

# Package ‘automerge’

January 31, 2026

**Type** Package

**Title** R Bindings for 'Automerge' 'CRDT' Library

**Version** 0.2.0

**Description** Provides R bindings to the 'Automerge' Conflict-free Replicated Data Type ('CRDT') library. 'Automerge' enables automatic merging of concurrent changes without conflicts, making it ideal for distributed systems, collaborative applications, and offline-first architectures. The approach of local-first software was proposed in Kleppmann, M., Wiggins, A., van Hardenberg, P., McGranaghan, M. (2019) <[doi:10.1145/3359591.3359737](https://doi.org/10.1145/3359591.3359737)>. This package supports all 'Automerge' data types (maps, lists, text, counters) and provides both low-level and high-level synchronization protocols for seamless interoperability with 'JavaScript' and other 'Automerge' implementations.

**License** MIT + file LICENSE

**URL** <https://github.com/posit-dev/automerge-r>,  
<https://posit-dev.github.io/automerge-r/>

**BugReports** <https://github.com/posit-dev/automerge-r/issues>

**Depends** R (>= 4.2)

**Suggests** knitr, rmarkdown, testthat (>= 3.0.0)

**VignetteBuilder** knitr

**Config/build/compilation-database** true

**Config/Needs/website** tidyverse/tidytemplate

**Config/testthat/edition** 3

**Encoding** UTF-8

**RoxygenNote** 7.3.3

**SystemRequirements** 'automerge-c', or Cargo (Rust's package manager),  
rustc >= 1.80 and CMake >= 3.25 to build from package sources.

**NeedsCompilation** yes

**Author** Charlie Gao [aut, cre] (ORCID: <<https://orcid.org/0000-0002-0750-061X>>),  
Posit Software, PBC [cph, fnd] (ROR: <<https://ror.org/03wc8by49>>),  
Authors of the dependency Rust crates [cph] (see inst/AUTHORS file)

**Maintainer** Charlie Gao <charlie.gao@posit.co>

**Repository** CRAN

**Date/Publication** 2026-01-31 19:30:02 UTC

## Contents

am_apply_changes . . . . .	3
am_commit . . . . .	4
am_counter . . . . .	5
am_counter_increment . . . . .	5
am_create . . . . .	6
am_cursor . . . . .	7
am_cursor_position . . . . .	8
am_delete . . . . .	9
am_delete_path . . . . .	10
am_fork . . . . .	10
am_get . . . . .	11
am_get_actor . . . . .	12
am_get_actor_hex . . . . .	12
am_get_changes . . . . .	13
am_get_changes_added . . . . .	14
am_get_change_by_hash . . . . .	15
am_get_heads . . . . .	15
am_get_history . . . . .	16
am_get_last_local_change . . . . .	17
am_get_path . . . . .	18
am_insert . . . . .	19
am_keys . . . . .	19
am_length . . . . .	20
am_list . . . . .	21
am_load . . . . .	21
am_map . . . . .	22
am_mark . . . . .	23
am_marks . . . . .	24
am_marks_at . . . . .	25
am_merge . . . . .	26
am_put . . . . .	27
am_put_path . . . . .	28
am_rollback . . . . .	28
am_save . . . . .	29
am_set_actor . . . . .	30
am_sync . . . . .	30
am_sync_decode . . . . .	31
am_sync_encode . . . . .	32
am_sync_state_new . . . . .	33
am_text . . . . .	34
am_text_content . . . . .	34

<i>am_apply_changes</i>	3
am_text_splice . . . . .	35
am_text_update . . . . .	36
am_uint64 . . . . .	37
am_values . . . . .	38
as.character.am_text . . . . .	38
as.list.am_doc . . . . .	39
as_automerge . . . . .	40
automerge-constants . . . . .	41
extract-am_doc . . . . .	42
extract-am_object . . . . .	43
from_automerge . . . . .	44
length.am_doc . . . . .	44
length.am_object . . . . .	45
names.am_doc . . . . .	45
names.am_map . . . . .	46
replace-am_doc . . . . .	46
replace-am_object . . . . .	47
<b>Index</b>	<b>49</b>

---

<code>am_apply_changes</code>	<i>Apply changes to a document</i>
-------------------------------	------------------------------------

---

### Description

Applies a list of changes (obtained from `am_get_changes()`) to a document. This is useful for manually syncing changes or for applying changes received over a custom network protocol.

### Usage

```
am_apply_changes(doc, changes)
```

### Arguments

<code>doc</code>	An Automerge document
<code>changes</code>	A list of raw vectors (serialized changes) from <code>am_get_changes()</code>

### Value

The document `doc` (invisibly, for chaining)

**Examples**

```
# Create two documents
doc1 <- am_create()
doc2 <- am_create()

# Make changes in doc1
am_put(doc1, AM_ROOT, "x", 1)
am_commit(doc1)

# Get changes and apply to doc2
changes <- am_get_changes(doc1, NULL)
am_apply_changes(doc2, changes)

# Now doc2 has the same data as doc1
```

---

am\_commit

*Commit pending changes*


---

**Description**

Commits all pending operations in the current transaction, creating a new change in the document's history. Commits can include an optional message (like a git commit message) and timestamp.

**Usage**

```
am_commit(doc, message = NULL, time = NULL)
```

**Arguments**

doc	An Automerge document
message	Optional commit message (character string)
time	Optional timestamp (POSIXct). If NULL, uses current time.

**Value**

The document doc (invisibly)

**Examples**

```
doc <- am_create()
am_put(doc, AM_ROOT, "key", "value")
am_commit(doc, "Add initial data")

# Commit with specific timestamp
am_commit(doc, "Update", Sys.time())
```

---

am_counter	<i>Create an Automerge counter</i>
------------	------------------------------------

---

**Description**

Creates a counter value for use with Automerge. Counters are CRDT types that support conflict-free increment and decrement operations.

**Usage**

```
am_counter(value = 0L)
```

**Arguments**

value	Initial counter value (default 0)
-------	-----------------------------------

**Value**

An am\_counter object

**Examples**

```
doc <- am_create()
am_put(doc, AM_ROOT, "score", am_counter(0))
```

---

am_counter_increment	<i>Increment a counter value</i>
----------------------	----------------------------------

---

**Description**

Increments an Automerge counter by the specified delta. Counters are CRDT types that support concurrent increments from multiple actors. Unlike regular integers, counter increments are commutative and do not conflict when merged.

**Usage**

```
am_counter_increment(doc, obj, key, delta)
```

**Arguments**

doc	An Automerge document
obj	An Automerge object ID (map or list), or AM_ROOT for the document root
key	For maps: a character string key. For lists: an integer index (1-based)
delta	Integer value to add to the counter (can be negative)

## Details

The delta can be negative to decrement the counter.

