Package ‘automerge’

February 17, 2026

Type Package

Title R Bindings for 'Automerge' 'CRDT' Library

Version 0.3.0

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>. 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 >= 1.85 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 2026-02-17 06:11:11 UTC

Contents

am_apply_changes *Apply changes to a document*

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

doc	An Automerge document
changes	A list of am_change objects (from `am_get_changes()`, `am_get_changes()`, or `am_change_from_bytes()`)

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_close(doc1)
am_close(doc2)
```

am_change_actor_id *Get the actor ID of a change*

Description

Returns the actor ID of the peer that created the change.

Usage

```
am_change_actor_id(change)
```

Arguments

change	An am_change object (from `am_get_changes()`, `am_get_changes()`, or `am_change_from_bytes()`)

Value

A raw vector containing the actor ID bytes

Examples

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

history <- am_get_changes(doc)
actor <- am_change_actor_id(history[[1]])
actor

# Should match the document's actor
identical(actor, am_get_actor(doc))  # TRUE

am_close(doc)
```

am_change_deps *Get the dependencies of a change*

Description

Returns the hashes of the changes that this change depends on (i.e., its parent changes in the causal graph). The first change in a document has no dependencies.

Usage

```
am_change_deps(change)
```

Arguments

`change`	An am_change object (from `am_get_changes()`, `am_get_changes()`, or `am_change_from_bytes()`)

Value

A list of raw vectors (change hashes), each 32 bytes. Returns an empty list for the first change in a document.

Examples

```
doc <- am_create()
am_put(doc, AM_ROOT, "x", 1)
am_commit(doc, "First")
am_put(doc, AM_ROOT, "y", 2)
am_commit(doc, "Second")

history <- am_get_changes(doc)
deps1 <- am_change_deps(history[[1]])
length(deps1)  # 0 (first change has no deps)

deps2 <- am_change_deps(history[[2]])
length(deps2)  # 1 (depends on first change)

am_close(doc)
```

`am_change_from_bytes` *Parse a serialized change from raw bytes*

Description

Deserializes a change from raw bytes into an am_change object. This is useful for restoring changes that were previously serialized with `am_change_to_bytes()` or saved to disk.

Usage

```
am_change_from_bytes(bytes)
```

Arguments

`bytes`	A raw vector containing a serialized change (from `am_change_to_bytes()`)

Details

Note: `am_get_changes()`, `am_get_changes()`, and other change-returning functions already return am_change objects directly, so this function is only needed when working with raw byte representations.

Value

An am_change object (external pointer) that can be passed to am_change_hash(), am_change_message(), am_change_time(), am_change_actor_id(), am_change_seq(), and am_change_deps().

Examples

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

# Serialize a change and restore it
history <- am_get_changes(doc)
bytes <- am_change_to_bytes(history[[1]])
change <- am_change_from_bytes(bytes)
change
am_change_message(change)  # "Add key"

am_close(doc)
```

am_change_hash   *Get the hash of a change*

Description

Returns the unique hash identifier of a change. Change hashes are used to reference specific points in document history (e.g., with am_get_change_by_hash() or am_fork()).

Usage

```
am_change_hash(change)
```

Arguments

change	An am_change object (from am_get_changes(), am_get_changes(), or am_change_from_bytes())

Value

A raw vector (32 bytes) containing the change hash

Examples

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

history <- am_get_changes(doc)
hash <- am_change_hash(history[[1]])
hash
```

```
length(hash)  # 32 bytes

am_close(doc)
```

---

am_change_message — *Get the commit message of a change*

Description

Returns the commit message attached to a change, or NULL if no message was provided when the change was committed.

