Epkg User Manual

for version 3.3.3-git

Jonas Bernoulli

Table of Contents

1 Introduction

Epkg is a package that provides access to a local copy of the Emacsmirror package database. It provides low-level functions for querying the database and a `package.el`-like user interface for browsing the database.

Epkg itself is not a package manager, but the closely related Borg[1] package manager makes use of it.

The Emacsmirror is a growing collection of Emacs Lisp packages. All mirrored packages are available as Git repositories. In most cases this is done by mirroring the upstream Git repository, but if upstream uses something else, then the mirror nevertheless makes the package available as a Git repository.

One primary purpose of the Emacsmirror is to provide a comprehensive list of available Emacs packages, including packages which have gone out of fashion (but might later prove to be useful still).

Older efforts attempting to provide a comprehensive list of available packages, such as the Emacs Lisp List, over time collected an impressive list of dead links to packages which were no longer available anywhere.

With the Emacsmirror this won't happen. If a package's upstream disappears, then a copy remains available on the mirror. Once its upstream has disappeared a package is usually moved from the Emacsmirror to the Emacsattic, where it is no longer updated. (The Emacsattic is a Github "organization" separate from the Emacsmirror organization, but it is considered part of the Emacsmirror project.)

For more information about the Emacsmirror visit its homepage[2] and the blog post in which the current incarnation was announced[3].

[1] `https://emacsair.me/2016/05/17/assimilate-emacs-packages-as-git-submodules`

[2] `https://emacsmirror.net`

[3] `https://emacsair.me/2016/04/16/re-introducing-the-emacsmirror`

2 Installation

Epkg is available from Melpa and Melpa-Stable. To install it and its dependencies run `M-x package-install RET epkg RET`.

The Epkg database is stored in an SQLite database, which it accesses using the EmacSQL package.

Because the command line tool that comes with SQLite is unreliable, EmacSQL uses its own binary. By default that binary is compiled every time EmacSQL is updated, and if that fails, then EmacSQL asks whether you want to download a pre-build binary.

The SQLite database file is stored in a Git repository. If Epkg cannot find your local clone of that repository, then it offers to clone it to the location specified by the option `epkg-repository`. It isn't necessary but preferable to clone the repository manually before loading `epkg`.

```
git clone https://github.com/emacsmirror/epkgs.git ~/.emacs.d/epkgs
```

If you cloned the repository to a different location, then you have to set the value of `epkg-repository` accordingly. Add the following to your init file and don't forget to evaluate that form so that it also takes effect in the current session. To do so place the cursor after the closing parentheses and type `C-M-x`.

```
(setq epkg-repository "/path/to/epkgs/")
```

`epkg-repository` [User Option]

This option specifies the location of the local Emacsmirror repository.

This repository contains the Epkg SQLite database and, if they have been initialized, all package repositories from the Emacsmirror and Emacsattic as submodules.

If you change the value of this option, then you should also manually move the repository. Otherwise it would be cloned again.

The local clone of the Epkg repository is not updated automatically, so you should periodically use `M-x epkg-update RET` to update the database.

`epkg-update` [Command]

This command updates the Epkg database by pulling the `master` branch in the `epkg-repository` and then reloading the Epkg database. It returns the database connection.

3 Listing Packages

Epkg provides several commands for listing packages.

In the buffer which lists packages, typing `RET` displays information about the package at point in another buffer.

`epkg-list-exclude-types` [User Option]
: The value of this option is a list of package types. Most commands that list packages exclude any package whose type matches one of the types listed here. The command `epkg-list-packages-of-type` does not respect this option, and you can tell the other commands to ignore it as well by using a prefix argument.

`epkg-list-columns` [User Option]
: This option lists the columns used in buffers that list packages.

  Each element has the form `(HEADER WIDTH SORT PROPS SLOT FORMAT)`.