## Value

The document (invisibly), allowing for chaining with pipes

## Examples

```
# Counter in document root (map)
doc <- am_create()
doc$score <- am_counter(0)
am_counter_increment(doc, AM_ROOT, "score", 10)
doc$score # 10

am_counter_increment(doc, AM_ROOT, "score", 5)
doc$score # 15

# Decrement with negative delta
am_counter_increment(doc, AM_ROOT, "score", -3)
doc$score # 12

# Counter in a nested map
doc$stats <- am_map(views = am_counter(0))
stats_obj <- doc$stats
am_counter_increment(doc, stats_obj, "views", 100)

# Counter in a list (1-based indexing)
doc$counters <- list(am_counter(0), am_counter(5))
counters_obj <- doc$counters
am_counter_increment(doc, counters_obj, 1, 1) # Increment first counter
am_counter_increment(doc, counters_obj, 2, 2) # Increment second counter
```

---

am\_create

*Create a new Automerge document*

---

## Description

Creates a new Automerge document with an optional custom actor ID. If no actor ID is provided, a random one is generated.

## Usage

```
am_create(actor_id = NULL)
```

**Arguments**

actor\_id            Optional actor ID. Can be:

- NULL (default) - Generate random actor ID
- Character string - Hex-encoded actor ID
- Raw vector - Binary actor ID bytes

**Value**

An external pointer to the Automerge document with class `c("am_doc", "automerger")`.

**Thread Safety**

The automerge package is NOT thread-safe. Do not access the same document from multiple R threads concurrently. Each thread should create its own document with `am_create()` and synchronize changes via `am_sync_*()` functions after thread completion.

**Examples**

```
# Create document with random actor ID
doc <- am_create()

# Create with custom hex actor ID
doc2 <- am_create("0123456789abcdef0123456789abcdef")

# Create with raw bytes actor ID
actor_bytes <- as.raw(1:16)
doc3 <- am_create(actor_bytes)
```

---

am\_cursor

*Create a cursor at a position in a text object*


---

**Description**

Cursors provide stable references to positions within text objects that automatically adjust as the text is edited. This enables features like maintaining selection positions across concurrent edits in collaborative editing scenarios.

**Usage**

```
am_cursor(obj, position)
```

**Arguments**

obj                    An Automerge object ID (must be a text object)

position               Integer position in the text (0-based inter-character position)

**Value**

An am\_cursor object (external pointer) that can be used with `am_cursor_position()` to retrieve the current position

**Indexing Convention**

**Cursor positions use 0-based indexing** (unlike list indices which are 1-based). This is because positions specify locations **between** characters, not the characters themselves:

- Position 0 = before the first character
- Position 1 = between 1st and 2nd characters
- Position 5 = after the 5th character

For the text "Hello":

```
H e l l o
0 1 2 3 4 5 <- positions (0-based, between characters)
```

This matches `am_text_splice()` behavior. Positions count Unicode code points (characters), not bytes.

**Examples**

```
doc <- am_create()
am_put(doc, AM_ROOT, "text", am_text("Hello World"))
text_obj <- am_get(doc, AM_ROOT, "text")

# Create cursor at position 5 (after "Hello", before " ")
cursor <- am_cursor(text_obj, 5)

# Modify text before cursor
am_text_splice(text_obj, 0, 0, "Hi ")

# Cursor position automatically adjusts
new_pos <- am_cursor_position(cursor)
new_pos # 8 (cursor moved by 3 characters)
```

---

am\_cursor\_position      *Get the current position of a cursor*

---

**Description**

Retrieves the current position of a cursor within a text object. The position automatically adjusts as text is inserted or deleted before the cursor's original position. The cursor remembers which text object it was created for, so you only need to pass the cursor itself.

**Usage**

```
am_cursor_position(cursor)
```



**Arguments**

cursor            An am\_cursor object created by [am\\_cursor\(\)](#)

**Value**

Integer position (0-based inter-character position) where the cursor currently points. See [am\\_cursor\(\)](#) for indexing details.

**Examples**

```
doc <- am_create()
am_put(doc, AM_ROOT, "text", am_text("Hello World"))
text_obj <- am_get(doc, AM_ROOT, "text")

# Create cursor
cursor <- am_cursor(text_obj, 5)

# Get position
pos <- am_cursor_position(cursor)
pos # 5
```

---

am\_delete

*Delete a key from a map or element from a list*


---

**Description**

Removes a key-value pair from a map or an element from a list.

**Usage**

```
am_delete(doc, obj, key)
```

**Arguments**

doc            An Automerge document  
obj            An Automerge object ID (from nested object), or AM\_ROOT for the document root  
key            For maps: character string key to delete. For lists: numeric index (1-based, like R vectors) to delete

**Value**

The document doc (invisibly)

**Examples**

```
doc <- am_create()
am_put(doc, AM_ROOT, "temp", "value")
am_delete(doc, AM_ROOT, "temp")
```

am\_delete\_path            *Delete value at path*

---

**Description**

Delete a value from an Automerge document using a path vector.

**Usage**

```
am_delete_path(doc, path)
```

**Arguments**

doc	An Automerge document
path	Character vector, numeric vector, or list of mixed types specifying the path to the value to delete

**Value**

The document (invisibly)

**Examples**

```
doc <- am_create()
am_put_path(doc, c("user", "address", "city"), "NYC")
am_put_path(doc, c("user", "name"), "Alice")

# Delete nested key
am_delete_path(doc, c("user", "address"))

# Address should be gone
am_get_path(doc, c("user", "address")) # NULL
```

---

am\_fork                    *Fork an Automerge document*

---

**Description**

Creates a fork of an Automerge document at the current heads or at a specific point in history. The forked document shares history with the original up to the fork point but can diverge afterwards.

**Usage**

```
am_fork(doc, heads = NULL)
```

**Arguments**

doc	An Automerge document
heads	Optional list of change hashes to fork at a specific point in the document's history. If NULL (default) or an empty list, forks at current heads. Each hash should be a raw vector (32 bytes).

**Value**

A new Automerge document (fork of the original)

**Examples**

```
doc1 <- am_create()
doc2 <- am_fork(doc1)

# Now doc1 and doc2 can diverge independently
```

---

am\_get *Get a value from an Automerge map or list*

---

**Description**

Retrieves a value from an Automerge map or list. Returns NULL if the key or index doesn't exist.

**Usage**

```
am_get(doc, obj, key)
```

**Arguments**

doc	An Automerge document
obj	An Automerge object ID (from nested object), or AM_ROOT for the document root
key	For maps: character string key. For lists: numeric index (1-based). Returns NULL for indices $\leq 0$ or beyond list length.

**Value**

The value at the specified key/position, or NULL if not found. Nested objects are returned as am\_object instances.

**Examples**

```
doc <- am_create()
am_put(doc, AM_ROOT, "name", "Alice")

name <- am_get(doc, AM_ROOT, "name")
name # "Alice"
```

---

am_get_actor	<i>Get the actor ID of a document</i>
--------------	---------------------------------------

---

**Description**

Returns the actor ID of an Automerge document as a raw vector. The actor ID uniquely identifies the editing session that created changes in the document.

