Package 'mx.crypto'

September 11, 2026

Type Package

Title Matrix End-to-End Encryption Primitives

Version 0.2.2

Date 2026-09-11

Description 'Olm' and 'Megolm' encryption ratchet primitives for the 'Matrix' messaging protocol <https://matrix.org/>, wrapping the 'vodozemac' Rust crate. Provides device-key generation, one-time-key management, 1:1 'Olm' sessions, and 'Megolm' group sessions. Pairs with the 'mx.api' package, which handles 'Matrix' HTTP transport.

License MIT + file LICENSE | Apache License 2.0

URL https://github.com/cornball-ai/mx.crypto

BugReports https://github.com/cornball-ai/mx.crypto/issues

Depends R (>= 4.3)

SystemRequirements Cargo (Rust's package manager), rustc (>= 1.85)

Suggests tinytest, mx.api (>= 0.2.0), simplermarkdown

VignetteBuilder simplermarkdown

Encoding UTF-8

NeedsCompilation yes

Author Troy Hernandez [aut, cre] (ORCID:
<https://orcid.org/0009-0005-4248-604X>),
cornball.ai [cph],
The Matrix.org Foundation C.I.C. [ctb, cph] (Authors of the bundled 'vodozemac' Rust crate; see inst/AUTHORS),
Authors of the dependency Rust crates [ctb] (see inst/AUTHORS for details)

Maintainer Troy Hernandez <troy@cornball.ai>

Repository CRAN

Date/Publication 2026-09-11 11:20:02 UTC

Contents

mx.crypto-package *Matrix End-to-End Encryption Primitives*

Description

'Olm' and 'Megolm' encryption ratchet primitives for the 'Matrix' messaging protocol <https://matrix.org/>, wrapping the 'vodozemac' Rust crate. Provides device-key generation, one-time-key management, 1:1 'Olm' sessions, and 'Megolm' group sessions. Pairs with the 'mx.api' package, which handles 'Matrix' HTTP transport.

Package Content

Index of help topics:

```
                        Inspect an outbound group session
mxc_megolm_outbound_new
                        Create an outbound Megolm group session
mxc_megolm_outbound_pickle
                        Pickle an outbound group session
mxc_megolm_outbound_unpickle
                        Restore an outbound group session from a pickle
mxc_olm_create_inbound
                        Build an inbound Olm session from a pre-key
                        message
mxc_olm_create_outbound
                        Start an outbound Olm session
mxc_olm_decrypt         Decrypt a message on an Olm session
mxc_olm_encrypt         Encrypt a message on an Olm session
mxc_olm_session_pickle
                        Pickle an Olm session
mxc_olm_session_unpickle
                        Restore an Olm session from a pickle
mxc_sas_bytes           Derive the six Matrix SAS display bytes
mxc_sas_commitment      Hash a Matrix SAS public key and canonical
                        start message
mxc_sas_establish       Establish a SAS shared secret
mxc_sas_mac             Authenticate a Matrix SAS key or key-id list
mxc_sas_new             Create an ephemeral Matrix SAS key agreement
mxc_sas_public          Read a SAS ephemeral public key
mxc_sas_verify_mac      Verify a Matrix SAS authentication tag
mxc_signing_key_new     Create a durable Ed25519 signing key
mxc_signing_key_pickle
                        Encrypt and pickle a signing key
mxc_signing_key_public
                        Read a signing key's public Ed25519 key
mxc_signing_key_sign    Sign canonical JSON with a signing key
mxc_signing_key_unpickle
                        Restore an encrypted signing key pickle
mxc_verify_device_keys
                        Verify a Matrix device-keys object
mxc_verify_one_time_key
                        Verify a signed one-time / fallback key
```

Maintainer

Troy Hernandez <troy@cornball.ai>

Author(s)

Troy Hernandez [aut, cre] (ORCID: <https://orcid.org/0009-0005-4248-604X>), cornball.ai [cph], The Matrix.org Foundation C.I.C. [ctb, cph] (Authors of the bundled 'vodozemac' Rust crate; see inst/AUTHORS), Authors of the dependency Rust crates [ctb] (see inst/AUTHORS for details)

mxc_account_fallback_key

Generate and return a fallback key

Description

Generates a fallback curve25519 key, which the homeserver hands out when an OTK pool is exhausted. Returns the freshly generated key. Calling it again rotates the previous fallback.