Usage

```
am_change_message(change)
```

Arguments

change	An am_change object (from am_get_changes(), am_get_changes(), or am_change_from_bytes())

Value

A character string containing the commit message, or NULL

Examples

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

history <- am_get_changes(doc)
am_change_message(history[[1]])  # "Add key"

am_close(doc)
```

---

am_change_seq — *Get the sequence number of a change*

Description

Returns the sequence number of the change within its actor's history. Sequence numbers start at 1 and increment with each change by the same actor.

Usage

```
am_change_seq(change)
```

Arguments

change	An am_change object (from `am_get_changes()`, `am_get_changes()`, or `am_change_from_bytes()`)

Value

A numeric value (double, since sequence numbers can exceed R's 32-bit integer range)

Examples

```
doc <- am_create()
am_put(doc, AM_ROOT, "x", 1)
am_commit(doc, "First")
am_put(doc, AM_ROOT, "y", 2)
am_commit(doc, "Second")

history <- am_get_changes(doc)
am_change_seq(history[[1]])  # 1
am_change_seq(history[[2]])  # 2

am_close(doc)
```

am_change_size *Get the number of operations in a change*

Description

Returns the number of operations contained in the change. Useful for estimating the size of changes before syncing or storing them.

Usage

```
am_change_size(change)
```

Arguments

change	An am_change object (from `am_get_changes()`, `am_get_changes()`, or `am_change_from_bytes()`)

Value

An integer (or double for very large values exceeding R's 32-bit integer range)

Examples

```
doc <- am_create()
am_put(doc, AM_ROOT, "x", 1)
am_put(doc, AM_ROOT, "y", 2)
am_commit(doc, "Add keys")

history <- am_get_changes(doc)
am_change_size(history[[1]])  # 2

am_close(doc)
```

`am_change_time` *Get the timestamp of a change*

Description

Returns the timestamp recorded when the change was committed. Note that timestamps are set by the committing peer and may not be accurate if the peer's clock is wrong.

Usage

```
am_change_time(change)
```

Arguments

`change`	An am_change object (from `am_get_changes()`, `am_get_changes()`, or `am_change_from_bytes()`)

Value

A `POSIXct` timestamp

Examples

```
doc <- am_create()
am_put(doc, AM_ROOT, "key", "value")
am_commit(doc, "Add key", Sys.time())

history <- am_get_changes(doc)
am_change_time(history[[1]])

am_close(doc)
```

`am_change_to_bytes` *Serialize a change to raw bytes*

Description

Converts an `am_change` object back to its serialized raw vector form.

Usage

```
am_change_to_bytes(change)
```

Arguments

`change`	An `am_change` object (from `am_get_changes()`, `am_get_changes()`, or `am_change_from_bytes()`)

Value

A raw vector containing the serialized change

Examples

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

history <- am_get_changes(doc)
bytes <- am_change_to_bytes(history[[1]])
bytes

# Round-trip
restored <- am_change_from_bytes(bytes)
identical(am_change_to_bytes(restored), bytes)  # TRUE

am_close(doc)
```

`am_clone` *Clone an Automerge document*

Description

Creates an independent deep copy of an Automerge document, preserving the same actor ID. Changes to the clone do not affect the original, and vice versa.

Usage

```
am_clone(doc)
```

Arguments

doc	An Automerge document

Details

Unlike am_fork(), which assigns a new actor ID to the copy, am_clone() preserves the original actor ID. This makes it suitable for archival snapshots or checkpoints where you want an exact copy of the document state. Do not use am_clone() to create branches that will make independent edits and later be merged — use am_fork() for that, as two documents sharing an actor ID can cause conflicts.