**Usage**

```
am_get_actor(doc)
```

**Arguments**

doc                    An Automerge document

**Details**

For a hex string representation, use [am\\_get\\_actor\\_hex\(\)](#).

**Value**

A raw vector containing the actor ID bytes

**Examples**

```
doc <- am_create()
actor <- am_get_actor(doc)

# Use am_get_actor_hex() for display
actor_hex <- am_get_actor_hex(doc)
cat("Actor ID:", actor_hex, "\n")
```

---

am_get_actor_hex	<i>Get the actor ID as a hex string</i>
------------------	---

---

**Description**

Returns the actor ID of an Automerge document as a hex-encoded string. This is more efficient than converting the raw bytes returned by [am\\_get\\_actor\(\)](#) using R-level string operations.

**Usage**

```
am_get_actor_hex(doc)
```

**Arguments**

doc                    An Automerge document

**Value**

A character string containing the hex-encoded actor ID

**Examples**

```
doc <- am_create()
actor_hex <- am_get_actor_hex(doc)
cat("Actor ID:", actor_hex, "\n")
```

---

am_get_changes	<i>Get changes since specified heads</i>
----------------	--

---

**Description**

Returns all changes that have been made to the document since the specified heads. If heads is NULL, returns all changes in the document's history.

**Usage**

```
am_get_changes(doc, heads = NULL)
```

**Arguments**

doc                    An Automerge document

heads                  A list of raw vectors (change hashes) returned by am\_get\_heads(), or NULL to get all changes.

**Details**

Changes are returned as serialized raw vectors that can be transmitted over the network and applied to other documents using am\_apply\_changes().

**Value**

A list of raw vectors, each containing a serialized change.

**Examples**

```
doc <- am_create()
am_put(doc, AM_ROOT, "x", 1)
am_commit(doc)

# Get all changes
all_changes <- am_get_changes(doc, NULL)
cat("Document has", length(all_changes), "change(s)\n")
```

---

am\_get\_changes\_added *Get changes in one document that are not in another*

---

### Description

Compares two documents and returns the changes that exist in doc2 but not in doc1. This is useful for determining what changes need to be applied to bring doc1 up to date with doc2, or for implementing custom synchronization logic.

### Usage

```
am_get_changes_added(doc1, doc2)
```

### Arguments

doc1	An Automerge document (base/reference document)
doc2	An Automerge document (comparison document)

### Value

A list of raw vectors, where each vector is a serialized change that exists in doc2 but not in doc1. Returns an empty list if doc1 already contains all changes from doc2.

### Examples

```
# Create two independent documents
doc1 <- am_create()
doc1$x <- 1
am_commit(doc1, "Add x")

doc2 <- am_create()
doc2$y <- 2
am_commit(doc2, "Add y")

# Find changes in doc2 that aren't in doc1
changes <- am_get_changes_added(doc1, doc2)
length(changes) # 1 change

# Apply those changes to doc1
am_apply_changes(doc1, changes)

# Now doc1 has both x and y
names(doc1) # "x" "y"
```

---

am\_get\_change\_by\_hash *Get a specific change by its hash*

---

**Description**

Retrieves a change from the document's history by its unique hash identifier. The hash is typically obtained from `am_get_heads()` or `am_get_changes()`.

**Usage**

```
am_get_change_by_hash(doc, hash)
```

**Arguments**

doc	An Automerge document
hash	A raw vector containing the change hash (must be exactly 32 bytes)

**Value**

A raw vector containing the serialized change, or NULL if the change hash is not found in the document.

**Examples**

```
doc <- am_create()
doc$key <- "value"
am_commit(doc, "Add key")

# Get the current heads (change hashes)
heads <- am_get_heads(doc)
head_hash <- heads[[1]]

# Retrieve the change by its hash
change <- am_get_change_by_hash(doc, head_hash)
str(change) # Raw vector
```

---

am\_get\_heads *Get the current heads of a document*

---

**Description**

Returns the current "heads" of the document - the hashes of the most recent changes. These identify the current state of the document and can be used for history operations.

**Usage**

```
am_get_heads(doc)
```

**Arguments**

doc                    An Automerge document

**Value**

A list of raw vectors, each containing a change hash. Usually there is only one head, but after concurrent edits there may be multiple heads until they are merged by a subsequent commit.

**Examples**

```
doc <- am_create()
am_put(doc, AM_ROOT, "x", 1)
am_commit(doc)

heads <- am_get_heads(doc)
cat("Document has", length(heads), "head(s)\n")
```

---

am\_get\_history                    *Get document history*

---

**Description**

Returns the full change history of the document as a list of change metadata. This provides a simpler interface than `am_get_changes()` for examining document history without needing to work with serialized changes directly.

**Usage**

```
am_get_history(doc)
```

**Arguments**

doc                    An Automerge document

**Details**

**Note:** A future implementation will add detailed change introspection functions to extract metadata like commit messages, timestamps, actor IDs, etc.

**Value**

A list of raw vectors (serialized changes), one for each change in the document's history, in chronological order.



**Examples**

```
doc <- am_create()
am_put(doc, AM_ROOT, "x", 1)
am_commit(doc, "Initial")
am_put(doc, AM_ROOT, "x", 2)
am_commit(doc, "Update")

history <- am_get_history(doc)
cat("Document history contains", length(history), "change(s)\n")
```

---

am\_get\_last\_local\_change

*Get the last change made by the local actor*

---

**Description**

Returns the most recent change created by this document's actor. Useful for tracking local changes or implementing undo/redo functionality.

**Usage**

```
am_get_last_local_change(doc)
```

**Arguments**

doc                    An Automerge document

**Value**

A raw vector containing the serialized change, or NULL if no local changes have been made.

**Examples**

```
doc <- am_create()

# Initially, no local changes
am_get_last_local_change(doc) # NULL

# Make a change
doc$key <- "value"
am_commit(doc, "Add key")

# Now we have a local change
change <- am_get_last_local_change(doc)
str(change) # Raw vector
```

---

am_get_path	<i>Navigate deep structures with path</i>
-------------	---

---

## Description

Get a value from an Automerge document using a path vector. The path can contain character keys (for maps), numeric indices (for lists, 1-based), or a mix of both.

## Usage

```
am_get_path(doc, path)
```

## Arguments

doc	An Automerge document
path	Character vector, numeric vector, or list of mixed types specifying the path to navigate

## Value

The value at the path, or NULL if not found

## Examples

```
doc <- am_create()
am_put(doc, AM_ROOT, "user", list(
  name = "Alice",
  address = list(city = "NYC", zip = 10001L)
))

# Navigate to nested value
am_get_path(doc, c("user", "address", "city")) # "NYC"

# Mixed navigation (map key, then list index)
doc$users <- list(
  list(name = "Bob"),
  list(name = "Carol")
)
am_get_path(doc, list("users", 1, "name")) # "Bob"
```

---

am_insert	<i>Insert a value into an Automerge list</i>
-----------	--

---

**Description**

This is an alias for `am_put()` with insert semantics for lists. For lists, `am_put()` with a numeric index replaces the element at that index, while `am_insert()` shifts elements to make room.

