# 

> Simple key-value storage with support for multiple backends [![Build Status](https://travis-ci.org/lukechilds/keyv.svg?branch=master)](https://travis-ci.org/lukechilds/keyv) [![Coverage Status](https://coveralls.io/repos/github/lukechilds/keyv/badge.svg?branch=master)](https://coveralls.io/github/lukechilds/keyv?branch=master) [![npm](https://img.shields.io/npm/dm/keyv.svg)](https://www.npmjs.com/package/keyv) [![npm](https://img.shields.io/npm/v/keyv.svg)](https://www.npmjs.com/package/keyv) Keyv provides a consistent interface for key-value storage across multiple backends via storage adapters. It supports TTL based expiry, making it suitable as a cache or a persistent key-value store. ## Features There are a few existing modules similar to Keyv, however Keyv is different because it: - Isn't bloated - Has a simple Promise based API - Suitable as a TTL based cache or persistent key-value store - [Easily embeddable](#add-cache-support-to-your-module) inside another module - Works with any storage that implements the [`Map`](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global\_Objects/Map) API - Handles all JSON types plus `Buffer` - Supports namespaces - Wide range of [\*\*efficient, well tested\*\*](#official-storage-adapters) storage adapters - Connection errors are passed through (db failures won't kill your app) - Supports the current active LTS version of Node.js or higher ## Usage Install Keyv. ``` npm install --save keyv ``` By default everything is stored in memory, you can optionally also install a storage adapter. ``` npm install --save @keyv/redis npm install --save @keyv/mongo npm install --save @keyv/sqlite npm install --save @keyv/postgres npm install --save @keyv/mysql ``` Create a new Keyv instance, passing your connection string if applicable. Keyv will automatically load the correct storage adapter. ```js const Keyv = require('keyv'); // One of the following const keyv = new Keyv(); const keyv = new Keyv('redis://user:pass@localhost:6379'); const keyv = new Keyv('mongodb://user:pass@localhost:27017/dbname'); const keyv = new Keyv('sqlite://path/to/database.sqlite'); const keyv = new Keyv('postgresql://user:pass@localhost:5432/dbname'); const keyv = new Keyv('mysql://user:pass@localhost:3306/dbname'); // Handle DB connection errors keyv.on('error', err => console.log('Connection Error', err)); await keyv.set('foo', 'expires in 1 second', 1000); // true await keyv.set('foo', 'never expires'); // true await keyv.get('foo'); // 'never expires' await keyv.delete('foo'); // true await keyv.clear(); // undefined ``` ### Namespaces You can namespace your Keyv instance to avoid key collisions and allow you to clear only a certain namespace while using the same database. ```js const users = new Keyv('redis://user:pass@localhost:6379', { namespace: 'users' }); const cache = new Keyv('redis://user:pass@localhost:6379', { namespace: 'cache' }); await users.set('foo', 'users'); // true await cache.set('foo', 'cache'); // true await users.get('foo'); // 'users' await cache.get('foo'); // 'cache' await users.clear(); // undefined await users.get('foo'); // undefined await cache.get('foo'); // 'cache' ``` ### Custom Serializers Keyv uses [`json-buffer`](https://github.com/dominictarr/json-buffer) for data serialization to ensure consistency across different backends. You can optionally provide your own serialization functions to support extra data types or to serialize to something other than JSON. ```js const keyv = new Keyv({ serialize: JSON.stringify, deserialize: JSON.parse }); ``` \*\*Warning:\*\* Using custom serializers means you lose any guarantee of data consistency. You should do extensive testing with your serialisation functions and chosen storage engine. ## Official Storage Adapters The official storage adapters are covered by [over 150 integration tests](https://travis-ci.org/lukechilds/keyv/jobs/260418145) to guarantee consistent behaviour. They are lightweight, efficient wrappers over the DB clients making use of indexes and native TTLs where available. Database | Adapter | Native TTL | Status ---|---|---|--- Redis | [@keyv/redis](https://github.com/lukechilds/keyv-redis) | Yes | [![Build Status](https://travis-ci.org/lukechilds/keyv-redis.svg?branch=master)](https://travis-ci.org/lukechilds/keyv-redis) [![Coverage Status](https://coveralls.io/repos/github/lukechilds/keyv-redis/badge.svg?branch=master)](https://coveralls.io/github/lukechilds/keyv-redis?branch=master) MongoDB | [@keyv/mongo](https://github.com/lukechilds/keyv-mongo) | Yes | [![Build Status](https://travis-ci.org/lukechilds/keyv-mongo.svg?branch=master)](https://travis-ci.org/lukechilds/keyv-mongo) [![Coverage Status](https://coveralls.io/repos/github/lukechilds/keyv-mongo/badge.svg?branch=master)](https://coveralls.io/github/lukechilds/keyv-mongo?branch=master) SQLite | [@keyv/sqlite](https://github.com/lukechilds/keyv-sqlite) | No | [![Build Status](https://travis-ci.org/lukechilds/keyv-sqlite.svg?branch=master)](https://travis-ci.org/lukechilds/keyv-sqlite) [![Coverage Status](https://coveralls.io/repos/github/lukechilds/keyv-sqlite/badge.svg?branch=master)](https://coveralls.io/github/lukechilds/keyv-sqlite?branch=master) PostgreSQL | [@keyv/postgres](https://github.com/lukechilds/keyv-postgres) | No | [![Build Status](https://travis-ci.org/lukechilds/keyv-postgres.svg?branch=master)](https://travis-ci.org/lukechildskeyv-postgreskeyv) [![Coverage Status](https://coveralls.io/repos/github/lukechilds/keyv-postgres/badge.svg?branch=master)](https://coveralls.io/github/lukechilds/keyv-postgres?branch=master) MySQL | [@keyv/mysql](https://github.com/lukechilds/keyv-mysql) | No | [![Build Status](https://travis-ci.org/lukechilds/keyv-mysql.svg?branch=master)](https://travis-ci.org/lukechilds/keyv-mysql) [![Coverage Status](https://coveralls.io/repos/github/lukechilds/keyv-mysql/badge.svg?branch=master)](https://coveralls.io/github/lukechilds/keyv-mysql?branch=master) ## Third-party Storage Adapters You can also use third-party storage adapters or build your own. Keyv will wrap these storage adapters in TTL functionality and handle complex types internally. ```js const Keyv = require('keyv'); const myAdapter = require('./my-storage-adapter'); const keyv = new Keyv({ store: myAdapter }); ``` Any store that follows the [`Map`](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global\_Objects/Map) api will work. ```js new Keyv({ store: new Map() }); ``` For example, [`quick-lru`](https://github.com/sindresorhus/quick-lru) is a completely unrelated module that implements the Map API. ```js const Keyv = require('keyv'); const QuickLRU = require('quick-lru'); const lru = new QuickLRU({ maxSize: 1000 }); const keyv = new Keyv({ store: lru }); ``` The following are third-party storage adapters compatible with Keyv: - [quick-lru](https://github.com/sindresorhus/quick-lru) - Simple "Least Recently Used" (LRU) cache - [keyv-file](https://github.com/zaaack/keyv-file) - File system storage adapter for Keyv - [keyv-dynamodb](https://www.npmjs.com/package/keyv-dynamodb) - DynamoDB storage adapter for Keyv ## Add Cache Support to your Module Keyv is designed to be easily embedded into other modules to add cache support. The recommended pattern is to expose a `cache` option in your modules options which is passed through to Keyv. Caching will work in memory by default and users have the option to also install a Keyv storage adapter and pass in a connection string, or any other storage that implements the `Map` API. You should also set a namespace for your module so you can safely call `.clear()` without clearing unrelated app data. Inside your module: ```js class AwesomeModule { constructor(opts) { this.cache = new Keyv({ uri: typeof opts.cache === 'string' && opts.cache, store: typeof opts.cache !== 'string' && opts.cache, namespace: 'awesome-module' }); } } ``` Now it can be consumed like this: ```js const AwesomeModule = require('awesome-module'); // Caches stuff in memory by default const awesomeModule = new AwesomeModule(); // After npm install --save keyv-redis const awesomeModule = new AwesomeModule({ cache: 'redis://localhost' }); // Some third-party module that implements the Map API const awesomeModule = new AwesomeModule({ cache: some3rdPartyStore }); ``` ## API ### new Keyv([uri], [options]) Returns a new Keyv instance. The Keyv instance is also an `EventEmitter` that will emit an `'error'` event if the storage adapter connection fails. ### uri Type: `String`

Default: `undefined` The connection string URI. Merged into the options object as options.uri. ### options Type: `Object` The options object is also passed through to the storage adapter. Check your storage adapter docs for any extra options. #### options.namespace Type: `String`

Default: `'keyv'` Namespace for the current instance. #### options.ttl Type: `Number`

Default: `undefined` Default TTL. Can be overridden by specififying a TTL on `.set()`. #### options.serialize Type: `Function`

Default: `JSONB.stringify` A custom serialization function. #### options.deserialize Type: `Function`

Default: `JSONB.parse` A custom deserialization function. #### options.store Type: `Storage adapter instance`

Default: `new Map()` The storage adapter instance to be used by Keyv. #### options.adapter Type: `String`

Default: `undefined` Specify an adapter to use. e.g `'redis'` or `'mongodb'`. ### Instance Keys must always be strings. Values can be of any type. #### .set(key, value, [ttl]) Set a value. By default keys are persistent. You can set an expiry TTL in milliseconds. Returns `true`. #### .get(key) Returns the value. #### .delete(key) Deletes an entry. Returns `true` if the key existed, `false` if not. #### .clear() Delete all entries in the current namespace. Returns `undefined`. ## License MIT © Luke Childs