Value

A new Automerge document (independent copy with same actor ID)

See Also

am_fork() for creating branches with a new actor ID

Examples

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

clone <- am_clone(doc)
clone$key  # "value"

# Clone preserves the actor ID
am_get_actor_hex(doc) == am_get_actor_hex(clone) # TRUE

# Changes to clone don't affect original
clone$key <- "changed"
doc$key  # still "value"

am_close(doc)
am_close(clone)
```

---

am_close *Close an Automerge document*

Description

Explicitly frees the resources associated with an Automerge document. After calling this function, the document becomes invalid and should not be used.

Usage

```
am_close(doc)
```

Arguments

`doc`	An Automerge document (created with `am_create()` or `am_load()`)

Details

This function is useful when you need deterministic cleanup rather than waiting for garbage collection. It is safe to call on a document that has already been closed.

Value

`NULL` (invisibly)

Examples

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

# Explicitly free resources
am_close(doc)

# Document is now invalid - do not use after closing
```

`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_close(doc)
```

`am_commit_empty` *Create an empty change*

Description

Creates a new change in the document's history without any operations. This is useful for creating merge commits or recording metadata (message, timestamp) without making data changes.

Usage

```
am_commit_empty(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()
doc$key <- "value"
am_commit(doc, "Initial data")

# Create empty change as a checkpoint
am_commit_empty(doc, "Checkpoint")

am_close(doc)
```

`am_counter` *Create an Automerge counter*

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_close(doc)
```

`am_counter_increment` *Increment a counter value*

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_close(doc)
```

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", "automerge")`.

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
doc1 <- am_create()
doc1

# 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_close(doc1)
am_close(doc2)
am_close(doc3)
```

`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, heads = NULL)
```

Arguments

obj	An Automerge object ID (must be a text object)
position	Integer position in the text (0-based inter-character position)
heads	Optional list of change hashes (raw vectors) to create the cursor at a historical document state. If NULL (default), uses the current state.

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)
cursor

# 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_close(doc)
```

`am_cursor_equal` *Test equality of two cursors*

Description

Compares two cursors to determine if they refer to the same position in a document. This compares the internal cursor representation, not just the current position.

Usage

```
am_cursor_equal(cursor1, cursor2)
```

Arguments

`cursor1`	An `am_cursor` object
`cursor2`	An `am_cursor` object

Value

A logical scalar: `TRUE` if the cursors are equal, `FALSE` otherwise

Examples

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

cursor1 <- am_cursor(text_obj, 5)
cursor2 <- am_cursor(text_obj, 5)
cursor3 <- am_cursor(text_obj, 3)

am_cursor_equal(cursor1, cursor2)  # TRUE
am_cursor_equal(cursor1, cursor3)  # FALSE

am_close(doc)
```

`am_cursor_from_bytes` *Restore a cursor from bytes*

Description

Restores a cursor from a raw vector previously created by `am_cursor_to_bytes()`. The text object is required to associate the cursor with a document.

Usage

```
am_cursor_from_bytes(bytes, obj)
```

Arguments

`bytes`	A raw vector containing a serialized cursor
`obj`	An Automerge text object to associate the cursor with

Value

An `am_cursor` object

See Also

`am_cursor_to_bytes()`

Examples

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

cursor <- am_cursor(text_obj, 5)
bytes <- am_cursor_to_bytes(cursor)

restored <- am_cursor_from_bytes(bytes, text_obj)
restored
am_cursor_position(restored)  # 5

am_close(doc)
```

`am_cursor_from_string` *Restore a cursor from a string*

Description

Restores a cursor from a string previously created by `am_cursor_to_string()`. The text object is required to associate the cursor with a document.

Usage

```
am_cursor_from_string(str, obj)
```

Arguments

`str`	A character string containing a serialized cursor
`obj`	An Automerge text object to associate the cursor with

Value

An am_cursor object

See Also

am_cursor_to_string()

Examples

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

cursor <- am_cursor(text_obj, 5)
str <- am_cursor_to_string(cursor)

restored <- am_cursor_from_string(str, text_obj)
restored
am_cursor_position(restored)  # 5

am_close(doc)
```

---

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, heads = NULL)
```

Arguments

cursor	An am_cursor object created by am_cursor()
heads	Optional list of change hashes (raw vectors) to query cursor position at a historical document state. If NULL (default), uses the current state.

