Remove the meaning of the second lowest bit in the flag byte of the header. The same information can be extracted from the sixth header byte instead. This allows to simplify the libgta API. Compatibility with older libgta versions is retained.
- **2010-06-06** Various extensions, cleanups, clarifications. The list of reserved tags was moved to an external file because it may change while the file format itself remains unchanged.
- **2010-04-16** Initial version.

Introduction

This document specifies version 1 of the Generic Tagged Array (GTA) file format. This file format has the following features:

- GTA files can store any kind of data in multidimensional arrays
- GTA files can use simple tags to describe the data
- GTA files are streamable, which allows direct reading from and writing to pipes, network sockets, or other non-seekable media
- GTA files allow easy out-of-core data access

Type	No.	Bytes	Description
`GTA_INT8`	1	1	C99 `int8_t`
`GTA_UINT8`	2	1	C99 `uint8_t`
`GTA_INT16`	3	2	C99 `int16_t`
`GTA_UINT16`	4	2	C99 `uint16_t`
`GTA_INT32`	5	4	C99 `int32_t`
`GTA_UINT32`	6	4	C99 `uint32_t`
`GTA_INT64`	7	8	C99 `int64_t`
`GTA_UINT64`	8	8	C99 `uint64_t`
`GTA_INT128`	9	16	C99 `int128_t` (unavailable on many platforms)
`GTA_UINT128`	10	16	C99 `uint128_t` (unavailable on many platforms)
`GTA_FLOAT32`	11	4	IEEE 754 single precision floating point (on many platforms: `float`)
`GTA_FLOAT64`	12	8	IEEE 754 double precision floating point (on many platforms: `double`)
`GTA_FLOAT128`	13	16	IEEE 754 quadrupel precision floating point (unavailable on many platforms, even if `long double` exists)
`GTA_CFLOAT32`	14	8	complex (re,im) based on `GTA_FLOAT32`
`GTA_CFLOAT64`	15	16	complex (re,im) based on `GTA_FLOAT64`
`GTA_CFLOAT128`	16	32	complex (re,im) based on `GTA_FLOAT128`
`GTA_BLOB`	0	—	Data blob; must be endianness-independent; user must specify the size

Table 1: GTA data types. All integer types use the common two-complement representation. All floating point types conform to IEEE 754. A corresponding C99 type might not exist on all platforms.

1 Definitions

1.1 Arrays

An array has zero or more *dimensions*, and a size of at least one in each dimension. These dimension sizes determine the number of array elements. An array element consists of zero or more *components*. Each component has a *type*. The types are defined in table 1. The special type `GTA_BLOB` allows the user to define custom types of arbitrary length.

For example, an image with 640×480 pixels in the common RGB format would have two dimensions, the first with size 640, the second with size 480. Each array element would consist of three components, all of type `GTA_UINT8`.

1.2 Tags

A *tag* consists of a tag name and a tag value.

A tag name is a non-empty, zero-terminated, UTF-8 encoded string that does not contain control characters (characters with a code less than 32 or equal to 127) and does not contain the character `=`.

A tag value is a zero-terminated, UTF-8 encoded string that does not contain control characters (characters with a code less than 32 or equal to 127). It can be empty, and it can contain the character `=`.

Leading or trailing white space is not ignored; it is part of the name or value string.

A *tag list* consists of zero or more tags.

An array has multiple tag lists:

- One global tag list, containing tags that are relevant to the array as a whole.
- A dimension tag list for each dimension, containing tags that are relevant to this particular dimension.
- A component tag list for each element component, containing tags that are relevant to this particular component.

An array does not need to have any tags in its tag lists. If it does have tags, these are subject to certain rules, see the external documentation about reserved tags.

For example, an image might have a global tag list containing the tag `DATE=Fri, 4 Dec 2009 22:29:43 +0100 (CET)`, a tag list for the first dimension containing the tag `INTERPRETATION=X`, a tag list for the second dimensions containing the tag `INTERPRETATION=Y`, and three component tag lists, containing the tags `INTERPRETATION=SRGB/RED`, `INTERPRETATION=SRGB/GREEN`, `INTERPRETATION=SRGB/BLUE` respectively.

1.3 Chunks

A *chunk* stores a number of chunk data bytes, either compressed or uncompressed. Note that compression is deprecated, so only legacy files may be compressed but newly written files must always be uncompressed. It begins with a chunk header, defined as follows:

- The first 8 bytes of the chunk header contain a value of type `GTA_UINT64`, subject to the endianness type of the file (see next section for details). This value is the number of bytes of the chunk data (excluding the header). Currently, it is limited to 2^{24}, so that a chunk cannot store more than 16 MiB.
- Only if the previous value is not zero, the next byte of the chunk contains a value of type `GTA_UINT8`. This value specifies the compression method of the chunk, as defined in Tab. 2.

Compression method	Number	Description
GTA_NONE	0	No compression
GTA_ZLIB	1	ZLIB compression with default level (fast, moderate compression ratio)
GTA_ZLIB1	4	ZLIB compression with level 1
GTA_ZLIB2	5	ZLIB compression with level 2
GTA_ZLIB3	6	ZLIB compression with level 3
GTA_ZLIB4	7	ZLIB compression with level 4
GTA_ZLIB5	8	ZLIB compression with level 5
GTA_ZLIB6	9	ZLIB compression with level 6
GTA_ZLIB7	10	ZLIB compression with level 7
GTA_ZLIB8	11	ZLIB compression with level 8
GTA_ZLIB9	12	ZLIB compression with level 9
GTA_BZIP2	2	BZIP2 compression (moderate speed, good compression ratio)
GTA_XZ	3	XZ compression (low compression speed, moderate decompression speed, good or very good compression rates)

Table 2: GTA compression methods. Note that compression is deprecated and only described here for backwards compatibility. New GTA files are always uncompressed.