**Usage**

```
am_insert(doc, obj, pos, value)
```

**Arguments**

doc	An Automerge document
obj	An Automerge object ID (must be a list)
pos	Numeric index (1-based, like R vectors) where to insert, or "end" to append
value	The value to insert

**Value**

The document doc (invisibly)

**Examples**

```
doc <- am_create()
# Create a list and get it
am_put(doc, AM_ROOT, "items", AM_OBJ_TYPE_LIST)
items <- am_get(doc, AM_ROOT, "items")

# Insert items
am_insert(doc, items, "end", "first")
am_insert(doc, items, "end", "second")
```

---

am_keys	<i>Get all keys from an Automerge map</i>
---------	---

---

**Description**

Returns a character vector of all keys in a map.

**Usage**

```
am_keys(doc, obj)
```

**Arguments**

doc            An Automerge document  
obj            An Automerge object ID (must be a map), or AM\_ROOT for the document root

**Value**

Character vector of keys (empty if map is empty)

**Examples**

```
doc <- am_create()
am_put(doc, AM_ROOT, "a", 1)
am_put(doc, AM_ROOT, "b", 2)

keys <- am_keys(doc, AM_ROOT)
keys # c("a", "b")
```

---

am\_length

*Get the length of an Automerge map or list*

---

**Description**

Returns the number of key-value pairs in a map or elements in a list.

**Usage**

```
am_length(doc, obj)
```

**Arguments**

doc            An Automerge document  
obj            An Automerge object ID, or AM\_ROOT for the document root

**Value**

Integer length/size

**Examples**

```
doc <- am_create()
am_put(doc, AM_ROOT, "a", 1)
am_put(doc, AM_ROOT, "b", 2)

len <- am_length(doc, AM_ROOT)
len # 2
```

---

am_list	<i>Create an Automerge list</i>
---------	---------------------------------

---

**Description**

Creates an R list with explicit Automerge list type. Use this when you need to create an empty list or force list type interpretation.

**Usage**

```
am_list(...)
```

**Arguments**

... Elements to include in the list

**Value**

A list with class `am_list_type`

**Examples**

```
# Empty list (avoids ambiguity)
am_list()

# Populated list
am_list("a", "b", "c")
```

---

am_load	<i>Load an Automerge document from binary format</i>
---------	--

---

**Description**

Deserializes an Automerge document from the standard binary format. The binary format is compatible across all Automerge implementations (JavaScript, Rust, etc.).

**Usage**

```
am_load(data)
```

**Arguments**

data A raw vector containing a serialized Automerge document

**Value**

An external pointer to the Automerge document with class `c("am_doc", "automerge")`.

### Examples

```
# Create, save, and reload
doc1 <- am_create()
bytes <- am_save(doc1)
doc2 <- am_load(bytes)

# Save to and load from file
file <- tempfile()
writeBin(am_save(doc1), file)

doc <- am_load(readBin(file, "raw", 1e5))
unlink(file)
```

---

am\_map

*Create an Automerge map*

---

### Description

Creates an R list with explicit Automerge map type. Use this when you need to create an empty map or force map type interpretation.

### Usage

```
am_map(...)
```

### Arguments

...                   Named elements to include in the map

### Value

A named list with class `am_map_type`

### Examples

```
# Empty map (avoids ambiguity)
am_map()

# Populated map
am_map(key1 = "value1", key2 = "value2")
```

---

am_mark	<i>Create a mark on a text range</i>
---------	--------------------------------------

---

### Description

Marks attach metadata or formatting information to a range of text. Unlike simple annotations, marks are CRDT-aware and merge correctly across concurrent edits.

### Usage

```
am_mark(obj, start, end, name, value, expand = AM_MARK_EXPAND_NONE)
```

### Arguments

obj	An Automerge object ID (must be a text object)
start	Integer start position (0-based inter-character position, inclusive)
end	Integer end position (0-based inter-character position, exclusive)
name	Character string identifying the mark (e.g., "bold", "comment")
value	The mark's value (any Automerge-compatible type: NULL, logical, integer, numeric, character, raw, POSIXct, or am_counter)
expand	Character string controlling mark expansion behavior when text is inserted at boundaries. Options: <b>"none"</b> Mark does not expand (default) <b>"before"</b> Mark expands to include text inserted before start <b>"after"</b> Mark expands to include text inserted after end <b>"both"</b> Mark expands in both directions Use the constants <a href="#">AM_MARK_EXPAND_NONE</a> , <a href="#">AM_MARK_EXPAND_BEFORE</a> , <a href="#">AM_MARK_EXPAND_AFTER</a> , or <a href="#">AM_MARK_EXPAND_BOTH</a> .

### Value

The text object obj (invisibly)

### Indexing Convention

**Mark positions use 0-based indexing** (unlike list indices which are 1-based). Positions specify locations **between** characters. The range [start, end) includes start but excludes end.

For the text "Hello":

```

H e l l o
0 1 2 3 4 5 <- positions (0-based, between characters)
```

Marking positions 0 to 5 marks all 5 characters. Marking 0 to 3 marks "Hel". Positions count Unicode code points (characters), not bytes.

## Expand Behavior

The expand parameter controls what happens when text is inserted exactly at the mark boundaries:

- "none": New text is never included in the mark
- "before": Text inserted at start is included
- "after": Text inserted at end is included
- "both": Text inserted at either boundary is included

## Examples

```
doc <- am_create()
am_put(doc, AM_ROOT, "text", am_text("Hello World"))
text_obj <- am_get(doc, AM_ROOT, "text")

# Mark "Hello" as bold (positions 0-4, characters 0-4)
am_mark(text_obj, 0, 5, "bold", TRUE)

# Mark "World" as italic with expansion
am_mark(text_obj, 6, 11, "italic", TRUE,
        expand = AM_MARK_EXPAND_BOTH)

# Get all marks
marks <- am_marks(text_obj)
marks
```

---

am\_marks

*Get all marks in a text object*

---

## Description

Retrieves all marks (formatting/metadata annotations) present in a text object at a specific document state.

## Usage

```
am_marks(obj)
```

## Arguments

obj                    An Automerge object ID (must be a text object)

## Value

A list of marks, where each mark is a list with fields:

**name** Character string identifying the mark

**value** The mark's value (various types supported)



**start** Integer start position (0-based inter-character position, inclusive)

**end** Integer end position (0-based inter-character position, exclusive)

Returns an empty list if no marks are present. See [am\\_mark\(\)](#) for indexing details.