  - HEADER is the string displayed in the header.
  - WIDTH is the width of the column.
  - SORT is a boolean or a function. If it is t, then the column can be sorted alphanumerically, if it is nil then it can not. If it is a function then that is used as `sort`'s PREDICATE.
  - PROPS is an alist, supported keys are `:right-align` and `:pad-right`.
  - SLOT is an Epkg object slot or `type`.
  - FORMAT is a function, which is called with one argument, the slot value, and has to return a representation of that. If FORMAT is `nil`, then the value is inserted as-is.

  If an elements SLOT is `downloads`, then the respective SORT should be `epkg-list-sort-by-downloads`. If an elements SLOT is `stars`, then the respective SORT should be `epkg-list-sort-by-stars`.

`epkg-list-mode-hook` [User Option]
: This hook is run after entering Epkg-List mode, the mode used in buffers that list packages.

`epkg-list-packages` [Command]
: This command displays a list of all mirrored (and possibly also shelved) packages.

`epkg-list-matching-packages` [Command]
: This command displays a list of packages whose name or summary matches a SQLite LIKE pattern, which is read in the minibuffer.

`epkg-list-keyworded-packages` [Command]
: This command displays a list of packages that have a keyword set, which is read in the minibuffer.

  Only keywords that are members of `finder-known-keywords` are offered as completion candidates, but you can also enter other keywords.

`epkg-list-packages-by-author` [Command]
: This command displays a list of packages which are authored or maintained by a person. The person, a name or email address, is read in the minibuffer.

By default all of the above commands omit shelved packages from their output. With a prefix argument or when `epkg-list-packages-omit-shelved` is `nil`, then they don't omit any packages. However the following command ignores this option and always lists shelved packages when appropriate.

`epkg-list-packages-of-type` [Command]
: This command displays a list of packages of a certain type. The type is read in the minibuffer. To list all packages of a certain type and its subtypes use `TYPE*` instead of just `TYPE`.

4 Describing a Package

To display details about a single package in a buffer use the command `epkg-describe-package`. In buffers which list packages `RET` is bound to `epkg-list-describe-package`, which displays the package at point in another buffer.

By default the description buffer shows a tree of the packages the described package depends on. Click on the symbol before the package name to expand the node to show the dependencies of that dependency.

The first column lists the names of package that provide the feature(s) in the third column. The second column shows the type of the package in the first column.

The features in the third column are displayed in bold or using the regular font weight to indicate whether it is a hard (mandatory) or soft (optional) dependency.

Note that dependencies are determined automatically and even when a feature is shown using a bold face it might actually be optional. This could for example be the case when a feature is only required by one library that isn't required by any of the other libraries of the package it belongs to. Or a feature might even only be required by a single command, and the respective `require` form is only evaluated when that command is called.

Reverse dependencies are also displayed in a second tree. Here the first column lists the names of packages which depend on features from the described package and the third column shows which of these libraries are required.

`epkg-describe-package` [Command]

This command displays information about a package in a separate buffer. The name of the package to be displayed is read in the minibuffer.

`epkg-list-describe-package` [Command]

This command displays information about the package at point in a separate buffer.

It is only intended to be used in buffers which list packages. In other buffers, or in a list buffer when you want to display a package other than the one at point use `epkg-describe-package`.

`epkg-describe-package-slots` [User Option]

The value of this option is a list of slots to be displayed when displaying information about an Epkg package in a help buffer.

Each element of the list can be a function, a slot symbol, or `nil`. Functions are called with one argument, the Epkg object, and should insert a representation of the value at point. Raw slot symbols cause its non-nil value to be inserted as-is. If a slot's value is `nil`, then nothing is inserted. Elements that are `nil` stand for empty lines.

`epkg-describe-package-slots-width` [User Option]

The value of this option specifies the width used to display slot names in buffers displaying information about an Epkg package.

5 Package Types

Each package has a **type**, which specifies how the package is distributed and mirrored.

Packages are implemented using the Eieio object system (more or less in implementation of CLOS). A TYPE corresponds to the class `epkg-TYPE-package`. The `epkg` package makes little use of methods, but `emir`, the package used to maintain the Emacsmirror, makes extensive use of them. There exist five abstract classes (there are no instances of abstract classes, only of its subclasses): `epkg-package`, `epkg-mirrored-package`, `epkg-gitish-package`, `epkg-subset-package`, and `epkg-mocking-package`. Except for the second these classes are mostly an implementation detail and not relevant when merely using Epkg to browse the packages.