Usage

```
mxc_account_fallback_key(account)
```

Arguments

account	An Account.

Value

Named list with ‘key_id‘ and ‘curve25519‘ (both base64).

mxc_account_generate_one_time_keys

Generate one-time keys

Description

Adds ‘n‘ curve25519 one-time keys to the Account’s pool. Call [mxc_account_one_time_keys] to read them out for upload, then [mxc_account_mark_published] once the homeserver has accepted them.

Usage

```
mxc_account_generate_one_time_keys(account, n)
```

Arguments

account	An Account.
n	Number of OTKs to generate.

Value

Invisible NULL; mutates ‘account‘ in place.

Examples

```
acct <- mxc_account_new()
mxc_account_generate_one_time_keys(acct, 5L)
```

mxc_account_identity_keys

Public identity keys for an Account

Description

Returns the device's two public keys: curve25519 (Olm sender / identity key, used to start sessions) and ed25519 (signing / fingerprint key). Both are unpadded base64 strings, ready for '/keys/upload'.

Usage

```
mxc_account_identity_keys(account)
```

Arguments

account	An Account from [mxc_account_new] or [mxc_account_unpickle].

Value

A named list with 'curve25519' and 'ed25519' strings.

Examples

```
k <- mxc_account_identity_keys(mxc_account_new())
k$curve25519
```

mxc_account_mark_published

Mark current one-time keys as published

Description

Call after the homeserver has accepted a '/keys/upload' containing the keys returned by [mxc_account_one_time_keys]. Future calls to that function will not re-include them.

Usage

```
mxc_account_mark_published(account)
```

Arguments

account	An Account.

Value

Invisible NULL; mutates 'account'.

mxc_account_new *Create a new Olm Account*

Description

Creates a fresh device identity. Holds long-lived curve25519 / ed25519 identity keys and a one-time-key pool. The returned object is an external pointer; persist it with [mxc_account_pickle].

Usage

```
mxc_account_new()
```

Value

An external pointer to a vodozemac Account.

Examples

```
acct <- mxc_account_new()
mxc_account_identity_keys(acct)
```

mxc_account_one_time_keys *Read pending one-time keys*

Description

Returns OTKs that have been generated but not yet marked as published. Each value is a curve25519 public key; the caller should sign each with [mxc_account_sign] and upload as ‘signed_curve25519:<key_id>‘.

Usage

```
mxc_account_one_time_keys(account)
```

Arguments

account	An Account.

Value

Named list mapping ‘key_id‘ to ‘curve25519_pub‘ (both unpadded base64 strings).

Examples

```
acct <- mxc_account_new()
mxc_account_generate_one_time_keys(acct, 5L)
otks <- mxc_account_one_time_keys(acct)
```

mxc_account_pickle *Pickle an Account to an encrypted blob*

Description

Serialises the Account's state and encrypts it under a 32-byte key. Restore with [mxc_account_unpickle]. Pickle after every state change that you want to survive a restart (OTK generation, fallback key rotation, mark-published).

Usage

```
mxc_account_pickle(account, key)
```

Arguments

account	An Account.
key	A 'raw' vector of length 32. The caller is responsible for key derivation; use a KDF on a passphrase, do not pass a passphrase.

Value

Base64 string containing the encrypted pickle.

mxc_account_sign *Sign canonical JSON with the Account's ed25519 key*

Description

The caller must canonicalise the JSON per the Matrix spec (UTF-8, sorted keys, no whitespace, no insignificant data) before passing it in. mx.crypto does not canonicalise; mx.api will.

Usage

```
mxc_account_sign(account, canonical_json)
```

Arguments

account	An Account.
canonical_json	A character string containing canonical JSON.

Value

Unpadded base64 ed25519 signature.

Examples

```
acct <- mxc_account_new()
sig <- mxc_account_sign(acct, '{"hello":"world"}')
```

mxc_account_unpickle *Restore an Account from an encrypted pickle*

Description

Restore an Account from an encrypted pickle

Usage

```
mxc_account_unpickle(blob, key)
```

Arguments

blob	A base64 string produced by [mxc_account_pickle].
key	The 32-byte key the pickle was encrypted under.

