Package: autosync (via r-universe)

July 6, 2026

Type Package

Title 'Automerge' Sync Server and Client

Version 0.1.0

Description A WebSocket-based implementation of the 'automerge-repo' synchronization protocol used by 'sync.automerge.org'. Acts as a sync server, enabling 'R' to serve as a synchronization hub for 'Automerge' clients in 'JavaScript', 'Rust', and other languages, and as a client for fetching, editing, and synchronizing documents hosted on remote servers.

License MIT + file LICENSE

URL https://posit-dev.github.io/autosync/, https://github.com/posit-dev/autosync

BugReports https://github.com/posit-dev/autosync/issues

Depends R (>= 4.4)

Imports automerge (>= 0.4.0), httr2 (>= 1.2.3), jose, later, nanonext (>= 1.8.1), promises, secretbase (>= 1.3.0)

Suggests openssl, testthat (>= 3.0.0)

Config/Needs/website tidyverse/tidytemplate

Config/roxygen2/markdown TRUE

Config/roxygen2/version 8.0.0.9000

Config/testthat/edition 3

Encoding UTF-8

NeedsCompilation no

Author Charlie Gao [aut, cre] (ORCID: <https://orcid.org/0000-0002-0750-061X>), Posit Software, PBC [cph, fnd] (ROR: <https://ror.org/03wc8by49>)

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

Repository https://cran.r-universe.dev

Date/Publication 2026-07-06 14:39:24 UTC

RemoteUrl https://github.com/cran/autosync

RemoteRef HEAD

RemoteSha 20450fe0c510cf81452adab9a9d3cf838cdc0eb8

Contents

`autosync-package` *autosync: 'Automerge' Sync Server and Client*

Description

A WebSocket-based implementation of the 'automerge-repo' synchronization protocol used by 'sync.automerge.org'. Acts as a sync server, enabling 'R' to serve as a synchronization hub for 'Automerge' clients in 'JavaScript', 'Rust', and other languages, and as a client for fetching, editing, and synchronizing documents hosted on remote servers.

Main Functions

`sync_server()` Create a new sync server with `$start()` and `$close()` methods

Document Management

`create_document()` Create a new document

`get_document()` Retrieve a document by ID

`list_documents()` List all document IDs

`generate_document_id()` Generate a new document ID

Protocol

The server implements the automerge-repo sync protocol over WebSockets. Messages are CBOR-encoded and include:

join/peer Handshake messages for connection establishment

request/sync Document synchronization messages

ephemeral Transient messages forwarded without persistence

error Error notifications

Example

```
# Create and start a server
server <- sync_server()
server$start()
server$url

# Stop when done
server$close()
```

Author(s)

Maintainer: Charlie Gao <charlie.gao@posit.co> (ORCID)

Authors:

- Charlie Gao <charlie.gao@posit.co> (ORCID)

Other contributors:

- Posit Software, PBC (ROR) [copyright holder, funder]

See Also

Useful links:

- https://posit-dev.github.io/autosync/
- https://github.com/posit-dev/autosync
- Report bugs at https://github.com/posit-dev/autosync/issues

auth_config *Create an authentication configuration*

Description

Creates a configuration object for enabling OIDC JWT authentication on an autosync server. When enabled, clients must include a valid JWT (ID token) as a Bearer token in the Authorization header of the WebSocket upgrade request. Connections without valid credentials are rejected immediately at connection time.

Usage

```
auth_config(
  client_id = Sys.getenv("OIDC_CLIENT_ID"),
  issuer = oidc_issuer(),
  allowed_emails = NULL,
  allowed_domains = NULL,
  custom_validator = NULL
)
```

Arguments

client_id	The OIDC client ID (application ID). Validated against the aud claim in JWTs. Defaults to the OIDC_CLIENT_ID environment variable.
issuer	The OIDC issuer URL. This is used to discover the provider's public keys via the .well-known/openid-configuration endpoint, and to validate the iss claim in JWTs. Defaults to the OIDC_ISSUER environment variable, falling back to Google ("https://accounts.google.com").
allowed_emails	Character vector of allowed email addresses. When set, a token is rejected unless it carries an email claim with email_verified explicitly TRUE.
allowed_domains	Character vector of allowed email domains (e.g., "mycompany.com"). Subject to the same verified-email requirement as allowed_emails.
custom_validator	Function(claims) returning TRUE/FALSE for custom validation logic. Receives the decoded JWT claims as a list.