- `mirrored`

  This is an abstract type. Unlike other abstract types it is also useful on the client side, e.g., when you want to list mirrored packages, but not built-in and shelved packages.

  Packages that are available as a repository on the Emacsmirror (`https://github.com/emacsmirror`).

  - `file`

    Packages that are distributed as plain files.

  - `gitish`

    This is an abstract type, useful when maintaining the mirror.

    Git and Mercurial packages. The name is due to an implementation detail: `hg` is never run directly, instead `git-remote-hg` is used.

    - `git`

      Git packages.

      - `github`

        Packages hosted on `https://github.com`.

        - `orphaned`

          Packages that are no longer maintained, but which still have to be mirrored because other packages depend on them. Please consider adopting an orphaned package.

      - `gitlab`

        Packages hosted on `https://gitlab.com`.

      - `subtree`

        Packages that are located in a subtree of a Git repository. The repository on the Emacsmirror limits the history to just that directory using `git subtree`.

      - `subset`

        This is an abstract type, useful when maintaining the mirror.

        - `wiki`

          Packages hosted as plain files on `https://emacswiki.org`.

        - `elpa`

          Packages hosted in a directory inside the `master` branch of the GNU Elpa repository. These package are available from `https://elpa.gnu.org`.
        - `elpa-branch`

          Packages hosted in the GNU Elpa repository, using a dedicated branch. These package are available from `https://elpa.gnu.org`.
    - `hg`

      Mercurial packages.
        - `bitbucket`

          Packages hosted on `https://bitbucket.org` in a Mercurial repository. Packages hosted in a Git repository on Bitbucket have the type `git`.
- `mocking`

  This is an abstract type, useful when maintaining the mirror.

  Packages that are *not* available as a repository on the Emacsmirror (`https://github.com/emacsmirror`).
    - `builtin`

      Packages that are part of the latest stable GNU Emacs releases. `emacs` is one of the packages that are "part of Emacs"; it contains all libraries that are not explicitly declared to be part of some other built-in package.
    - `shelved`

      Packages that are available as a repository on the Emacsattic (`https://github.com/emacsattic`).

      These repository are not being updated anymore, because upstream has disappeared or because the package has issues which have to be resolved before it can be moved back to the Emacsmirror.

6 Using Epkg Objects

Most users won't have to access the Epkg objects directly and can just use the commands described in the preceding sections, but if you would like to extend Epkg, then you should know about the following functions.

Epkg objects are implemented using Eieio, which more or less is an implementation of CLOS. It's useful to learn about that, but to get started you may just use `oref` to obtain information about a package, e.g., `(oref (epkg "magit") url)`.

`epkg` *name* [Function]
: This function returns an `epkg-package` object for the package named NAME. NAME is the name of a package, a string.

`epkgs` **&optional** *select predicates* [Function]
: This function returns a list of `epkg-package` objects, column values or database rows. The list is ordered by the package names in ascending order.

  If optional SELECT is non-nil, then it has to be symbol naming a column in the 'packages' table or a vector of such columns. In those cases the returned value is a list of column values or a list of database rows. If SELECT is nil, return a list of objects.

  If optional TYPES is non-nil, then it has to be a vector of package types, such as `github`. To include subtypes, add an asterisk to the symbol name, e.g., `mirrored*`. For backward compatibility, TYPES can also be a list of predicate functions `epkg-TYPE-package-p` or `epkg-TYPE-package--eieio-childp`, or a single such function.

  This function is more limited than `epkg-sql` but it's often much less verbose. For example `(epkgs nil [gitlab])` returns the same value as:

```
(mapcar (apply-partially #'closql--remake-instance
                         'epkg-package (epkg-db))
        (epkg-sql [:select * :from packages
                   :where class :in $v1
                   :order-by [(asc name)]]
                  (closql-where-class-in 'gitlab)))
```

While it is possible to get a list of provided or required features, or a package's type using `oref`, the values of these slots contains additional information, which is mostly useful when maintaining the Emacsmirror, but not in a client. And the `required` slot only lists features but not the packages that provide them. The following functions return these values in a form that is generally more useful.