Value

An Account external pointer.

mxc_ed25519_verify *Verify an Ed25519 signature*

Description

Thin wrapper over vodozemac's strict Ed25519 verifier (`verify_strict`). Returns `TRUE` when the signature is valid and `FALSE` when it isn't; raises an error if any of `public_key`, `message`, or `signature` is malformed (bad base64, wrong length, etc.).

Usage

```
mxc_ed25519_verify(public_key, message, signature)
```

Arguments

public_key	Character. Unpadded base64 ed25519 public key (the same shape `mxc_account_identity_keys` returns).
message	A `raw` vector. The byte sequence the signer ran ed25519 over; typically the output of `mx.api::mx_canonical_json()`.
signature	Character. Unpadded base64 ed25519 signature (86 chars).

Value

Single logical: `TRUE` for a valid signature, `FALSE` otherwise.

Examples

```
acct <- mxc_account_new()
ids <- mxc_account_identity_keys(acct)
sig <- mxc_account_sign(acct, "{\"hello\":\"world\"}")
mxc_ed25519_verify(ids$ed25519, charToRaw("{\"hello\":\"world\"}"), sig)
```

`mxc_megolm_decrypt` *Decrypt a room message*

Description

Decrypt a room message

Usage

```
mxc_megolm_decrypt(igs, ciphertext_b64)
```

Arguments

`igs`	An InboundGroupSession.
`ciphertext_b64`	Base64 ciphertext (the ‘ciphertext‘ field of the ‘m.room.encrypted‘ event).

Value

Named list: ‘plaintext‘ (raw) and ‘message_index‘ (integer). Use ‘message_index‘ to dedupe replays.

`mxc_megolm_encrypt` *Encrypt a room message*

Description

Encrypt a room message

Usage

```
mxc_megolm_encrypt(gs, plaintext)
```

Arguments

`gs`	An outbound GroupSession.
`plaintext`	A ‘raw‘ vector. The caller is responsible for producing the canonical JSON event body.

Value

Base64 ciphertext (the ‘ciphertext‘ field of ‘m.room.encrypted‘).

`mxc_megolm_inbound_export` *Export an inbound Megolm session for forwarding*

Description

Exports at the first known message index, retaining the maximum history this recipient can legitimately share. The result is the ‘session_key‘ value for ‘m.forwarded_room_key‘.

Usage

```
mxc_megolm_inbound_export(igs)
```

Arguments

`igs`	An InboundGroupSession.

Value

Base64 exported session key.

`mxc_megolm_inbound_import` *Import a forwarded inbound Megolm session*

Description

Imports the exported session-key format carried by ‘m.forwarded_room_key‘. Unlike [mxc_megolm_inbound_new()], the resulting session does not treat the original sender signature as verified; the caller must validate the forwarding chain and sender identity.

Usage

```
mxc_megolm_inbound_import(session_key)
```

Arguments

`session_key`	Base64 exported session key.

Value

An InboundGroupSession external pointer.

`mxc_megolm_inbound_info` *Inspect an inbound Megolm session*

Description

Inspect an inbound Megolm session

Usage

```
mxc_megolm_inbound_info(igs)
```

Arguments

`igs`	An InboundGroupSession.

Value

Named list with ‘session_id‘ and ‘first_known_index‘.

`mxc_megolm_inbound_new` *Build an inbound Megolm session from a shared session_key*

Description

Constructs the receiver-side ratchet using the ‘session_key‘ that was delivered (over Olm) in an ‘m.room_key‘ event. The receiver stores the resulting object indexed by ‘(sender_curve25519, session_id)‘ and re-uses it to decrypt incoming messages on that session.

Usage

```
mxc_megolm_inbound_new(session_key)
```

Arguments

`session_key`	Base64 session_key from ‘m.room_key‘.

Value

An InboundGroupSession external pointer.

mxc_megolm_inbound_pickle

Pickle an inbound group session

Description

Pickle after every decrypt so the ratchet state survives restart.

Usage

```
mxc_megolm_inbound_pickle(igs, key)
```

Arguments

igs	An InboundGroupSession.
key	32-byte raw vector.

Value

Base64 string.

mxc_megolm_inbound_unpickle

Restore an inbound group session from a pickle

Description

Restore an inbound group session from a pickle

Usage

```
mxc_megolm_inbound_unpickle(blob, key)
```

Arguments

blob	Base64 string.
key	32-byte raw vector.