### Examples

```
doc <- am_create()
am_put(doc, AM_ROOT, "text", am_text("Hello World"))
text_obj <- am_get(doc, AM_ROOT, "text")

am_mark(text_obj, 0, 5, "bold", TRUE)
am_mark(text_obj, 6, 11, "italic", TRUE)

marks <- am_marks(text_obj)
marks
# List of 2 marks with name, value, start, end
```

---

am_marks_at	<i>Get marks at a specific position</i>
-------------	---

---

### Description

Retrieves marks that include a specific position in a text object. This function efficiently filters marks at the C level, avoiding the overhead of converting all marks to R objects.

### Usage

```
am_marks_at(obj, position)
```

### Arguments

obj	An Automerge object ID (must be a text object)
position	Integer position (0-based inter-character position) to query. See <a href="#">am_mark()</a> for indexing details.

### Value

A list of marks that include the specified position. Returns an empty list if no marks cover that position.

### Examples

```
doc <- am_create()
am_put(doc, AM_ROOT, "text", am_text("Hello World"))
text_obj <- am_get(doc, AM_ROOT, "text")

am_mark(text_obj, 0, 5, "bold", TRUE)
am_mark(text_obj, 2, 7, "underline", TRUE)
```

```
# Get marks at position 3 (inside "Hello")
marks_at_3 <- am_marks_at(text_obj, 3)
marks_at_3
# List of 2 marks (both "bold" and "underline" include position 3)
```

---

am\_merge

*Merge changes from another document*

---

## Description

Merges all changes from another Automerge document into this one. This is a one-way merge: changes flow from other into doc, but other is not modified. For bidirectional synchronization, use [am\\_sync\(\)](#).

## Usage

```
am_merge(doc, other)
```

## Arguments

doc	Target document (will receive changes)
other	Source document (provides changes)

## Value

The target document doc (invisibly)

## Examples

```
doc1 <- am_create()
doc2 <- am_create()

# Make changes in each document
am_put(doc1, AM_ROOT, "x", 1)
am_put(doc2, AM_ROOT, "y", 2)

# Merge doc2's changes into doc1
am_merge(doc1, doc2)
# Now doc1 has both x and y
```

---

`am_put`*Put a value into an Automerge map or list*

---

## Description

Inserts or updates a value in an Automerge map or list. The function automatically dispatches to the appropriate operation based on the object type and key/position type.

## Usage

```
am_put(doc, obj, key, value)
```

## Arguments

<code>doc</code>	An Automerge document
<code>obj</code>	An Automerge object ID (from nested object), or <code>AM_ROOT</code> for the document root
<code>key</code>	For maps: character string key. For lists: numeric index (1-based) or "end" to append
<code>value</code>	The value to store. Supported types: <ul style="list-style-type: none"><li>• <code>NULL</code> - stores null</li><li>• <code>Logical</code> - stores boolean (must be scalar)</li><li>• <code>Integer</code> - stores integer (must be scalar)</li><li>• <code>Numeric</code> - stores double (must be scalar)</li><li>• <code>Character</code> - stores string (must be scalar)</li><li>• <code>Raw</code> - stores bytes</li><li>• <code>AM_OBJ_TYPE_LIST/MAP/TEXT</code> - creates nested object</li></ul>

## Value

The document `doc` (invisibly).

## Examples

```
doc <- am_create()

# Put values in root map (returns doc invisibly)
am_put(doc, AM_ROOT, "name", "Alice")
am_put(doc, AM_ROOT, "age", 30L)
am_put(doc, AM_ROOT, "active", TRUE)

# Create nested list and retrieve it
am_put(doc, AM_ROOT, "items", AM_OBJ_TYPE_LIST)
items <- am_get(doc, AM_ROOT, "items")
```

---

am_put_path	<i>Set value at path</i>
-------------	--------------------------

---

**Description**

Set a value in an Automerge document using a path vector. Can optionally create intermediate objects automatically.

**Usage**

```
am_put_path(doc, path, value, create_intermediate = TRUE)
```

**Arguments**

doc	An Automerge document
path	Character vector, numeric vector, or list of mixed types specifying the path to the value
value	Value to set at the path
create_intermediate	Logical. If TRUE, creates intermediate maps as needed. Default TRUE.

**Value**

The document (invisibly)

**Examples**

```
doc <- am_create()

# Create nested structure with automatic intermediate objects
am_put_path(doc, c("user", "address", "city"), "Boston")
am_put_path(doc, c("user", "address", "zip"), 02101L)
am_put_path(doc, c("user", "name"), "Alice")

# Verify
am_get_path(doc, c("user", "address", "city")) # "Boston"
```

---

am_rollback	<i>Roll back pending operations</i>
-------------	-------------------------------------

---

**Description**

Cancels all pending operations in the current transaction without committing them. This allows you to discard changes since the last commit.

**Usage**

```
am_rollback(doc)
```

**Arguments**

doc                    An Automerge document

**Value**

The document doc (invisibly)

**Examples**

```
doc <- am_create()
am_put(doc, AM_ROOT, "key", "value")
# Changed my mind, discard the put
am_rollback(doc)
```

---

am\_save                    *Save an Automerge document to binary format*

---

**Description**

Serializes an Automerge document to the standard binary format, which can be saved to disk or transmitted over a network. The binary format is compatible across all Automerge implementations (JavaScript, Rust, etc.).

**Usage**

```
am_save(doc)
```

**Arguments**

doc                    An Automerge document (created with am\_create() or am\_load())

**Value**

A raw vector containing the serialized document

**Examples**

```
doc <- am_create()
bytes <- am_save(doc)

# Save to file
file <- tempfile()
writeBin(am_save(doc), file)
unlink(file)
```

---

am_set_actor	<i>Set the actor ID of a document</i>
--------------	---------------------------------------

---

### Description

Sets the actor ID for an Automerge document. This should typically be done before making any changes. Changing the actor ID mid-session is not recommended as it can complicate change attribution.

### Usage

```
am_set_actor(doc, actor_id)
```

### Arguments

doc	An Automerge document
actor_id	The new actor ID. Can be: <ul style="list-style-type: none"><li>• NULL - Generate new random actor ID</li><li>• Character string - Hex-encoded actor ID</li><li>• Raw vector - Binary actor ID bytes</li></ul>

### Value

The document doc (invisibly)

### Examples

```
doc <- am_create()

# Set custom actor ID from hex string
am_set_actor(doc, "0123456789abcdef0123456789abcdef")

# Generate new random actor ID
am_set_actor(doc, NULL)
```

---

am_sync	<i>Bidirectional synchronization</i>
---------	--------------------------------------

---

### Description

Automatically synchronizes two documents by exchanging messages until they converge to the same state. This is a high-level convenience function that handles the entire sync protocol automatically.