- Only if the compression method is not GTA_NONE, the next 8 bytes contain a value of type GTA_UINT64, subject to the endianness type of the file. This value is the number of bytes that the compressed chunk data uses. It is guaranteed to be less than or equal to the number of bytes of the chunk data. If a compression method would produce compressed chunk data that is larger than the uncompressed chunk data, the chunk is simply not compressed but stored uncompressed (with the compression method GTA_NONE).

After the chunk header (8, 9, or 17 bytes, depending on the chunk data size and compression method), the chunk data follows, either uncompressed, using the number of bytes given in the first header value, or compressed, using the number of bytes given in the third header value.

A *chunk list* is a list of one or more chunks. A chunk with the chunk data size zero marks the last chunk in a chunk list.

2 File Format

A GTA consists of the GTA header and the GTA data. The GTA header stores information about the file format, the list of dimensions, the list of element components, and all tag lists. See Sec. 2.1. The GTA data is stored immediately after the header. See Sec. 2.2.

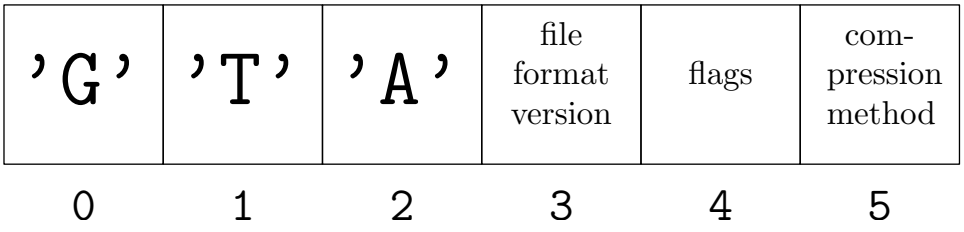

Figure 1: The first six bytes of a GTA header. Note that the compression method must be `GTA_NONE` for all new files since compression is deprecated.

2.1 GTA Header

Each GTA starts with a header. The first six bytes of the header identify the GTA version and properties. See Fig. 1.

The first three bytes of a GTA header are of type `GTA_UINT8` and contain the UTF-8 codes of the characters `G`, `T`, and `A`.

The fourth byte is of type `GTA_UINT8` and contains the GTA file format version number. Currently, this value must always be 1.

The fifth byte is of type `GTA_UINT8`, and each bit represents a flag that can be set (1) or unset (0).

The lowest bit flag determines if multibyte values in the file are in big endian (1) or little endian (0) byte ordering. This flag applies to all values in the header and in the data. (The array element component type `GTA_BLOB` is special: it must be independent of byte ordering, so that it is valid regardless of this flag.)

The second lowest bit used to have a meaning in previous versions of this specification. For compatibility reasons, this flag must be ignored when reading a GTA. When writing a GTA, it must be unset (0) if the value `GTA_NONE` is stored in the sixth header byte (described below), and it must be set (1) otherwise.

All six remaining bit flags are reserved and must currently always be unset (0).

The sixth byte in the header is of type `GTA_UINT8` and contains a recommendation for the compression type of the GTA, which must be one of the values defined in Tab. 2. Since compression is deprecated, the values of this field must be `GTA_NONE` for all new files.

If this value is `GTA_NONE`, then the GTA data must not be stored in chunks. If this value is not `GTA_NONE`, then the GTA data must be stored in chunks.

This value sets the recommended compression method for all following chunks in the GTA (header or data), but note that each chunk uses its own compression method which may differ from this recommendation.

At byte 7 of the header, a chunk list starts. This chunk list contains the information described in the next paragraphs.

The first information in the chunk list is the list of element components. It consists of zero or more component definitions. A component definition starts with one byte of type `GTA_UINT8` that contains one of the values defined in Tab. 1. If the

type is `GTA_BLOB`, then the component definition contains additional 8 bytes of type `GTA_UINT64` (subject to the endianness type of the GTA). This additional value stores the size in bytes of the component. A component definition that contains the special value 255 in its type field signals the end of the component list.

After the list of element components, the list of dimensions is stored. This list consists of zero or more dimension definitions. A dimension definition consists of 8 bytes of type `GTA_UINT64` (subject to the endianness type of the GTA). This value stores the size of the dimension, and must be greater than zero. A dimension definition that stores the special value 0 in its size field signals the end of the dimension list.

After the list of dimensions, the global tag list is stored. The global tag list stores each tag name and its tag value as-is (UTF-8 strings, zero terminated). An empty name (consisting only of the value 0) signals the end of the global tag list.

After the global tag list, the tag lists of all element components follows, and after that the tag lists of all dimensions. These tag lists are stored in the same manner as the global tag list.

After these tag list definitions, the GTA header chunk list ends, i.e. a single zero-sized chunk must signal the end of the header chunk list.

The GTA array data follows immediately after the header.

2.2 GTA Data

The array data itself is stored in a compact form without alignment offsets or other fill bytes of any kind. The linear format of a multidimensional array is the same as that of a C array when the first dimension is specified last (row-major for two-dimensional data). The following is an example for a three-dimensional array: `element_type array[size2][size1][size0]`.

The array data is stored as-is if the compression method recommended in the sixth byte of the header is `GTA_NONE` (which it is for all new files since compression is deprecated). Otherwise, it is stored in a chunk list.

When stored as-is, individual data elements can easily be accessed if the input is seekable.

Immediately after the array data, another GTA may follow.