Value

An InboundGroupSession external pointer.

`mxc_megolm_outbound_info` *Inspect an outbound group session*

Description

Returns the ‘session_id‘, current ‘session_key‘ (the value to ship via ‘m.room_key‘ to recipient devices), and ‘message_index‘.

Usage

```
mxc_megolm_outbound_info(gs)
```

Arguments

`gs`	An outbound GroupSession.

Value

Named list: ‘session_id‘ (str), ‘session_key‘ (str), ‘message_index‘ (int).

`mxc_megolm_outbound_new` *Create an outbound Megolm group session*

Description

The sender side of a Megolm ratchet. Use [mxc_megolm_outbound_info] to read out the ‘session_key‘ that must be shared (over Olm) with each recipient device, and [mxc_megolm_encrypt] to encrypt room messages. Rotate (create a new one) on membership changes or after your chosen number-of-messages / time threshold.

Usage

```
mxc_megolm_outbound_new()
```

Value

An outbound GroupSession external pointer.

`mxc_megolm_outbound_pickle` *Pickle an outbound group session*

Description

Pickle an outbound group session

Usage

```
mxc_megolm_outbound_pickle(gs, key)
```

Arguments

`gs`	An outbound GroupSession.
`key`	32-byte raw vector.

Value

Base64 string.

`mxc_megolm_outbound_unpickle` *Restore an outbound group session from a pickle*

Description

Restore an outbound group session from a pickle

Usage

```
mxc_megolm_outbound_unpickle(blob, key)
```

Arguments

`blob`	Base64 string.
`key`	32-byte raw vector.

Value

An outbound GroupSession external pointer.

`mxc_olm_create_inbound`

Build an inbound Olm session from a pre-key message

Description

Consumes the matching one-time key from the local Account. The result has the new Session and the decrypted plaintext of the pre-key message; subsequent messages on this session use [mxc_olm_decrypt].

Usage

```
mxc_olm_create_inbound(account, peer_curve25519, prekey_b64)
```

Arguments

`account`	The local Account (will be mutated: an OTK is consumed).
`peer_curve25519`	Sender's curve25519 identity key (base64).
`prekey_b64`	Body of the pre-key message (base64).

Value

Named list: 'session' (external pointer) and 'plaintext' (raw).

`mxc_olm_create_outbound`

Start an outbound Olm session

Description

Use a peer's published curve25519 identity key + one of their signed one-time keys to bootstrap a 1:1 ratchet. The first ciphertext from this session is a pre-key message ('type = 0').

Usage

```
mxc_olm_create_outbound(account, peer_curve25519, peer_otk)
```

Arguments

`account`	The local Account.
`peer_curve25519`	Peer's curve25519 identity key (base64).
`peer_otk`	Peer's one-time key (base64).

Value

An Olm Session external pointer.

`mxc_olm_decrypt` *Decrypt a message on an Olm session*

Description

Decrypt a message on an Olm session

Usage

```
mxc_olm_decrypt(session, type, body)
```

Arguments

`session`	An Olm Session.
`type`	Message type: ‘0L‘ for pre-key, ‘1L‘ for normal.
`body`	Ciphertext (base64).

Value

A ‘raw‘ vector of plaintext bytes.

`mxc_olm_encrypt` *Encrypt a message on an Olm session*

Description

Encrypt a message on an Olm session

Usage

```
mxc_olm_encrypt(session, plaintext)
```

Arguments

`session`	An Olm Session.
`plaintext`	A ‘raw‘ vector.

Value

Named list: ‘type‘ (‘0L‘ pre-key, ‘1L‘ normal) and ‘body‘ (base64).

mxc_olm_session_pickle

Pickle an Olm session

Description

Pickle after every encrypt/decrypt to keep the ratchet state on disk.

Usage

```
mxc_olm_session_pickle(session, key)
```

Arguments

session	An Olm Session.
key	32-byte raw vector.

Value

Base64 string.

mxc_olm_session_unpickle

Restore an Olm session from a pickle

Description

Restore an Olm session from a pickle

Usage

```
mxc_olm_session_unpickle(blob, key)
```

Arguments

blob	Base64 string produced by [mxc_olm_session_pickle].
key	32-byte raw vector.