**Usage**

```
am_sync(doc1, doc2)
```

**Arguments**

doc1	First Automerge document
doc2	Second Automerge document

**Details**

The function exchanges sync messages back and forth between the two documents until both sides report no more messages to send (`am_sync_encode()` returns NULL). The Automerge sync protocol is mathematically guaranteed to converge.

**Value**

An integer indicating the number of sync rounds completed (invisibly). Both documents are modified in place to include each other's changes.

**Examples**

```
# Create two documents with different changes
doc1 <- am_create()
doc2 <- am_create()

# Make changes in each document
am_put(doc1, AM_ROOT, "x", 1)
am_put(doc2, AM_ROOT, "y", 2)

# Synchronize them (documents modified in place)
rounds <- am_sync(doc1, doc2)
cat("Synced in", rounds, "rounds\n")

# Now both documents have both x and y
```

---

am_sync_decode	<i>Receive and apply a sync message</i>
----------------	---

---

**Description**

Receives a synchronization message from a peer and applies the changes to the local document. This updates both the document and the sync state to reflect the received changes.

**Usage**

```
am_sync_decode(doc, sync_state, message)
```

**Arguments**

doc	An Automerge document
sync_state	A sync state object (created with am_sync_state_new())
message	A raw vector containing an encoded sync message

**Value**

The document doc (invisibly, for chaining)

**Examples**

```
doc <- am_create()
sync_state <- am_sync_state_new()

# Receive message from peer
# message <- ... (received from network)
# am_sync_decode(doc, sync_state, message)
```

---

am_sync_encode	<i>Generate a sync message</i>
----------------	--------------------------------

---

**Description**

Generates a synchronization message to send to a peer. This message contains the changes that the peer needs to bring their document up to date with yours.

**Usage**

```
am_sync_encode(doc, sync_state)
```

**Arguments**

doc	An Automerge document
sync_state	A sync state object (created with am_sync_state_new())

**Details**

If the function returns NULL, it means there are no more messages to send (synchronization is complete from this side).

**Value**

A raw vector containing the encoded sync message, or NULL if no message needs to be sent.



## Examples

```
doc <- am_create()
sync_state <- am_sync_state_new()

# Generate first sync message
msg <- am_sync_encode(doc, sync_state)
if (!is.null(msg)) {
  # Send msg to peer...
}
```

---

am_sync_state_new	<i>Create a new sync state</i>
-------------------	--------------------------------

---

## Description

Creates a new synchronization state for managing communication with a peer. The sync state tracks what changes have been sent and received, enabling efficient incremental synchronization.

## Usage

```
am_sync_state_new()
```

## Details

**IMPORTANT:** Sync state is document-independent. The same sync state is used across multiple sync message exchanges with a specific peer. The document is passed separately to `am_sync_encode()` and `am_sync_decode()`.

## Value

An external pointer to the sync state with class "am\_syncstate".

## Examples

```
# Create two documents
doc1 <- am_create()
doc2 <- am_create()

# Create sync states for each peer
sync1 <- am_sync_state_new()
sync2 <- am_sync_state_new()

# Use with am_sync_encode() and am_sync_decode()
```

---

am_text	<i>Create an Automerge text object</i>
---------	--

---

**Description**

Creates a text object for collaborative character-level editing. Unlike regular strings (which use last-write-wins semantics), text objects support character-level CRDT merging of concurrent edits, cursor stability, and marks/formatting.

**Usage**

```
am_text(initial = "")
```

**Arguments**

initial	Initial text content (default "")
---------	-----------------------------------

**Details**

Use text objects for collaborative document editing. Use regular strings for metadata, labels, and IDs (99\

**Value**

A character vector with class `am_text_type`

**Examples**

```
# Empty text object
am_text()

# Text with initial content
am_text("Hello, World!")
```

---

am_text_content	<i>Get text content from a text object</i>
-----------------	--

---

**Description**

Retrieve the full text content from a text object as a string.

**Usage**

```
am_text_content(text_obj)
```

**Arguments**

text\_obj      An Automerge text object ID

**Value**

Character string with the full text

**Examples**

```
doc <- am_create()
am_put(doc, AM_ROOT, "doc", am_text("Hello"))
text_obj <- am_get(doc, AM_ROOT, "doc")

text <- am_text_content(text_obj)
text # "Hello"
```

---

am\_text\_splice      *Splice text in a text object*

---

**Description**

Insert or delete characters in a text object. This is the primary way to edit text CRDT objects.

**Usage**

```
am_text_splice(text_obj, pos, del_count, text)
```

**Arguments**

text\_obj      An Automerge text object ID  
 pos            Character position to start splice (0-based inter-character position)  
 del\_count     Number of characters to delete (counts Unicode code points)  
 text          Text to insert

**Value**

The text object text\_obj (invisibly)

**Indexing Convention**

**Text positions use 0-based indexing** (unlike list indices which are 1-based). This is because positions specify locations **between** characters, not the characters themselves:

- Position 0 = before the first character
- Position 1 = between 1st and 2nd characters
- Position 5 = after the 5th character

For the text "Hello":

```
H e l l o
0 1 2 3 4 5 <- positions (0-based, between characters)
```

Positions count Unicode code points (characters), not bytes. The word "Français" counts as 8 characters, matching R's `nchar()` behavior.

## Examples

```
doc <- am_create()
am_put(doc, AM_ROOT, "doc", am_text("Hello"))
text_obj <- am_get(doc, AM_ROOT, "doc")

# Insert " World" at position 5 (after "Hello")
am_text_splice(text_obj, 5, 0, " World")

# Get the full text
am_text_content(text_obj) # "Hello World"

# Works naturally with multibyte characters
am_put(doc, AM_ROOT, "greet", am_text(""))
text_obj2 <- am_get(doc, AM_ROOT, "greet")
am_text_splice(text_obj2, 0, 0, "Column café")
# Position 11 is after "café" (character index, not bytes)
am_text_splice(text_obj2, 11, 0, "!")
am_text_content(text_obj2) # "Column café!"
```

---

am_text_update	<i>Update text content</i>
----------------	----------------------------

---

## Description

An optimized function for collaborative editing that computes the minimal diff between old and new text and applies it directly to the text object. This avoids intermediate R object allocation, making it more efficient than separate diff computation and splice operations.

## Usage

```
am_text_update(text_obj, old_text, new_text)
```

## Arguments

text_obj	An Automerge text object ID
old_text	The previous text content (single string)
new_text	The new text content (single string)

**Details**

Positions use Unicode code points (matching R's `nchar()` behavior), not bytes. This means multi-byte characters like emoji count as single characters.

**Value**

Invisible NULL (called for side effect)

**Examples**

```
doc <- am_create()
am_put(doc, AM_ROOT, "content", am_text("Hello"))
text_obj <- am_get(doc, AM_ROOT, "content")

# Efficiently update text by computing and applying diff in one step
am_text_update(text_obj, "Hello", "Hello World")
am_text_content(text_obj) # "Hello World"

# Works with Unicode
am_text_update(text_obj, "Hello World", "Hello World!")
am_text_content(text_obj) # "Hello World!"
```

---

am\_uint64

*Create an unsigned 64-bit integer value*


---

**Description**

Creates an `am_uint64` object for storing unsigned 64-bit integers in Automerge documents. This preserves type fidelity when syncing with other language bindings (JavaScript BigInt, Python int, etc.).