Details

Works with any OIDC-compliant identity provider: Google, Microsoft Entra, Okta, Auth0, etc.

Value

An object of class "autosync_auth_config".

Examples

```
# Google (default issuer)
auth_config(
  client_id = "123456789.apps.googleusercontent.com",
  allowed_domains = "mycompany.com"
)

# Microsoft Entra
auth_config(
  client_id = "abcdef-1234-5678",
  issuer = "https://login.microsoftonline.com/common/v2.0",
  allowed_emails = "alice@mycompany.com"
)

# Custom validator
auth_config(
  client_id = "0oaXXXXXXXX",
  issuer = "https://dev-123456.okta.com/oauth2/default",
  custom_validator = function(claims) "editors" %in% claims$groups
)
```

---

create_document *Create a new document on the server*

Description

Creates a new empty Automerge document and registers it with the server.

Usage

```
create_document(server, doc_id = NULL)
```

Arguments

server	An autosync_server object.
doc_id	Optional document ID. If NULL, generates a new ID.

Value

Document ID string.

Examples

```
server <- sync_server()
doc_id <- create_document(server)
server$close()
```

generate_document_id *Generate a new document ID*

Description

Creates a new unique document ID compatible with automerge-repo. The ID is a 16-byte random value encoded with Base58Check.

Usage

```
generate_document_id()
```

Value

Character string (Base58Check encoded).

Examples

```
generate_document_id()
```

get_document *Get a document from the server*

Description

Retrieves an Automerge document by its ID.

Usage

```
get_document(server, doc_id)
```

Arguments

server	An autosync_server object.
doc_id	Document ID string.

Value

Automerge document object, or NULL if not found.

Examples

```
server <- sync_server()
doc_id <- create_document(server)
get_document(server, doc_id)
server$close()
```

list_documents *List all document IDs*

Description

Returns the IDs of all documents currently loaded in the server.

Usage

```
list_documents(server)
```

Arguments

server	An autosync_server object.

Value

Character vector of document IDs.

Examples

```
server <- sync_server()
create_document(server)
list_documents(server)
server$close()
```

sync_client *Open a persistent sync connection*

Description

Connects to an automerge-repo sync server and maintains a persistent WebSocket connection. The connection performs the protocol handshake but holds no documents on its own; open one or more live documents over it with the `$open_doc()` method. Each document stays synced — receiving real-time updates from other peers and flushing local changes — for as long as the connection is open. Unlike `sync_fetch()`, which performs a one-off retrieval over a throwaway connection, several documents can share a single connection here.

Usage

```
sync_client(url, timeout = 5000L, tls = NULL, token = NULL, interval = 1000L)
```

Arguments

`url`	WebSocket URL of the sync server (e.g., "ws://localhost:3030/" or "wss://sync.automerge.org/"). Note: trailing slash may be required.
`timeout`	Timeout in milliseconds for each receive operation. Default 5000.
`tls`	(optional) for secure wss:// connections to servers with self-signed or custom CA certificates, a TLS configuration object created by `nanonext::tls_config()`.
`token`	(optional) JWT (ID token) for authenticated servers. Sent as a Bearer token in the Authorization header of the WebSocket upgrade request.
`interval`	Interval in milliseconds for pushing local changes to the server. Default 1000. Uses `later::later()` to periodically check for and send local changes for every open document. This is a cheap no-op when there are no changes.

Details

Opening the connection performs a synchronous handshake before returning. `$open_doc()` then performs a synchronous initial sync, so the returned handle's `$doc` has meaningful content immediately. After that, incoming changes are received asynchronously via a self-chaining promise loop, and local changes are flushed periodically via a `later::later()` timer.

Neither `close()` flushes pending local changes. Call `$push()` first if you have unsynced edits — otherwise any changes made since the last sync-interval tick may be lost.

Value

An environment of class `"autosync_client"` with reference semantics, representing the connection:

`open_doc(doc_id, timeout)` Open a live document over this connection and return a `autosync_doc` handle for it (see below). Repeated calls for the same `doc_id` reuse the document already open on the connection rather than requesting it again.

`close()` Disconnect and stop syncing all open documents.

`active` Logical, whether the connection is active.

