Package: keyring (via r-universe)

June 28, 2024

Title Access the System Credential Store from R

Version 1.3.2.9000

Description Platform independent 'API' to access the operating system's credential store. Currently supports: 'Keychain' on 'macOS', Credential Store on 'Windows', the Secret Service 'API' on 'Linux', and simple, platform independent stores implemented with environment variables or encrypted files. Additional storage back-ends can be added easily.

License MIT + file LICENSE

URL https://keyring.r-lib.org/, https://github.com/r-lib/keyring

BugReports https://github.com/r-lib/keyring/issues

Depends R (>= 3.6)

Imports askpass, assertthat, filelock, openssl, R6, rappdirs, sodium, tools, utils, yaml

Suggests callr, covr, mockery, testthat (>= 3.0.0), withr

Config/Needs/website tidyverse/tidytemplate

Config/testthat/edition 3

Encoding UTF-8

Roxygen list(markdown = TRUE, r6 = FALSE)

RoxygenNote 7.2.3

SystemRequirements Optional: libsecret on Linux (libsecret-1-dev on Debian/Ubuntu, libsecret-devel on Fedora/CentOS)

Collate 'api.R' 'assertions.R' 'backend-class.R' 'backend-env.R' 'backend-file.R' 'backend-macos.R' 'backend-secret-service.R' 'backend-wincred.R' 'default_backend.R' 'keyring-package.R' 'package.R' 'pass.R' 'utils.R'

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

RemoteUrl https://github.com/r-lib/keyring

RemoteRef HEAD

RemoteSha b01faec046b0d050ce58b7133e3895626af12d09

Contents

---

backend *Abstract class of a minimal keyring backend*

Description

To implement a new keyring backend, you need to inherit from this class and then redefine the `get`, `set`, `set_with_value` and `delete` methods. Implementing the `list` method is optional. Additional methods can be defined as well.

Details

These are the semantics of the various methods:

```
get(service, username = NULL, keyring = NULL)
get_raw(service, username = NULL, keyring = NULL)
set(service, username = NULL, keyring = NULL, prompt = "Password: ")
set_with_value(service, username = NULL, password = NULL,
               keyring = NULL)
set_with_raw_value(service, username = NULL, password = NULL,
               keyring = NULL)
delete(service, username = NULL, keyring = NULL)
list(service = NULL, keyring = NULL)
```

What these functions do:

- `get()` queries the secret in a keyring item.
- `get_raw()` is similar to `get()`, but returns the result as a raw vector.
- `set()` sets the secret in a keyring item. The secret itself is read in interactively from the keyboard.
- `set_with_value()` sets the secret in a keyring item to the specified value.
- `set_with_raw_value()` sets the secret in keyring item to the byte sequence of a raw vector.
- `delete()` remotes a keyring item.

- `list()` lists keyring items.

The arguments:

- `service` String, the name of a service. This is used to find the secret later.
- `username` String, the username associated with a secret. It can be `NULL`, if no username belongs to the secret. It uses the value of the `keyring_username`, if set.
- `keyring` String, the name of the keyring to work with. This only makes sense if the platform supports multiple keyrings. `NULL` selects the default (and maybe only) keyring.
- `password` The value of the secret, typically a password, or other credential.
- `prompt` String, the text to be displayed above the textbox.

See Also

Other keyring backend base classes: `backend_keyrings`

`backends` *Select the default backend and default keyring*

Description

The default backend is selected

1. based on the `keyring_backend` option. See `base::options()`. This can be set to a character string, and then the *backend*`_string` class is used to create the default backend.
2. If this is not set, then the `R_KEYRING_BACKEND` environment variable is checked.
3. If this is not set, either, then the backend is selected automatically, based on the OS:
   (a) On Windows, the Windows Credential Store (`"wincred"`) is used.
   (b) On macOS, Keychain services are selected (`"macos"`).
   (c) Linux uses the Secret Service API (`"secret_service"`), and it also checks that the service is available. It is typically only available on systems with a GUI.
   (d) If the file backend (`"file"`) is available, it is selected.
   (e) On other operating systems, secrets are stored in environment variables (`"env"`).