**Usage**

```
am_uint64(value = 0)
```

**Arguments**

value            Numeric value (default 0). Values beyond  $2^{53}$  may lose precision.

**Value**

An `am_uint64` object

**Examples**

```
doc <- am_create()
am_put(doc, AM_ROOT, "id", am_uint64(12345))
```

---

am\_values *Get all values from a map or list*

---

### Description

Returns all values from an Automerge map or list as an R list.

### Usage

```
am_values(doc, obj)
```

### Arguments

doc            An Automerge document  
obj            An Automerge object ID, or AM\_ROOT for the document root

### Value

R list of values

### Examples

```
doc <- am_create()
am_put(doc, AM_ROOT, "a", 1)
am_put(doc, AM_ROOT, "b", 2)
am_put(doc, AM_ROOT, "c", 3)

values <- am_values(doc, AM_ROOT)
values # list(1, 2, 3)
```

---

as.character.am\_text *Convert text object to character string*

---

### Description

Extracts the full text content from an Automerge text object as a standard character string.

### Usage

```
## S3 method for class 'am_text'
as.character(x, ...)
```

### Arguments

x            An Automerge text object  
...          Additional arguments (unused)

**Value**

Character string with the full text content

**Examples**

```
doc <- am_create()
am_put(doc, AM_ROOT, "notes", am_text("Hello World"))
text_obj <- am_get(doc, AM_ROOT, "notes")

text_string <- as.character(text_obj)
text_string # "Hello World"

identical(as.character(text_obj), am_text_content(text_obj)) # TRUE
```

---

as.list.am\_doc

*Convert document root to R list*

---

**Description**

Recursively converts the root of an Automerge document to a standard R list. Maps become named lists, lists become unnamed lists, and nested objects are recursively converted.

**Usage**

```
## S3 method for class 'am_doc'
as.list(x, ...)
```

**Arguments**

x	An Automerge document
...	Additional arguments (unused)

**Value**

Named list with document contents

**Examples**

```
doc <- am_create()
doc$name <- "Alice"
doc$age <- 30L

as.list(doc) # list(name = "Alice", age = 30L)
```

---

as_automerge	<i>Convert R list to Automerge document</i>
--------------	---

---

## Description

Converts an R list to an Automerge document. This leverages the recursive conversion built into `am_put()` from Phase 3, allowing nested structures to be created in a single call.

## Usage

```
as_automerge(x, doc = NULL, actor_id = NULL)
```

## Arguments

<code>x</code>	R list, vector, or scalar value to convert
<code>doc</code>	Optional existing Automerge document. If NULL, creates a new one.
<code>actor_id</code>	Optional actor ID for new documents (raw bytes or hex string)

## Value

An Automerge document

## Examples

```
# Convert nested list to Automerge
data <- list(
  name = "Alice",
  age = 30L,
  scores = list(85, 90, 95),
  metadata = list(
    created = Sys.time(),
    tags = list("user", "active")
  )
)

doc <- as_automerge(data)
doc[["name"]] # "Alice"
doc[["age"]]  # 30L
```



---

automerger-constants     *Automerger Constants*

---

## Description

Constants used throughout the automerger package for object types, root references, and mark expansion modes.

## Usage

AM\_ROOT

AM\_OBJ\_TYPE\_LIST

AM\_OBJ\_TYPE\_MAP

AM\_OBJ\_TYPE\_TEXT

AM\_MARK\_EXPAND\_NONE

AM\_MARK\_EXPAND\_BEFORE

AM\_MARK\_EXPAND\_AFTER

AM\_MARK\_EXPAND\_BOTH

## Format

An object of class NULL of length 0.

An object of class am\_obj\_type of length 1.

An object of class am\_obj\_type of length 1.

An object of class am\_obj\_type of length 1.

An object of class character of length 1.

An object of class character of length 1.

An object of class character of length 1.

An object of class character of length 1.

## Root Object

**AM\_ROOT** Reference to the root object of an Automerger document. Use this as the obj parameter when operating on the top-level map. Value is NULL which maps to the C API's AM\_ROOT.

## Object Types

String constants for creating Automerger objects:

**AM\_OBJ\_TYPE\_LIST** Create a list (array) object. Lists are ordered sequences accessed by numeric index (1-based in R).

**AM\_OBJ\_TYPE\_MAP** Create a map (object) object. Maps are unordered key-value collections accessed by string keys.

**AM\_OBJ\_TYPE\_TEXT** Create a text object for collaborative editing. Text objects support character-level CRDT operations, cursor stability, and formatting marks. Use text objects for collaborative document editing rather than regular strings (which use last-write-wins semantics).

## Mark Expansion Modes

Constants for controlling how text marks expand when text is inserted at their boundaries (used with `am_mark`):

**AM\_MARK\_EXPAND\_NONE** Mark does not expand when text is inserted at either boundary.

**AM\_MARK\_EXPAND\_BEFORE** Mark expands to include text inserted immediately before its start position.

**AM\_MARK\_EXPAND\_AFTER** Mark expands to include text inserted immediately after its end position.

**AM\_MARK\_EXPAND\_BOTH** Mark expands to include text inserted at either boundary (before start or after end).

---

extract-am\_doc

*Extract from Automerger document root*

---

## Description

Extract values from the root of an Automerger document using `[]` or `$`. These operators provide R-idiomatic access to document data.

## Usage

```
## S3 method for class 'am_doc'
x[[i]]
```

```
## S3 method for class 'am_doc'
x$name
```

## Arguments

<code>x</code>	An Automerger document
<code>i</code>	Key name (character)
<code>name</code>	Key name (for <code>\$</code> operator)

**Value**

The value at the specified key

**Examples**

```
doc <- am_create()
am_put(doc, AM_ROOT, "name", "Alice")
am_put(doc, AM_ROOT, "age", 30L)

doc[["name"]] # "Alice"
doc$age      # 30L
```

---

extract-am_object	<i>Extract from Automerge object</i>
-------------------	--------------------------------------

---

**Description**

Extract values from an Automerge object (map or list) using `[[` or `$`.

**Usage**

```
## S3 method for class 'am_object'
x[[i]]

## S3 method for class 'am_object'
x$name
```

**Arguments**

<code>x</code>	An Automerge object
<code>i</code>	Key name (character) for maps, or position (integer) for lists
<code>name</code>	Key name (for <code>\$</code> operator, maps only)

**Value**

The value at the specified key/position

**Examples**

```
doc <- am_create()
am_put(doc, AM_ROOT, "user", list(name = "Bob", age = 25L))
user <- am_get(doc, AM_ROOT, "user")

user[["name"]] # "Bob"
user$age      # 25L
```

---

from_automerge	<i>Convert Automerge document to R list</i>
----------------	---

---

**Description**

Converts an Automerge document to a standard R list. This is equivalent to `as.list.am_doc()`.