`epkg-provided` *package* **&optional** *include-bundled* [Function]
: This function returns a list of features provided by the package PACKAGE. PACKAGE is an `epkg-package` object or a package name, a string.

  Bundled features are excluded from the returned list unless optional INCLUDE-BUNDLED is non-nil.

`epkg-required` *package* [Function]
This function returns a list of packages and features required by the package PACKAGE. PACKAGE is an `epkg-package` object or a package name, a string.

Each element has the form `(DEPENDENCY FEATURES)`, where DEPENDENCY is the name of a required package, a string, and FEATURES is a list of features provided by DEPENDENCY and required by PACKAGE.

If a feature is represented using a symbol, then that indicates that it is a mandatory dependency; if a string is used, then it is an optional dependency.

There may be a single element `(nil FEATURES)`, which means that it is unknown which package or packages provide the feature or features listed in FEATURES.

`epkg-provided-by` *feature* [Function]
Return the name of the package provided by FEATURE. FEATURE has to be a symbol.

`epkg-reverse-dependencies` *package* [Function]
This function returns a list of packages that depend on PACKAGE. PACKAGE is an `epkg-package` object or a package name, a string.

Each element has the form `(DEPENDANT FEATURES)`, where DEPENDANT is the name of a package that depends on PACKAGE, a string, and FEATURES is a list of features provided by PACKAGE and required by DEPENDANT.

If a feature is represented using a symbol, then that indicates that it is a mandatory dependency; if a string is used, then it is an optional dependency.

`epkg-type` *arg* [Function]
This function returns the type of the object or class ARG.

ARG has to be the class `epkg-package`, a subclass of that, an `epkg-package` object, or an object of a subclass. The type represents the class and is used in the user interface, where it would be inconvenient to instead use the actual class name, because the latter is longer and an implementation detail.

`epkg-package-types` *subtypes* [Function]
This function returns a list of all package types.

If optional SUBTYPES is non-nil, then it also returns symbols of the form `TYPE*`, which stands for `"TYPE` and its subtypes`"`.

`epkg-read-type` *prompt* **&optional** *default subtypes* [Function]
This function reads an Epkg type in the minibuffer and returns it as a symbol.

If optional DEFAULT is non-nil, then that is offered as default choice. If optional CHILDP is non-nil, then entries of the form `TYPE*`, which stands for `"TYPE` and its subtypes`"`, are also offered as completion candidates.

`epkg-read-package` *prompt* **&optional** *default* [Function]
This function reads the name of an Epkg package in the minibuffer and returns it as a string.

Optional DEFAULT, if non-nil, is offered as default choice.

7 Querying the Database

If you are more interested in information about all or a subset of mirrored packages, as opposed to individual packages, then you should query the database directly instead of using the functions `epkg` and `epkgs`.

This is usually much more efficient, but requires that you know a bit about SQL, specifically SQLite[1], and that you make yourself familiar with the syntax used by EmacSQL[2] to express SQL statements.

The statistics about the Emacsmirror and related package archives[3] for the most part use `epkg-sql`, you might find the tools[4] used to create those statistics useful when getting started with that function.

`epkg-db` [Function]

This function returns the connection to the Epkg database.

If the `epkg-repository`, which contains the SQLite database file, does not exist yet, then this function first asks the user whether they want to clone the repository.

`epkg-sql` *sql* **&rest** *args* [Function]

This function sends the SQL S-expression to the Epkg database and returns the result. This is a wrapper around `emacsql` that lacks the CONNECTION argument. Instead it uses the connection returned by `epkg-db`.

[1] `https://sqlite.org/lang.html`

[2] `https://github.com/skeeto/emacsql`

[3] `https://emacsmirror.net/stats`

[4] `https://github.com/emacsmirror/epkg-reports`

Appendix A Function and Command Index

Appendix B Variable Index