Usage

```
default_backend(keyring = NULL)
```

Arguments

`keyring`	Character string, the name of the keyring to use, or `NULL` for the default keyring.

Details

Most backends support multiple keyrings. For these the keyring is selected from:

1. the supplied `keyring` argument (if not `NULL`), or
2. the `keyring_keyring` option.
   - You can change this by using `options(keyring_keyring = "NEWVALUE")`
3. If this is not set, the `R_KEYRING_KEYRING` environment variable.
   - Change this value with `Sys.setenv(R_KEYRING_KEYRING = "NEWVALUE")`, either in your script or in your `.Renviron` file. See base::Startup for information about using `.Renviron`
4. Finally, if neither of these are set, the OS default keyring is used.
   - Usually the keyring is automatically unlocked when the user logs in.

Value

The backend object itself.

See Also

backend_env, backend_file, backend_macos, backend_secret_service, backend_wincred

---

`backend_env` *Environment variable keyring backend*

Description

This is a simple keyring backend, that stores/uses secrets in environment variables of the R session.

Details

It does not support multiple keyrings. It also does not support listing all keys, since there is no way to distinguish keys from regular environment variables.

It does support service names and usernames: they will be separated with a `:` character in the name of the environment variable. (Note that such an environment variable typically cannot be set or queried from a shell, but it can be set and queried from R or other programming languages.)

See backend for the documentation of the class's methods.

See Also

Other keyring backends: `backend_file`, `backend_macos`, `backend_secret_service`, `backend_wincred`

Examples

```
## Not run:
env <- backend_env$new()
env$set("r-keyring-test", username = "donaldduck")
env$get("r-keyring-test", username = "donaldduck")
Sys.getenv("r-keyring-test:donaldduck")

# This is an error
env$list()

# Clean up
env$delete("r-keyring-test", username = "donaldduck")

## End(Not run)
```

---

backend_file *Encrypted file keyring backend*

Description

This is a simple keyring backend, that stores/uses secrets in encrypted files.

Details

It supports multiple keyrings.

See backend for the documentation of the individual methods.

See Also

Other keyring backends: backend_env, backend_macos, backend_secret_service, backend_wincred

Examples

```
## Not run:
kb <- backend_file$new()

## End(Not run)
```

backend_keyrings *Abstract class of a backend that supports multiple keyrings*

Description

To implement a new keyring that supports multiple keyrings, you need to inherit from this class and redefine the `get`, `set`, `set_with_value`, `delete`, `list` methods, and also the keyring management methods: `keyring_create`, `keyring_list`, `keyring_delete`, `keyring_lock`, `keyring_unlock`, `keyring_is_locked`, `keyring_default` and `keyring_set_default`.

Details

See backend for the first set of methods. This is the semantics of the keyring management methods:

```
keyring_create(keyring)
keyring_list()
keyring_delete(keyring = NULL)
keyring_lock(keyring = NULL)
keyring_unlock(keyring = NULL, password = NULL)
keyring_is_locked(keyring = NULL)
keyring_default()
keyring_set_default(keyring = NULL)
```

- `keyring_create()` creates a new keyring.
- `keyring_list()` lists all keyrings.
- `keyring_delete()` deletes a keyring. It is a good idea to protect the default keyring, and/or a non-empty keyring with a password or a confirmation dialog.
- `keyring_lock()` locks a keyring.
- `keyring_unlock()` unlocks a keyring.
- `keyring_is_locked()` checks whether a keyring is locked.
- `keyring_default()` returns the default keyring.
- `keyring_set_default()` sets the default keyring.

Arguments:

- `keyring` is the name of the keyring to use or create. For some methods in can be `NULL` to select the default keyring.
- `password` is the password of the keyring.

See Also

Other keyring backend base classes: backend

backend_macos *macOS Keychain keyring backend*

Description

This backend is the default on macOS. It uses the macOS native Keychain Service API.

Details

It supports multiple keyrings.

See backend for the documentation of the individual methods.