**Usage**

```
from_automerge(doc)
```

**Arguments**

doc	An Automerge document
-----	-----------------------

**Value**

Named list with document contents

**Examples**

```
doc <- am_create()
doc$name <- "Alice"
doc$age <- 30L

from_automerge(doc) # list(name = "Alice", age = 30L)
```

---

length.am_doc	<i>Get length of document root</i>
---------------	------------------------------------

---

**Description**

Returns the number of keys in the root map of an Automerge document.

**Usage**

```
## S3 method for class 'am_doc'
length(x)
```

**Arguments**

x	An Automerge document
---	-----------------------

**Value**

Integer length

**Examples**

```
doc <- am_create()
doc$a <- 1
doc$b <- 2
length(doc) # 2
```

---

length.am_object	<i>Get length of Automerge object</i>
------------------	---------------------------------------

---

**Description**

Returns the number of elements/keys in an Automerge object.

**Usage**

```
## S3 method for class 'am_object'
length(x)
```

**Arguments**

x                    An Automerge object

**Value**

Integer length

---

names.am_doc	<i>Get names from document root</i>
--------------	-------------------------------------

---

**Description**

Returns the keys from the root map of an Automerge document.

**Usage**

```
## S3 method for class 'am_doc'
names(x)
```

**Arguments**

x                    An Automerge document

**Value**

Character vector of key names

**Examples**

```
doc <- am_create()
doc$name <- "Alice"
doc$age <- 30L
names(doc) # c("name", "age")
```

---

names.am_map	<i>Get names from Automerge map object</i>
--------------	--

---

**Description**

Returns the keys from a map object.

**Usage**

```
## S3 method for class 'am_map'
names(x)
```

**Arguments**

x                    An Automerge map object

**Value**

Character vector of key names

---

replace-am_doc	<i>Replace in Automerge document root</i>
----------------	---

---

**Description**

Replace or insert values at the root of an Automerge document using `[[<-` or `$<-`. These operators provide R-idiomatic modification.

**Usage**

```
## S3 replacement method for class 'am_doc'
x[[i]] <- value

## S3 replacement method for class 'am_doc'
x$name <- value
```

**Arguments**

x	An Automerge document
i	Key name (character)
value	Value to store
name	Key name (for \$<- operator)

**Value**

The document (invisibly)

**Examples**

```
doc <- am_create()
doc[["name"]] <- "Bob"
doc$age <- 25L
```

---

replace-am\_object      *Replace in Automerge object*

---

**Description**

Replace or insert values in an Automerge object using [[<- or \$<-.

**Usage**

```
## S3 replacement method for class 'am_object'
x[[i]] <- value

## S3 replacement method for class 'am_object'
x$name <- value
```

**Arguments**

x	An Automerge object
i	Key name (character) for maps, or position (integer) for lists
value	Value to store
name	Key name (for \$<- operator, maps only)

**Value**

The object (invisibly)

**Examples**

```
doc <- am_create()
am_put(doc, AM_ROOT, "user", list(name = "Bob", age = 25L))
user <- am_get(doc, AM_ROOT, "user")

user[["name"]] <- "Alice"
user$age <- 30L
```



# Index

## \* datasets

- automerge-constants, 41
- [[.am\_doc (extract-am\_doc), 42
- [[.am\_object (extract-am\_object), 43
- [[<-.am\_doc (replace-am\_doc), 46
- [[<-.am\_object (replace-am\_object), 47
- \$.am\_doc (extract-am\_doc), 42
- \$.am\_object (extract-am\_object), 43
- \$<-.am\_doc (replace-am\_doc), 46
- \$<-.am\_object (replace-am\_object), 47

- am\_apply\_changes, 3
- am\_commit, 4
- am\_counter, 5
- am\_counter\_increment, 5
- am\_create, 6
- am\_cursor, 7
- am\_cursor(), 9
- am\_cursor\_position, 8
- am\_cursor\_position(), 8
- am\_delete, 9
- am\_delete\_path, 10
- am\_fork, 10
- am\_get, 11
- am\_get\_actor, 12
- am\_get\_actor(), 12
- am\_get\_actor\_hex, 12
- am\_get\_actor\_hex(), 12
- am\_get\_change\_by\_hash, 15
- am\_get\_changes, 13
- am\_get\_changes\_added, 14
- am\_get\_heads, 15
- am\_get\_history, 16
- am\_get\_last\_local\_change, 17
- am\_get\_path, 18
- am\_insert, 19
- am\_keys, 19
- am\_length, 20
- am\_list, 21
- am\_load, 21

- am\_map, 22
- am\_mark, 23
- am\_mark(), 25
- AM\_MARK\_EXPAND\_AFTER, 23
- AM\_MARK\_EXPAND\_AFTER
  - (automerge-constants), 41
- AM\_MARK\_EXPAND\_BEFORE, 23
- AM\_MARK\_EXPAND\_BEFORE
  - (automerge-constants), 41
- AM\_MARK\_EXPAND\_BOTH, 23
- AM\_MARK\_EXPAND\_BOTH
  - (automerge-constants), 41
- AM\_MARK\_EXPAND\_NONE, 23
- AM\_MARK\_EXPAND\_NONE
  - (automerge-constants), 41
- am\_marks, 24
- am\_marks\_at, 25
- am\_merge, 26
- AM\_OBJ\_TYPE\_LIST (automerge-constants), 41
- AM\_OBJ\_TYPE\_MAP (automerge-constants), 41
- AM\_OBJ\_TYPE\_TEXT (automerge-constants), 41
- am\_put, 27
- am\_put\_path, 28
- am\_rollback, 28
- AM\_ROOT (automerge-constants), 41
- am\_save, 29
- am\_set\_actor, 30
- am\_sync, 30
- am\_sync(), 26
- am\_sync\_decode, 31
- am\_sync\_encode, 32
- am\_sync\_state\_new, 33
- am\_text, 34
- am\_text\_content, 34
- am\_text\_splice, 35
- am\_text\_update, 36

`am_uint64`, [37](#)  
`am_values`, [38](#)  
`as.character.am_text`, [38](#)  
`as.list.am_doc`, [39](#)  
`as_automerge`, [40](#)  
`automerge-constants`, [41](#)

`extract-am_doc`, [42](#)  
`extract-am_object`, [43](#)

`from_automerge`, [44](#)

`length.am_doc`, [44](#)  
`length.am_object`, [45](#)

`names.am_doc`, [45](#)  
`names.am_map`, [46](#)

`replace-am_doc`, [46](#)  
`replace-am_object`, [47](#)