Value

An Olm Session external pointer.

mxc_sas_bytes *Derive the six Matrix SAS display bytes*

Description

Derive the six Matrix SAS display bytes

Usage

```
mxc_sas_bytes(sas, info)
```

Arguments

sas	An established SAS handle.
info	The protocol's ordered SAS context string.

Value

A raw vector of length six, for decimal or emoji presentation.

mxc_sas_commitment *Hash a Matrix SAS public key and canonical start message*

Description

Computes SHA-256 over the concatenated UTF-8 strings and returns unpadded base64. The caller must canonicalize the complete start content first. This function does not parse JSON or implement protocol policy.

Usage

```
mxc_sas_commitment(public_key, canonical_start)
```

Arguments

public_key	Unpadded base64 ephemeral Curve25519 public key.
canonical_start	Canonical JSON of the complete start content.

Value

The unpadded base64 SHA-256 commitment.

mxc_sas_establish *Establish a SAS shared secret*

Description

Consumes the handle's ephemeral secret exactly once. A low-order peer key is rejected and consumes the handle. No shared secret is returned to R.

Usage

```
mxc_sas_establish(sas, peer_key)
```

Arguments

sas	An opaque SAS handle, modified in place.
peer_key	The peer's unpadded base64 ephemeral public key.

Value

Invisibly NULL.

mxc_sas_mac *Authenticate a Matrix SAS key or key-id list*

Description

Authenticate a Matrix SAS key or key-id list

Usage

```
mxc_sas_mac(sas, input, info)
```

Arguments

sas	An established SAS handle.
input	Public key or sorted key-id list to authenticate.
info	The protocol's direction-specific MAC context string.

Value

An unpadded base64 HMAC-SHA-256 tag for hkdf-hmac-sha256.v2.

mxc_sas_new *Create an ephemeral Matrix SAS key agreement*

Description

Wraps vodozemac's short-authentication-string key agreement. The handle cannot be saved or restored; restart interrupted verification with a new transaction and handle. Protocol negotiation, commitments, user prompts, timeouts, and cross-signing policy belong to mx.client.

Usage

```
mxc_sas_new()
```

Value

An opaque SAS handle.

mxc_sas_public *Read a SAS ephemeral public key*

Description

Read a SAS ephemeral public key

Usage

```
mxc_sas_public(sas)
```

Arguments

sas	An opaque SAS handle.

Value

An unpadded base64 Curve25519 public key.

mxc_sas_verify_mac *Verify a Matrix SAS authentication tag*

Description

Uses vodozemac's constant-time MAC verification. Malformed tags return FALSE; an invalid or unestablished handle raises an error.

Usage

```
mxc_sas_verify_mac(sas, input, info, mac)
```

Arguments

sas	An established SAS handle.
input	Public key or sorted key-id list being authenticated.
info	The protocol's direction-specific MAC context string.
mac	Unpadded base64 HMAC-SHA-256 tag.

Value

TRUE for a matching tag, otherwise FALSE.

mxc_signing_key_new *Create a durable Ed25519 signing key*

Description

Creates a key suitable for a Matrix cross-signing role. Persist it with [mxc_signing_key_pickle()]; the private material never needs to be exposed as a string or raw vector.

Usage

```
mxc_signing_key_new()
```

Value

An opaque signing-key handle.

mxc_signing_key_pickle

Encrypt and pickle a signing key

Description

Uses the same vodozemac pickle encryption as an Olm account. Although the carrier also contains an unused Curve25519 identity, only the Ed25519 key is part of the signing-key contract.

Usage

```
mxc_signing_key_pickle(key, pickle_key)
```

Arguments

key	A signing-key handle.
pickle_key	A 32-byte raw encryption key.

Value

Encrypted base64 pickle.

mxc_signing_key_public

Read a signing key's public Ed25519 key

Description

Read a signing key's public Ed25519 key

Usage

```
mxc_signing_key_public(key)
```

Arguments

key	A signing-key handle.

Value

Unpadded base64 Ed25519 public key.

mxc_signing_key_sign *Sign canonical JSON with a signing key*

Description

Sign canonical JSON with a signing key

Usage

```
mxc_signing_key_sign(key, canonical_json)
```

Arguments

key	A signing-key handle.
canonical_json	A canonical JSON character string.