See Also

Other keyring backends: backend_env, backend_file, backend_secret_service, backend_wincred

Examples

```
## Not run:
## This only works on macOS
kb <- backend_macos$new()
kb$keyring_create("foobar")
kb$set_default_keyring("foobar")
kb$set_with_value("service", password = "secret")
kb$get("service")
kb$delete("service")
kb$delete_keyring("foobar")

## End(Not run)
```

backend_secret_service *Linux Secret Service keyring backend*

Description

This backend is the default on Linux. It uses the libsecret library, and needs a secret service daemon running (e.g. Gnome Keyring, or KWallet). It uses DBUS to communicate with the secret service daemon.

Details

This backend supports multiple keyrings.

See backend for the documentation of the individual methods. The `is_available()` method checks is a Secret Service daemon is running on the system, by trying to connect to it. It returns a logical scalar, or throws an error, depending on its argument:

```
is_available = function(report_error = FALSE)
```

Argument:

- `report_error` Whether to throw an error if the Secret Service is not available.

See Also

Other keyring backends: backend_env, backend_file, backend_macos, backend_wincred

Examples

```
## Not run:
## This only works on Linux, typically desktop Linux
kb <- backend_secret_service$new()
kb$keyring_create("foobar")
kb$set_default_keyring("foobar")
kb$set_with_value("service", password = "secret")
kb$get("service")
kb$delete("service")
kb$delete_keyring("foobar")

## End(Not run)
```

---

backend_wincred *Windows Credential Store keyring backend*

Description

This backend is the default on Windows. It uses the native Windows Credential API, and needs at least Windows XP to run.

Details

This backend supports multiple keyrings. Note that multiple keyrings are implemented in the `keyring` R package, using some dummy keyring keys that represent keyrings and their locked/unlocked state.

See backend for the documentation of the individual methods.

See Also

Other keyring backends: backend_env, backend_file, backend_macos, backend_secret_service

Examples

```
## Not run:
## This only works on Windows
kb <- backend_wincred$new()
kb$keyring_create("foobar")
kb$set_default_keyring("foobar")
kb$set_with_value("service", password = "secret")
kb$get("service")
kb$delete("service")
kb$delete_keyring("foobar")

## End(Not run)
```

---

has_keyring_support *Operations on keyrings*

Description

On most platforms keyring supports multiple keyrings. This includes Windows, macOS and Linux (Secret Service) as well. A keyring is a collection of keys that can be treated as a unit. A keyring typically has a name and a password to unlock it. Once a keyring is unlocked, it remains unlocked until the end of the user session, or until it is explicitly locked again.

Usage

```
has_keyring_support()

keyring_create(keyring, password = NULL)

keyring_list()

keyring_delete(keyring = NULL)

keyring_lock(keyring = NULL)

keyring_unlock(keyring = NULL, password = NULL)

keyring_is_locked(keyring = NULL)
```

Arguments

keyring	The name of the keyring to create or to operate on. For functions other than keyring_create, it can also be NULL to select the default keyring.
password	The initial password or the password to unlock the keyring. If not specified or NULL, it will be read from the console.

Details

Platforms typically have a default keyring, which is unlocked automatically when the user logs in. This keyring does not need to be unlocked explicitly.

You can configure the keyring to use via R options or environment variables (see `default_backend()`), or you can also specify it directly in the `default_backend()` call, or in the individual `keyring` calls.

`has_keyring_support` checks if a backend supports multiple keyrings.

`keyring_create` creates a new keyring. It asks for a password if no password is specified.

`keyring_list` lists all existing keyrings.

`keyring_delete` deletes a keyring. Deleting a non-empty keyring requires confirmation, and the default keyring can only be deleted if specified explicitly. On some backends (e.g. Windows Credential Store), the default keyring cannot be deleted at all.

`keyring_lock` locks a keyring. On some backends (e.g. Windows Credential Store), the default keyring cannot be locked.

`keyring_unlock` unlocks a keyring. If a password is not specified, it will be read in interactively.

`keyring_is_locked` queries whether a keyring is locked.

