Binary Document Format

Alex Good – alex@memoryandthought.me · Andrew Jeffery – andrewjeffery97@gmail.com | draft

Table of Contents

Introduction

Automerge is a library that allows people to collaboratively work together without a central co-ordination or a reliable connection. It is a specific implementation of of a conflict-free replicated data type (or CRDT).

This document describes the storage format used when serializing Automerge documents and changes for storage or transfer.

We strongly encourage people to use a library based on the reference implementation automerge-rs (which is available as a C shared library or a WebAssembly module for ease of integration). That said, this document

should let you get started building your own, or at least understanding how Automerge works.
The storage format is designed for compactness and speed of parsing. Automerge stores the full history of changes to the document: this is a large amount of data but in practice it is very repetitive and amenable to compression.

In addition to parsing the storage format, an Automerge library must resolve conflicts between concurrent operations in a consistent way. This document does not yet discuss how to do that (pull requests welcome :D), but the reference implementation should serve as a guide.

Terminology and Conventions

The key words “MUST”, “MUST NOT”, “REQUIRED”, “SHALL”, “SHALL NOT”, “SHOULD”, “SHOULD NOT”, “RECOMMENDED”, “NOT RECOMMENDED”, “MAY”, and “OPTIONAL” in this document are to be interpreted as described in [RFC2119] and [RFC8174] when, and only when, they appear in all capitals, as shown here.

Concepts

Document

An Automerge document is a collaboratively editable JSON-like structure. The serialized form of a document contains complete history of changes and operations that collaborators have applied.

Automerge documents have a root value that is a map from string keys to arbitrary values.

Change

A change is a group of operations that modify a document, analagous to a "commit" in a version control system like git.

Each change is made by an actor, and has a (possibly empty) set of predecessor changes. Changes have an optional wall-clock timestamp, to keep track of when a change was committed, and an optional message to describe meaningful changes.

A change is identified by its change hash which is the [SHA256] hash of the binary representation of the change.

Actor

An actor makes applies a linear sequence of operations to a document and commits them in a sequence of changes. Each actor has an actor ID that uniquely identifies it. An actor ID is an arbitrary sequence of bytes, which should be generated in a way that will not collide with other actors (the reference library generates 128-bit random identifiers).

There is a small amount of per-actor overhead, so if you have one process that edits a document several times sequentially, it is preferrable to re-use the same actor ID for each change.

Operation

An operation is an individual mutation made by an actor to a document. For example, setting a key to a value or inserting a character into some text.

An operation has an action which identifies what it does, and various fields depending on which action. For example a `"set"` operation has to identify the object being modified, the key in that object, and the new value.

Each operation is identified by an operation ID. An operation ID is a pair of (actor ID, counter), where the counter is a unique always-incrementing value per actor.

Object

An object represents a collaboratively editable value in an Automerge document. There are three kinds of object:

- `map` which maps string keys to values,
- `list` which is an ordered list of values
- `text` which is a collaboratively editable utf-8 string.

Each object is created by an operation with an `action` of `"makeMap"`, `"makeList"` or `"makeText"`, and is identified by its object ID. The object ID is the operation ID of the operation that created the object.

Each document has a root `map` which is identified by the object ID with a `null` actor id and `null` counter.

Value

Automerge objects are dynamically typed, and can contain any of the following kinds of value:

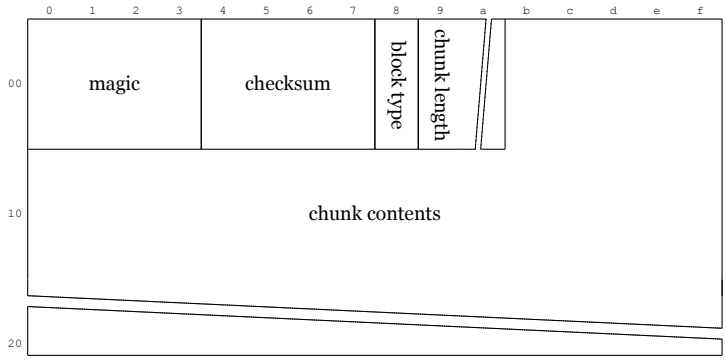
- `map`, `list`, `text` - the collaboratively editable objects
- `null` - an typed null
- `bool` - either `true` or `false`
- `float` - a 64-bit IEEE754 float
- `int` - a 64-bit signed int
- `uint` - a 64-bit unsigned int
- `string` - a utf-8 encoding string (possibly containing U+0000)
- `bytes` - an arbitrary sequence of bytes
- `timestamp` - a 64-bit signed integer representing milliseconds since the unix epoch

- `counter` - a 64-bit signed intenger that collaborators increment or decrement (instead of overwriting)

File structure

An Automerge file consists of one or more length delimited chunks. Implementations must attempt to read chunks until the end of the file.