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_close(doc)
```

`am_cursor_to_bytes` *Serialize a cursor to bytes*

Description

Converts a cursor to a raw vector representation that can be persisted and later restored with `am_cursor_from_bytes()`. This enables saving cursor positions across R sessions.

Usage

```
am_cursor_to_bytes(cursor)
```

Arguments

`cursor`	An `am_cursor` object created by `am_cursor()`

Value

A raw vector containing the serialized cursor

See Also

`am_cursor_from_bytes()`, `am_cursor_to_string()`

Examples

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

cursor <- am_cursor(text_obj, 5)
bytes <- am_cursor_to_bytes(cursor)
bytes
```

```
# Restore cursor later
restored <- am_cursor_from_bytes(bytes, text_obj)
am_cursor_position(restored)  # 5

am_close(doc)
```

`am_cursor_to_string`   *Serialize a cursor to a string*

Description

Converts a cursor to a string representation that can be persisted and later restored with `am_cursor_from_string()`.

Usage

```
am_cursor_to_string(cursor)
```

Arguments

`cursor`	An `am_cursor` object created by `am_cursor()`

Value

A character string containing the serialized cursor

See Also

`am_cursor_from_string()`, `am_cursor_to_bytes()`

Examples

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

cursor <- am_cursor(text_obj, 5)
str <- am_cursor_to_string(cursor)
str

# Restore cursor later
restored <- am_cursor_from_string(str, text_obj)
am_cursor_position(restored)  # 5

am_close(doc)
```

`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_close(doc)
```

`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_close(doc)
```

---

am_equal *Test document equality*

Description

Tests whether two Automerge documents have the same content. Documents are equal if they have the same set of changes applied, regardless of how they were created.

Usage

```
am_equal(doc1, doc2)
```

Arguments

doc1	An Automerge document
doc2	An Automerge document

Value

A logical scalar: `TRUE` if the documents are equal, `FALSE` otherwise.

Examples

```
doc1 <- am_create()
doc1$key <- "value"
am_commit(doc1)

doc2 <- am_clone(doc1)
am_equal(doc1, doc2)  # TRUE

doc2$key <- "different"
```

```
am_equal(doc1, doc2)  # FALSE

am_close(doc1)
am_close(doc2)
```

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. The fork is assigned a new actor ID, so changes made on the fork are distinguishable from the original when merged or synced.

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).

Details

Use `am_fork()` when creating independent branches of a document that may later be merged or synced. Use `am_clone()` instead if you need an exact copy that preserves the original actor ID (e.g. for archival or snapshotting purposes).

Value

A new Automerge document (fork of the original)

See Also

`am_clone()` for an exact copy preserving the actor ID

Examples

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

# Fork has a different actor ID
am_get_actor_hex(doc1) != am_get_actor_hex(doc2) # TRUE

# Now doc1 and doc2 can diverge independently
am_close(doc1)
am_close(doc2)
```

`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 <= `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_close(doc)
```

`am_get_actor` *Get the actor ID of a document*

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)
actor

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

am_close(doc)
```

`am_get_actor_hex` *Get the actor ID as a hex string*

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)
actor_hex

am_close(doc)
```

`am_get_changes` *Get changes since specified heads*

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 `am_change` objects that can be inspected with `am_change_hash()`, `am_change_message()`, etc., serialized with `am_change_to_bytes()`, or applied to other documents using `am_apply_changes()`.

Value

A list of `am_change` objects.

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_close(doc)
```

---

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 am_change objects representing changes that exist 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)
changes
```

```
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_close(doc1)
am_close(doc2)
```

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_change_hash()`.

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

An `am_change` object, 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)
change
am_change_message(change)  # "Add key"

am_close(doc)
```

`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_close(doc)
```

`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

An `am_change` object, 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)
change
am_change_message(change)  # "Add key"

am_close(doc)
```

`am_get_missing_deps` *Get missing dependencies*

Description

Returns the change hashes of dependencies that are referenced by the document but not present in its change history. This can happen when changes are applied out of order or when a document is partially synced.