A `autosync_doc` handle returned by `$open_doc()` is itself an environment with:

`doc` The live automerge document, kept in sync with the server.

`push()` Push this document's local changes to the server immediately.

`close()` Stop syncing this one document (detach it from the connection); the connection and its other documents are unaffected.

`active` Logical, whether the document is still open on an active connection.

Examples

```
server <- sync_server()
server$start()
doc_id <- create_document(server)

conn <- sync_client(server$url)
```

```
doc <- conn$open_doc(doc_id)
automerge::am_keys(doc$doc)

# Make local changes and push
automerge::am_put(doc$doc, automerge::AM_ROOT, "key", "value")
doc$push()

# Open another document over the same connection
other <- conn$open_doc(create_document(server))

# Disconnect (closes every document on the connection)
conn$close()
server$close()
```

`sync_fetch` *Fetch a document from a sync server*

Description

Connects to an automerge-repo sync server and retrieves a document by ID. This is useful for debugging sync issues or fetching documents from remote servers like sync.automerge.org.

Usage

```
sync_fetch(
  url,
  doc_id,
  timeout = 5000L,
  tls = NULL,
  token = NULL,
  verbose = FALSE
)
```

Arguments

`url`	WebSocket URL of the sync server (e.g., "ws://localhost:3030/" or "wss://sync.automerge.org/"). Note: trailing slash may be required.
`doc_id`	Document ID (base58check encoded string)
`timeout`	Timeout in milliseconds for each receive operation. Default 5000.
`tls`	(optional) for secure wss:// connections to servers with self-signed or custom CA certificates, a TLS configuration object created by `nanonext::tls_config()`.
`token`	(optional) JWT (ID token) for authenticated servers. Sent as a Bearer token in the Authorization header of the WebSocket upgrade request.
`verbose`	Logical, print debug messages. Default FALSE.

Details

The function implements the automerge-repo sync protocol:

1. Connects via WebSocket
2. Sends join message with peer ID
3. Receives peer response from server
4. Sends sync request for the specified document
5. Receives and applies sync messages until complete

Sync is considered complete when no new messages arrive within the timeout after at least one sync round.

Value

An automerge document object containing the fetched data.

Examples

```
# Fetch from public sync server
doc <- sync_fetch("wss://sync.automerge.org", "4F63WJPDzbHkkfKa66h1Qrr1sC5U")

# Fetch from local server with debug output
doc <- sync_fetch(server$url, "myDocId", verbose = TRUE)

# Fetch from server with self-signed certificate
cert <- nanonext::write_cert()
tls <- nanonext::tls_config(client = cert$client)
doc <- sync_fetch(server$url, "myDocId", tls = tls)

# Fetch from authenticated server
doc <- sync_fetch(
  "wss://secure.example.com",
  "myDocId",
  token = "eyJhbGciOi..."
)

# Inspect the document
automerge::am_keys(doc)
```

---

sync_server *Create an Automerge sync server*

Description

Creates a WebSocket server that implements the automerge-repo sync protocol, compatible with JavaScript, Rust, and other Automerge clients.

Usage

```
sync_server(
  port = 0L,
  host = "127.0.0.1",
  data_dir = tempfile("autosync"),
  auto_create_docs = TRUE,
  storage_id = NULL,
  tls = NULL,
  auth = NULL,
  share = NA
)
```

Arguments

port	Port to listen on. Default 0 (binds to a random available port). The actual URL is retrieved via `server$url`.
host	Host address to bind to. Default "127.0.0.1" (localhost).
data_dir	Directory for document storage. Defaults to a session-temporary directory, so documents do not persist across R sessions. Supply an explicit path to persist documents across sessions.
auto_create_docs	Logical, whether to auto-create documents when clients request unknown document IDs. Default TRUE.
storage_id	Optional storage ID for this server. If NULL (default), generates a new persistent identity. Set to NA for an ephemeral server (no persistence identity).
tls	(optional) for secure wss:// connections, a TLS configuration object created by `nanonext::tls_config()`.
auth	Optional authentication configuration created by `auth_config()`. When provided, clients must include a valid JWT (ID token) as a Bearer token in the Authorization header of the WebSocket upgrade request. Connections without valid credentials are rejected immediately. Note: TLS is required when authentication is enabled to protect tokens.
share	Controls document sharing policy for connected clients. This unified parameter governs both proactive document announcement (pushing documents to clients) and access control (allowing clients to request documents). Accepts one of: • `NA` (default) — never announce but allow all requests. • `TRUE` — announce all documents to all clients and allow all requests. • `FALSE` — never announce and deny all requests (sends `doc-unavailable`). • A function with signature `function(client_id, doc_id)` returning `TRUE` (announce and allow), `NA` (allow on request only), or `FALSE` (deny access). Called per client and per document.