Chunks



Field	Byte Length	Description
Magic bytes	4	The sequence [0x85, 0x6f, 0x4a, 0x83]
Checksum	4	Validates the integrity of the chunk
Chunk type	1	The type of this chunk
Chunk length	Variable (64-bit uLEB)	The length of the following chunk bytes

Chunk contents	Variable	The actual bytes for the chunk

If the first four bytes are not exactly the magic bytes implementations MUST abort.

The checksum is the first four bytes of the [SHA256] hash of the concatenation of the chunk type, chunk length and chunk contents fields. Implementations MUST abort reading if the checksum does not match.

Chunk Type

The chunk type is either:

Value	Type	Description
0x00	Document chunk	Contains a graph of related changes
0x01	Change chunk	Contains a single change and its operations
0x02	Compressed change chunk	Contains a single change deflate compressed

Document Chunk

The fields in a document chunk, in order, are:

Field	**Type**	**Description**
Actors	Array of Actor IDs	The actor IDs in sorted order
Heads	Array of Change Hashes	The hashes of the change graph in sorted

		order
Change columns metadata	Column Metadata	Description of the change columns
Operation columns metadata	Column Metadata	Description of the operation columns
Change columns	Column Data	The actual bytes for the change columns
Operation columns	Column Data	The actual bytes for the operation columns
Heads index	Heads Index	A lookup from change hash to change

A document contains a set of changes that represent the history of a collaboratively edited document. A document always contains a complete history of changes: for each change in the document, all the changes that were made to the document before that change was made are also included.

Document chunks use a columnar storage format for both changes and operations that assumes that the values of various fields are similar across adjacent changes and operations to optimize for high compression ratios and fast decoding.

Most fields are of arbitrary length, so parsing the document must proceed in order; for example it is not possible to know the length of the column fields until the column metadata has been parsed.

By implication, a document with no changes consists of `0x00 0x00 0x00 0x00`, as the counts of actors, heads, change columns and operations columns are all zero. With the chunk header, this gives a file consisting of the following bytes: `0x85 0x6f 0x4a 0x83 0xb8 0x1a 0x95 0x44 0x00 0x04 0x00 0x00 0x00 0x00`.

Change Chunk

The fields in a change chunk, in order, are:

Field	Type	Description
Dependencies	Array of Change Hashes	The set of changes that this change depends on
Actor length	64-bit uLEB	The length of the actor ID
Actor	bytes	The actor ID
Sequence number	64-bit uLEB	The sequence number
Start op	64-bit uLEB	The counter of the first op in this change
Time	64-bit LEB	The time this change was created in milliseconds since the unix epoch
Message length	64-bit uLEB	The length of the message in bytes
Message	UTF-8 encoded string	The message associated with this change
Other actors	Array of Actor IDs	Other actor IDs in this change
Operation columns metadata	Column Metadata	Description of the operation columns
Operation columns	Column Data	The actual bytes for the operation columns

Extra bytes	bytes	All data remaining in the chunk

A change chunk just contains a single change, its metadata and operations. It does not include any dependent changes, so you can only apply the change to a document that already contains those dependent changes.

Change chunks use a columnar storage format that assumes that the values of various fields are similar across adjacent operations to optimize for high compression ratios and fast decoding.

The extra bytes must be retained when processing changes. If future versions of automerge add new metadata to changes, this will allow old clients to collaborate with new clients without limiting which features the new clients can use.

Compressed Change Chunk

A compressed change chunk is a change chunk where the contents have been compressed using [DEFLATE]. The checksum of a compressed chunk is calculated as through the chunk was not compressed (with type = 1, the length of the uncompressed data, and the original uncompressed data).

For example if you have a change which has 372 bytes of data (excluding the header) which compresses to 323 bytes. If the original change chunk header consisted of the following bytes: `0x85 0x6f 0x4a 0x83 0x80 0xb7 0x5d 0x54 0x01 0xf4 0x02` , the compressed change chunk header would consist of: `0x85 0x6f 0x4a 0x83 0x80 0xb7 0x5d 0x54 0x02 0xc3 0x02`

To decode a compressed change chunk first decompress the chunk contents, and then construct a new chunk with the same magic bytes and checksum, but with type = 1 and the length of the decompressed data. Then parse that as a change chunk.

Simple types

uLEB

uLEB is an unsigned little endian base 128 value. This is a variable length encoding used throughout.

To encode a uLEB, represent the number in binary and pad it with leading zeros so that it has a length which is a multiple of 7. Take each group of 7 bytes from least-significant to most-significant and output them in bytes - the first bit of every byte is 1 except for the last byte which is 0.