Usage

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

Arguments

`doc`	An Automerge document
`heads`	Optional list of change hashes (raw vectors) to check for missing dependencies. If `NULL` (default), checks the current heads.

Value

A list of raw vectors (change hashes of missing dependencies). Returns an empty list if no dependencies are missing.

Examples

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

# Complete document has no missing deps
missing <- am_get_missing_deps(doc)
length(missing)  # 0

am_close(doc)
```

---

am_get_path — *Navigate deep structures with path*

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_close(doc)
```

am_insert *Insert a value into an Automerge list*

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")
items

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

am_close(doc)
```

am_items *Get full item details from an object*

Description

Returns detailed information about each entry in a map or list, including the key (or index) and value for each item. This provides more information than `am_values()` alone.

Usage

```
am_items(doc, obj, heads = NULL)
```

Arguments

`doc`	An Automerge document
`obj`	An Automerge object ID, or `AM_ROOT` for the document root
`heads`	Optional list of change hashes (raw vectors) for historical query. If `NULL` (default), uses the current state.

Value

A list of lists, where each inner list has fields:

key For maps: the character key. For lists: the 1-based integer index.

value The value at this entry.

Note

When called on a text object, this iterates individual characters as list items. Use `am_text_content()` to retrieve text as a string instead.

Examples

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

items <- am_items(doc, AM_ROOT)
items[[1]]$key    # "age" (sorted lexicographically)
items[[1]]$value  # 30

am_close(doc)
```

am_keys *Get all keys from an Automerge map*

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_close(doc)
```

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_close(doc)
```

`am_list` *Create an Automerge list*

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_list_get_all` *Get all conflicting values at a list position*

Description

Returns all values stored at a list position, including conflicts from concurrent edits. When there are no conflicts, the list contains a single element (the winning value).

Usage

```
am_list_get_all(doc, obj, pos, heads = NULL)
```

Arguments

`doc`	An Automerge document
`obj`	An Automerge object ID (must be a list)
`pos`	Numeric index (1-based, like R vectors)
`heads`	Optional list of change hashes (raw vectors) for historical query. If `NULL` (default), uses the current state.

Value

A list of all values at the position. Returns an empty list if the position does not exist.

Examples

```
doc <- am_create()
doc$items <- list("a", "b", "c")
items <- doc$items

# Single value (no conflict)
values <- am_list_get_all(doc, items, 1)
length(values)  # 1

am_close(doc)
```

`am_list_range` *Get a range of list items*

Description

Returns list elements within the index range `[begin, end]`. Uses 1-based indexing consistent with R conventions.

Usage

```
am_list_range(doc, obj, begin, end, heads = NULL)
```

Arguments

`doc`	An Automerge document
`obj`	An Automerge object ID (must be a list)
`begin`	Start index (1-based, inclusive)
`end`	End index (1-based, inclusive)
`heads`	Optional list of change hashes (raw vectors) for historical query. If `NULL` (default), uses the current state.

Value

A list of values in the index range.

Examples

```
doc <- am_create()
doc$items <- list("a", "b", "c", "d", "e")
items <- doc$items

# Get elements 2 through 4 -> "b", "c", "d"
range <- am_list_range(doc, items, 2, 4)
length(range)  # 3

am_close(doc)
```

`am_load` *Load an Automerge document from binary format*

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)
doc2

# Save to and load from file
doc3 <- am_create()
file <- tempfile()
writeBin(am_save(doc3), file)

doc4 <- am_load(readBin(file, "raw", 1e5))

unlink(file)
am_close(doc1)
am_close(doc2)
am_close(doc3)
am_close(doc4)
```

---

am_load_changes *Load a document as individual changes*

Description

Decomposes a serialized document into its individual changes. This is useful for inspecting the full change history or for selectively applying changes to another document.

Usage

```
am_load_changes(data)
```

Arguments

data	A raw vector containing a serialized Automerge document (from am_save())

Value

A list of am_change objects. Returns an empty list for an empty document.

Examples