Details

The returned server inherits from nanonext's nanoServer class and provides `$start()` and `$close()` methods for non-blocking operation.

Value

An autosync_server object inheriting from 'nanoServer', with `$start()` and `$close()` methods.

Examples

```
# Create and start a server
server <- sync_server()
server$start()

# Server is now running in the background
# ...do other work...

# Stop when done
server$close()

# With TLS for secure connections
cert <- nanonext::write_cert()
tls <- nanonext::tls_config(server = cert$server)
server <- sync_server(tls = tls)
server$start()
server$url
server$close()

# Server with OIDC authentication (requires TLS)
cert <- nanonext::write_cert()
tls <- nanonext::tls_config(server = cert$server)
server <- sync_server(
  tls = tls,
  auth = auth_config(
    client_id = "123456789.apps.googleusercontent.com",
    allowed_domains = "mycompany.com"
  )
)
```

---

sync_token *Obtain an OIDC token interactively*

Description

Performs the OAuth 2.0 Authorization Code flow with PKCE to obtain a JWT (ID token) from an OIDC provider, delegating the browser handshake and token exchange to **httr2**. Endpoints are discovered from the issuer's `.well-known` metadata via `httr2::oauth_server_metadata()`, and the flow is run by `httr2::oauth_flow_auth_code()`: it opens the system browser, listens on a loopback redirect for the callback, and returns the ID token for use with `sync_fetch()`.

Usage

```
sync_token(
  client_id = Sys.getenv("OIDC_CLIENT_ID"),
  client_secret = Sys.getenv("OIDC_CLIENT_SECRET"),
  issuer = oidc_issuer(),
  scopes = "openid email",
  redirect_uri = oauth_redirect_uri()
)
```

Arguments

client_id	The OIDC client ID (application ID). Defaults to the OIDC_CLIENT_ID environment variable.
client_secret	The OIDC client secret. Required by Google (Desktop app) and "Web application" client types; leave unset for native / public clients, which authenticate via PKCE alone. Defaults to the OIDC_CLIENT_SECRET environment variable.
issuer	The OIDC issuer URL. Defaults to the OIDC_ISSUER environment variable, falling back to Google ("https://accounts.google.com").
scopes	Space-separated OAuth scopes to request. Default "openid email".
redirect_uri	Local redirect URI for the OAuth callback. Defaults to httr2::oauth_redirect_uri(), i.e. "http://localhost" with an OS-assigned random port (or the HTTR2_OAUTH_REDIRECT_URL environment variable on hosted platforms). Supply an explicit port (e.g. "http://localhost:8080") when your OIDC provider requires the redirect URI to match a pre-registered value.

Details

For Google, register the OAuth client as a "Desktop app" and set both OIDC_CLIENT_ID and OIDC_CLIENT_SECRET. Google's Desktop app secret is required in the token exchange but, unlike a "Web application" secret, is not treated as confidential: Google states that for installed apps "the client secret is obviously not treated as a secret" (https://developers.google.com/identity/protocols/oauth2#installed), consistent with the OAuth 2.0 for Native Apps standard (RFC 8252 section 8.5, https://datatracker.ietf.org/doc/html/rfc8252#section-8.5). Providers that support native / public clients (Microsoft Entra, Okta, Auth0, etc.) need only client_id, authenticating via PKCE alone; leave client_secret unset for these.

Value

A JWT (ID token) as a character string.

Examples

```
# Uses OIDC_CLIENT_ID and OIDC_CLIENT_SECRET env vars by default
token <- sync_token()

# Or supply credentials directly
token <- sync_token(
  client_id = "YOUR_CLIENT_ID.apps.googleusercontent.com",
```

```
  client_secret = "YOUR_CLIENT_SECRET"
)

# Use with sync_fetch
doc <- sync_fetch(server$url, "myDocId", token = token, tls = tls)
```

Index