- Unsigned ints 0 - 127 are stored as one byte: `0b00000000` - `0b01111111`
- Unsigned ints 128 - 16383 are stored as two bytes: `0b10000000` `0b00000001` - `0b11111111` `0b01111111` etc.

To decode a uLEB, read bytes up to and including the first byte with a 0 as the first bit. Take the latter 7 bits from each byte (the last byte contains the most significant bits, so you need to concatenate them in the opposite order to which the bytes are represented on disk).

Although uLEB encoding can support numbers of arbitrary bitsize, unsigned integers in Automerge must not exceed 64 bits. Implementations should fail to parse documents with uLEBs that decode to a value too large to be represented in a 64-bit unsigned integer.

Implementations must generate the shortest possible uLEB encodings, and should reject documents with overly long encodings. For example using the decoding rules above the bytes `0b10000000` `0b00000000` would be decoded as 0; but this is overly long: 0 can be represented in just one byte as `0b00000000`, so should be rejected.

LEB

LEB is a signed variant little endian base 128 value

To encode a uLEB, represent the number in twos complement, and sign-extend it so that it has a length which is a multiple of seven. If the

number is negative the padding will be of 1-bits and if the number is positive the padding will be 0-bits.

- `0` is represented as one byte: `0b0000000`
- Ints from 1 to 63 are represented as one byte: `0b00000001` - `0b00111111`
- Ints from -1 to -64 are represented as one byte: `0b01111111` - `0b010000000`
- Ints from 64 to 8191 are represented as two bytes: `0b11000000` `0b00000000` - `0b11111111` `0b00111111`
- Ints from -65 to -8192 are represented as two bytes: `0b10111111` `0b01111111` - `0b10000000` `0b01000000` etc.

To decode an LEB, read bytes up to and including the first byte with a 0 as the first bit. Take the latter 7 bits from each byte (the last byte contains the most signfiicant bits, so you need to concatenate them in the opposite order to which the bytes are represented on disk). If the first bit of your number is 1 (from the second bit of the last byte in encoded form) then you have a negative number and you can take twos complement to get to its absolute value; otherwise you have a positive number (or 0).

Although LEB encoding can support numbers of arbitrary bitsize, signed integers in Automerge must not exceed 64 bits. Implementations should fail to parse documents with uLEBs that decode to a value too large to be represented in a 64-bit signed integer.

Implementations must generate the shortest possible LEB for a given integer, and should reject documents with overly long encodings. For example the decoding rules above the bytes `0b11111111` `0b01111111` would be decoded as -1; but this is overly long: -1 can be represented as just one byte `0b01000000`, so should be rejected.

Change Hash

A change hash is the 32-byte [SHA256] hash of the concatenation of the chunk type (0x01) chunk length and chunk contents fields of a change

represented as a change chunk.
The first four bytes of the change hash are used as a checksum when a change chunk is serialized.

Action

The actions of the reference data model are encoded in the storage format as a byte as follows:

Byte	Action	Description
0x00	makeMap	Creates a new map object
0x01	set	Sets a key of a map, overwrites an item in a list, inserts an item in a list, or edits text
0x02	makeList	Creates a new list object
0x03	del	Unsets a key of a map, or removes an item from a list (reducing its length)
0x04	makeText	Creates a new text object
0x05	inc	Increments a counter stored in a map or a list

Future versions of automerge may add new actions, and implementations must preserve operations containing actions they don’t support when processing changes for forward compatibility.

Column Specification

Column specifications are a 32-bit uLEB interpreted as a bitfield:

- The least significant three bits encode the column type
- The 4th least significant bit is `1` if the column is [DEFLATE] compressed and `0` otherwise
- The remaining bits are the column ID

If the deflate bit is set then the column data must first be decompressed using DEFLATE before proceeding with decoding the values.

The DEFLATE bit is only permitted in document chunks, implementations must abort if they find compressed columns in change chunks.

The ID defines the purpose of the column for either Change Columns or Operation Columns, and implementations must preserve columns that they do not understand.

The column type specifies how the data in the column is encoded. The possible types are:

Value	Description
0	Group Column
1	Actor Column
2	uLEB Column
3	Delta Column
4	Boolean Column
5	String Column

6	Value Metadata Column
7	Value Column

Compound types

Array of Actor IDs

The actor ID array consists of a 64-bit uLEB giving the count of actor ids, followed by each actor ID as a length-prefixed byte array.

Each item in the array consists of a 64-bit uLEB giving the length in bytes, and then that number of bytes.

For example an array consisting of the single actor ID `[0xab, 0xcd, 0xef]` would be encoded as: `0x01 0x03 0xab 0xcd 0xef`.