```
doc <- am_create()
doc$key <- "value"
am_commit(doc, "Add key")
doc$key2 <- "value2"
am_commit(doc, "Add key2")
bytes <- am_save(doc)
```

```
# Load as individual changes
changes <- am_load_changes(bytes)
length(changes)  # 2
am_change_message(changes[[1]])  # "Add key"
am_change_message(changes[[2]])  # "Add key2"

# Apply to a new document
doc2 <- am_create()
am_apply_changes(doc2, changes)
doc2$key   # "value"
doc2$key2  # "value2"

am_close(doc)
am_close(doc2)
```

`am_load_incremental` *Load incremental changes into a document*

Description

Applies incremental changes (from `am_save_incremental()`) to a document. This is more efficient than loading a full document when only a few changes need to be applied.

Usage

```
am_load_incremental(doc, data)
```

Arguments

`doc`	An Automerge document
`data`	A raw vector containing incremental changes (from `am_save_incremental()`)

Value

The number of operations applied (numeric scalar, invisibly).

See Also

`am_save_incremental()`, `am_load()`

Examples

```
doc1 <- am_create()
doc1$key <- "value"
am_commit(doc1)
bytes <- am_save(doc1)
```

```
doc1$key2 <- "value2"
am_commit(doc1)
incremental <- am_save_incremental(doc1)

# Load base document and apply incremental changes
doc2 <- am_load(bytes)
am_load_incremental(doc2, incremental)
doc2$key2  # "value2"

am_close(doc1)
am_close(doc2)
```

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_map_get_all` *Get all conflicting values at a map key*

Description

Returns all values stored at a map key, including conflicts from concurrent edits by different actors. When there are no conflicts, the list contains a single element (the winning value).

Usage

```
am_map_get_all(doc, obj, key, heads = NULL)
```

Arguments

`doc`	An Automerge document
`obj`	An Automerge object ID (must be a map), or `AM_ROOT`
`key`	Character string key
`heads`	Optional list of change hashes (raw vectors) for historical query. If `NULL` (default), uses the current state.

Value

A list of all values at the key. Returns an empty list if the key does not exist.

Examples

```
doc <- am_create()
doc$key <- "value"

# Single value (no conflict)
values <- am_map_get_all(doc, AM_ROOT, "key")
length(values)  # 1

am_close(doc)
```

`am_map_range` *Get a range of map items by key*

Description

Returns map entries whose keys fall within the lexicographic range `[begin, end]` (inclusive on both sides).

Usage

```
am_map_range(doc, obj, begin = "", end = "", heads = NULL)
```

Arguments

doc	An Automerge document
obj	An Automerge object ID (must be a map), or AM_ROOT
begin	Start key (inclusive). Use "" (default) for unbounded start.
end	End key (inclusive). Use "" (default) for unbounded end.
heads	Optional list of change hashes (raw vectors) for historical query. If NULL (default), uses the current state.

Value

A named list of values in the key range.

Examples

```
doc <- am_create()
doc$a <- 1
doc$b <- 2
doc$c <- 3
doc$d <- 4

# Get entries with keys in [b, c] -> b and c
range <- am_map_range(doc, AM_ROOT, "b", "c")
names(range)  # "b" "c"

# Get all entries
all <- am_map_range(doc, AM_ROOT, "", "")

am_close(doc)
```

---

am_mark   *Create a mark on a text range*

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: **"none"** Mark does not expand (default) **"before"** Mark expands to include text inserted before start **"after"** Mark expands to include text inserted after end **"both"** Mark expands in both directions. Use the constants AM_MARK_EXPAND_NONE, AM_MARK_EXPAND_BEFORE, AM_MARK_EXPAND_AFTER, or AM_MARK_EXPAND_BOTH.

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_close(doc)
```

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, heads = NULL)
```

Arguments

`obj`	An Automerge object ID (must be a text object)
`heads`	Optional list of change hashes (raw vectors) to query marks at a historical document state. If `NULL` (default), uses the current state.

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_close(doc)
```