Examples

```
default_backend()
has_keyring_support()
backend_env$new()$has_keyring_support()

## This might ask for a password, so we do not run it by default
## It only works if the default backend supports multiple keyrings
## Not run:
keyring_create("foobar")
key_set_with_value("R-test-service", "donaldduck", password = "secret",
                   keyring = "foobar")
key_get("R-test-service", "donaldduck", keyring = "foobar")
key_list(keyring = "foobar")
keyring_delete(keyring = "foobar")

## End(Not run)
```

`key_get` *Operations on keys*

Description

These functions manipulate keys in a keyring. You can think of a keyring as a secure key-value store.

Usage

```
key_get(service, username = NULL, keyring = NULL)

key_get_raw(service, username = NULL, keyring = NULL)

key_set(service, username = NULL, keyring = NULL, prompt = "Password: ")

key_set_with_value(service, username = NULL, password = NULL, keyring = NULL)

key_set_with_raw_value(
  service,
  username = NULL,
  password = NULL,
  keyring = NULL
)

key_delete(service, username = NULL, keyring = NULL)

key_list(service = NULL, keyring = NULL)
```

Arguments

service	Service name, a character scalar.
username	Username, a character scalar, or NULL if the key is not associated with a username.
keyring	For systems that support multiple keyrings, specify the name of the keyring to use here. If NULL, then the default keyring is used. See also has_keyring_support().
prompt	The character string displayed when requesting the secret
password	The secret to store. For key_set, it is read from the console, interactively. key_set_with_value can be also used in non-interactive mode.

Details

key_get queries a key from the keyring.

key_get_raw queries a key and returns it as a raw vector. Most credential stores allow storing a byte sequence with embedded null bytes, and these cannot be represented as traditional null bytes terminated strings. If you don't know whether the key contains an embedded null, it is best to query it with key_get_raw instead of key_get.

key_set sets a key in the keyring. The contents of the key is read interactively from the terminal.

key_set_with_value is the non-interactive pair of key_set, to set a key in the keyring.

key_set_raw_with_value sets a key to a byte sequence from a raw vector.

key_delete deletes a key.

key_list lists all keys of a keyring, or the keys for a certain service (if service is not NULL).

Encodings:

On Windows, if required, an encoding can be specified using either an R option (`keyring.encoding_windows`) or environment variable (`KEYRING_ENCODING_WINDOWS`). This will be applied when both getting and setting keys. The option takes precedence over the environment variable, if both are set.

This is reserved primarily for compatibility with keys set with other software, such as Python's implementation of keyring. For a list of encodings, use `iconvlist()`, although it should be noted that not *every* encoding can be properly converted, even for trivial cases. For best results, use UTF-8 if you can.

Value

`key_get` returns a character scalar, the password or other confidential information that was stored in the key.

`key_list` returns a list of keys, i.e. service names and usernames, in a data frame.

Examples

```
# These examples use the default keyring, and they are interactive,
# so, we don't run them by default
## Not run:
key_set("R-keyring-test-service", "donaldduck")
key_get("R-keyring-test-service", "donaldduck")
if (has_keyring_support()) key_list(service = "R-keyring-test-service")
key_delete("R-keyring-test-service", "donaldduck")

## This is non-interactive, assuming that that default keyring
## is unlocked
key_set_with_value("R-keyring-test-service", "donaldduck",
                   password = "secret")
key_get("R-keyring-test-service", "donaldduck")
if (has_keyring_support()) key_list(service = "R-keyring-test-service")
key_delete("R-keyring-test-service", "donaldduck")

## This is interactive using backend_file
## Set variables to be used in keyring
kr_name <- "my_keyring"
kr_service <- "my_database"
kr_username <- "my_username"

## Create a keyring and add an entry using the variables above
kb <- keyring::backend_file$new()
## Prompt for the keyring password, used to unlock keyring
kb$keyring_create(kr_name)
## Prompt for the secret/password to be stored in the keyring
kb$set(kr_service, username=kr_username, keyring=kr_name)
# Lock the keyring
kb$keyring_lock(kr_name)

## The keyring file is stored at ~/.config/r-keyring/ on Linux

## Output the stored password
```

```
keyring::backend_file$new()$get(service = kr_service,
  user = kr_username,
  keyring = kr_name)

## End(Not run)
```

Index