Implementations must store actor ids lexicographically, and should error when reading a document with actor ids in the wrong order.

Array of Change Hashes

The heads array consists of a 64-bit uLEB N giving the count of heads, followed by N change hashes each exactly 32-bytes long.

For example an array consisting of the heads `f986a4318d1f1cc0e2e10e421e7a9a4cd0b70a89dae98bc1d76d789c2bf7904c` and `4355a46b19d348dc2f57c046f8ef63d4538ebb936000f3c9ee954a27460dd865` would be represented as `0x02 0xf9 0x86 ..{28 bytes elided).. 0x90 0x4c 0x43 0x55 ..{28 bytes elided}.. 0xd8 0x65`

Heads Index

The heads index provides a lookup table from the change hash to the change in a document. Very old automerge documents may be missing this field.

The index consists of N 64-bit uLEB's (one per head in the Heads array of the document chunk), and each uLEB gives the index of that head's change in the columnar change storage.

In a well-formed document, the change hash of the change indicated will match the change hash in the heads array, but implementations may chose to not validate this when parsing documents to avoid having to recompute every change hash.

Column Metadata

The column metadata consists of a 64-bit uLEB N giving the number of columns, followed by N pairs describing each columns in the chunks column data.

Field	Description
Column Specification	a 32-bit uLEB encoded Column Specification
Column Length	64-bit uLEB of the length (in bytes) of the column data block

The column specifications must be unique and sorted. Implementations must not include both an uncompressed and a compressed column with the same ID and type, and the column order should be sorted with the deflate bit set to 0.

A column that contains only null values, or is otherwise empty, should be omitted from the chunk. In this case there will be no column specification in the column metadata and no data in the column data.

In the case that there are no changes or operations at all, then the column metadata will be encoded as `0x00` to indicate that there are no columns at all, and there will be no column data in the chunk.

Column Data

Columns are stored one after the other with no separators or length indicators. The columns are stored in order they appear in the column metadata and each can be decoded according to its column specification.

All columns must have the same number of items (or the same number of arrays of items for grouped columns), though as they are compressed differently they may have vastly different byte counts.

For future compatibility it is important that programs which edit Automerge documents maintain all columns, even those that they don't understand the meaning of. When new changes or operations are added to a document with an unknown column a null should be added following the encoding rules of its specification.

Change Columns

The currently defined columns for changes in a document chunk are:

Name	Specification	ID	Type	Description
actor	1	0	Actor Column	The actor that made the change
sequence number	3	0	Delta Column	The sequence number for each change
maxOp	19	1	Delta Column	The largest counter that occurs in each change
time	35	2	Delta Column	The (optional) wallclock time at which

				each change was made
message	53	2	String Column	The (optional) commit message for each change
dependencies group	64	4	Group Column	The number of dependencies for each change
dependencies index	67	4	Grouped Delta Column	The indices of the changes this change depends on
extra metadata	86	5	Value Metadata Column	The metadata for any extra data for this change
extra data	87	5	Value Column	Any extra data for this change

Each value in the `dependencies index` column is an index into the changes that are stored in the document's columns. Implementations MUST abort if an index is out of bounds.

The `sequence number` of a change should be `1` if it is the first change by a given actor. Each subsequent change must have a sequence number exactly `1` higher than the previous change by the same actor. Implementations MUST abort if there are missing changes for a given actor ID.

The `maxOp` field of the change refers to the largest counter component of an operation ID in the set of operations in this change. For a given actor ID this must always increase. Implementations MUST abort if the `maxOp` of a change is not larger than all the `maxOp` of changes from that actor with smaller `seq`.

After decoding all the columns, and de-referencing indices into other columns, you will have an array of changes, where each change conceptually has the following fields:

Field	Type	Mapping
actor ID	array of bytes	The id of the actor that made the change
seq	64-bit uint	The sequence number of the change
ops	array of operations	The operations for this change (take all operations with counter greater the previous change's maxOp and less than or equal to this change's maxOp)
deps	array of changes	The changes this change depends on (look up each index in the dependencies index in this documents changes columns)
time	64-bit int	The (optional) wallclock time of the change

message	utf-8 string	The (optional) message of the change
extra data	any	The (optional) extra data (parse the extra data column according to the extra metadata column)

Operation Columns

The currently defined columns for operations are:

Field	Specification	ID	Type	Description
object actor ID	1	0	Actor Column	actor index of object ID each operation targets
object counter	2	0	uLEB Column	counter of the object ID each operation targets
key actor ID	17	1	Actor Column	actor of the operation ID of the key of each operation
key counter	19	1	uLEB Column	counter of the operation ID of the key of each operation