---

am_marks_at *Get marks at a specific position*

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, heads = NULL)
```

Arguments

`obj`	An Automerge object ID (must be a text object)
`position`	Integer position (0-based inter-character position) to query. See `am_mark()` for indexing details.
`heads`	Optional list of change hashes (raw vectors) to query marks at a historical document state. If `NULL` (default), uses the current state.

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_close(doc)
```

`am_mark_clear` *Clear marks from a text range*

Description

Removes marks matching the given name from a range of text. This is the inverse of `am_mark()`.

Usage

```
am_mark_clear(obj, start, end, name, 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 to clear (e.g., "bold")
`expand`	Character string controlling mark clearing behavior at boundaries. Options: `"none"` (default), `"before"`, `"after"`, `"both"`. Use the constants AM_MARK_EXPAND_NONE, AM_MARK_EXPAND_BEFORE, AM_MARK_EXPAND_AFTER, or AM_MARK_EXPAND_BOTH

Value

The text object `obj` (invisibly)

Indexing Convention

Uses the same 0-based inter-character position indexing as `am_mark()`.

See Also

`am_mark()`, `am_marks()`

Examples

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

# Add a mark
am_mark(text_obj, 0, 11, "bold", TRUE)
length(am_marks(text_obj))  # 1

# Clear the mark
am_mark_clear(text_obj, 0, 11, "bold")
length(am_marks(text_obj))  # 0

am_close(doc)
```

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_close(doc1)
am_close(doc2)
```

`am_pending_ops` *Get the number of pending operations*

Description

Returns the number of operations that have been applied to the document but not yet committed. This is useful for determining whether a commit is needed.

Usage

```
am_pending_ops(doc)
```

Arguments

`doc`	An Automerge document

Value

An integer indicating the number of pending operations. Returns 0 if there are no uncommitted changes.

Examples

```
doc <- am_create()
am_pending_ops(doc)  # 0

doc$key <- "value"
am_pending_ops(doc)  # > 0

am_commit(doc)
am_pending_ops(doc)  # 0

am_close(doc)
```

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

`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) or `"end"` to append
`value`	The value to store. Supported types: • `NULL` - stores null • Logical - stores boolean (must be scalar) • Integer - stores integer (must be scalar) • Numeric - stores double (must be scalar) • Character - stores string (must be scalar) • Raw - stores bytes • `AM_OBJ_TYPE_LIST/MAP/TEXT` - creates nested object

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")
items

am_close(doc)
```

am_put_path *Set value at path*

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_close(doc)
```

`am_rollback` *Roll back pending operations*

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_close(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)
bytes

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

unlink(file)
am_close(doc)
```

---

am_save_incremental *Save incremental changes*

Description

Serializes only the changes made since the last call to `am_save()` or `am_save_incremental()`. This is more efficient than saving the entire document when only a few changes have been made.

Usage

```
am_save_incremental(doc)
```

Arguments

doc	An Automerge document

Details

Use `am_load_incremental()` to apply these changes to another document.

Value

A raw vector containing the incremental changes. May be empty (zero-length) if no new changes have been made since the last save.

See Also

`am_load_incremental()`, `am_save()`

Examples

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

# Save full document
full <- am_save(doc)

# Make more changes
doc$key2 <- "value2"
am_commit(doc)

# Save only the new changes
incremental <- am_save_incremental(doc)
length(incremental) < length(full)  # TRUE (smaller)

am_close(doc)
```

`am_set_actor` *Set the actor ID of a document*

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: • `NULL` - Generate new random actor ID • Character string - Hex-encoded actor ID • Raw vector - Binary actor ID bytes

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_close(doc)
```

---

am_sync *Bidirectional synchronization*

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_close(doc1)
am_close(doc2)
```

`am_sync_decode` *Receive and apply a sync message*

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()`)
`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()

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

am_close(doc)
```

am_sync_encode *Generate a sync message*

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()`)