Value

Unpadded base64 Ed25519 signature.

mxc_signing_key_unpickle *Restore an encrypted signing key pickle*

Description

Restore an encrypted signing key pickle

Usage

```
mxc_signing_key_unpickle(blob, pickle_key)
```

Arguments

blob	An encrypted pickle from [mxc_signing_key_pickle()].
pickle_key	The same 32-byte raw encryption key.

Value

An opaque signing-key handle.

mxc_verify_device_keys *Verify a Matrix device-keys object*

Description

Validates a single device_keys object as returned by /_matrix/client/v3/keys/query: checks that the structural fields (user_id, device_id, algorithms, keys, signatures) are present and match the expected identity, then verifies the ed25519 self-signature over the canonical-JSON byte sequence (with signatures and unsigned stripped). Any of the following raises an error rather than returning FALSE: missing field, wrong user_id or device_id, missing curve25519 / ed25519 key, missing signatures block, signature under the wrong user or device, invalid signature, malformed base64. This function deliberately accepts *any* ed25519 key the object claims for itself; it does not pin against a previously trusted key. Identity pinning is the caller's responsibility (typically TOFU + cross-signing).

Usage

```
mxc_verify_device_keys(device_keys, expected_user_id, expected_device_id,
                       required_algorithms = NULL)
```

Arguments

device_keys	Named list. The device_keys object, exactly as parsed from a homeserver response (no signatures block stripped).
expected_user_id	Character. Matrix user id the device is supposed to belong to.
expected_device_id	Character. The device id we're verifying.
required_algorithms	Character vector or NULL. Algorithm identifiers that must appear in device_keys$algorithms for the device to be accepted. NULL (the default) means "both Olm and Megolm" — i.e. m.olm.v1.curve25519-aes-sha2 and m.megolm.v1.aes-sha2. Pass character(0) to skip the check, but be aware that the audit plan requires it.

Value

Named list with the verified curve25519 and ed25519 public keys (both base64). Use these as the inputs for Olm session establishment.

Examples

```
## Not run:
q <- mx.api::mx_keys_query(s, list("@alice:server" = "ALICEDEV"))
dk <- q$device_keys[["@alice:server"]][["ALICEDEV"]]
keys <- mxc_verify_device_keys(dk, "@alice:server", "ALICEDEV")
keys$curve25519
```

```
## End(Not run)
```

`mxc_verify_one_time_key` *Verify a signed one-time / fallback key*

Description

Validates a single signed_curve25519 object returned by `/_matrix/client/v3/keys/claim` (or the `fallback_keys` block of `/keys/upload`). The signing ed25519 key must come from a previously verified `device_keys` object — *this function does not look it up for you*, because doing so would silently re-trust whatever the homeserver hands back. The caller must pass the outer map key (`"<algorithm>:<key_id>"`) alongside the inner key object so the helper can reject anything but `signed_curve25519`. The helper also confirms the `key` value decodes to a 32-byte curve25519 public key; a signature over garbage bytes is not a useful "verified" result.

Usage

```
mxc_verify_one_time_key(algorithm_key_id, key_object, signing_ed25519,
                        expected_user_id, expected_device_id)
```

Arguments

`algorithm_key_id`
: Character. The outer map key from the `one_time_keys` or `fallback_keys` response, e.g. `"signed_curve25519:AAAAAAAAAAA"`. The algorithm prefix must be `signed_curve25519`.

`key_object`
: Named list. The signed key object (must contain `key` and `signatures`).

`signing_ed25519`
: Character. The base64 ed25519 public key that should have signed this OTK (i.e. the ed25519 returned by `mxc_verify_device_keys` for the same device).

`expected_user_id`
: Character. Matrix user id the OTK is supposed to come from.

`expected_device_id`
: Character. Matrix device id.

Value

Character. The verified curve25519 OTK (base64), ready to feed into `mxc_olm_create_outbound`.

Examples

```
## Not run:
cl <- mx.api::mx_keys_claim(s, list(
  "@alice:server" = list("ALICEDEV" = "signed_curve25519")
))
entry <- cl$one_time_keys[["@alice:server"]][["ALICEDEV"]]
algo_kid <- names(entry)[[1]]   # e.g. "signed_curve25519:AAAA"
otk <- mxc_verify_one_time_key(
  algo_kid, entry[[1]], alice_ed, "@alice:server", "ALICEDEV"
)

## End(Not run)
```

Index