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()

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

am_close(doc)
```

`am_sync_state` *Create a new sync state*

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()
```

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()
sync1
sync2 <- am_sync_state()

# Use with am_sync_encode() and am_sync_decode()

am_close(doc1)
am_close(doc2)
```

`am_sync_state_decode` *Deserialize a sync state*

Description

Restores a sync state from a raw vector previously created by `am_sync_state_encode()`. This allows continuing a sync session from where it left off.

Usage

```
am_sync_state_decode(data)
```

Arguments

`data`	A raw vector containing a serialized sync state

Value

An `am_syncstate` object.

See Also

`am_sync_state_encode()`, `am_sync_state()`

Examples

```
sync_state <- am_sync_state()
bytes <- am_sync_state_encode(sync_state)

# Restore sync state
restored <- am_sync_state_decode(bytes)
restored
```

`am_sync_state_encode` *Serialize a sync state*

Description

Encodes a sync state to a raw vector for persistence or transmission. The encoded state can later be restored with `am_sync_state_decode()`.

Usage

```
am_sync_state_encode(sync_state)
```

Arguments

`sync_state`	A sync state object (created with `am_sync_state()`)

Details

This is useful for persisting sync progress across sessions, avoiding the need to re-sync from scratch.

Value

A raw vector containing the serialized sync state.

See Also

am_sync_state_decode(), am_sync_state()

Examples

```
sync_state <- am_sync_state()

# Encode for storage
bytes <- am_sync_state_encode(sync_state)
bytes

# Restore later
restored <- am_sync_state_decode(bytes)
restored
```

---

am_text *Create an Automerge text object*

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` *Get text content from a text object*

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_obj

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

am_close(doc)
```

`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")
text_obj

# 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_close(doc)
```

am_text_update *Update text content*

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")
text_obj

# 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_close(doc)
```

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_close(doc)
```

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)

am_close(doc)
```

`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_obj

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

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

am_close(doc)
```

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)

am_close(doc)
```

as_automerge *Convert R list to Automerge document*

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

`x`	R list, vector, or scalar value to convert
`doc`	Optional existing Automerge document. If NULL, creates a new one.
`actor_id`	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
doc[["name"]]  # "Alice"
doc[["age"]]   # 30L

am_close(doc)
```

`automerge-constants` *Automerge Constants*

Description

Constants used throughout the automerge 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 Automerge 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 Automerge 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 Automerge document root*

Description

Extract values from the root of an Automerge 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

`x`	An Automerge document
`i`	Key name (character)
`name`	Key name (for `$` 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

am_close(doc)
```

extract-am_object *Extract from Automerge object*

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

`x`	An Automerge object
`i`	Key name (character) for maps, or position (integer) for lists
`name`	Key name (for `$` 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

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

am_close(doc)
```

`from_automerge` *Convert Automerge document to R list*

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)

am_close(doc)
```

`length.am_doc` *Get length of document root*

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
am_close(doc)
```

`length.am_object` *Get length of Automerge object*

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` *Get names from document root*

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")
am_close(doc)
```

---

names.am_map *Get names from Automerge map object*

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 *Replace in Automerge document root*

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
am_close(doc)
```

---

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

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

am_close(doc)
```

`str.am_doc` *Display the structure of an Automerge document*

Description

S3 method for `utils::str()` that displays the structure of an Automerge document in a human-readable format.

Usage

```
## S3 method for class 'am_doc'
str(object, max.level = 2, ...)
```

Arguments

`object`	An automerge document object.
`max.level`	Maximum depth to recurse into nested structures. Default 2.
`...`	Additional arguments (ignored).

Value

Invisibly returns `NULL`.

Examples

```
doc <- am_create()
doc$name <- "Alice"
doc$data <- list(x = 1L, y = 2L)
str(doc)
str(doc, max.level = 1)
am_close(doc)
```

Index