Café Guix - 27 février 2024

Comment faire un paquet Guix avec l'outil d'aide Guix Packager ?

Philippe Virouleau et Alice Brenon

Outline

Inria

01

Génèse et (non) fonctionnalités

Inria

Problématiques personnelles

- Développement C/C++
- Création de paquet
- Nouvel utilisateur de Guix

Obstacles

- Transposer les concepts de créations de paquets
- (nouveau (language))

Solutions approximatives

- Réutilisation de codes existant
- "Salut Ludo"

guix import

- Excellent pour des exemples simples (cf python/haskell)
- Manque d'aspect "autonome", manque de configuration
- Dépendances manuelles, CMake ?

=> Interface utilisateur pour une configuration basique de paquet

Fonctionnalités

- Recette valide à partir de métadonnées
- Utilisation d'idiomes "recommandés"
- Configuration supplémentaires "basique" :
  - source (url, git, pypi, hackage)
  - système de build et configuration
  - ajout de dépendances → *use-module*
  - aide pour le calcul des hash
- Quelques fonctionnalités bonus (eg: openmpi)

Non fonctionalités

Pas un éditeur complet

C/C++

Un hello-world configurable

Python/Haskell

- html5-entity → ghc-html5-entity
- gpxpy → python-gpxpy

Les fichiers en détail

```
(define-module (guix-packager)
  #:use-module (guix)
  #:use-module ((guix licenses) #:prefix license:)
  #:use-module (gnu packages)
  #:use-module (guix build-system haskell))

(define-public ghc-html5-entity
  (package
    (name "ghc-html5-entity")
    (version "0.2.0.3")
    (source
      (origin
        (method url-fetch)
        (uri (hackage-uri "html5-entity" version))
        (sha256 (base32 "0bmmzshxanzw5y2y0hvgzz9yw18jqgv535i1xq2a5lf7w8wpj1if"))))
    (build-system haskell-build-system)
    (arguments (list
                    #:tests? #t))
    (home-page "https://github.com/zudov/html5-entity/")
    (synopsis "A library for looking up and validating HTML5 entities")
    (description "This package provides a library for looking up and validating HTML5 entities.
The <http://html.spec.whatwg.org/multipage/entities.json following> document is
used as an authoritative source of the valid entity names and their corresponding
codepoints.  You can think of this library as about bindings to
the data from that file.  For usage see the Text.Html5.Entity module.")
    (license license:bsd-3)))

;; This allows you to run guix shell -f example.scm.
;; Remove this line if you just want to define a package.
ghc-html5-entity
```


Inria

Les fichiers en détail

```
(define-module (guix-packager)
  #:use-module (guix)
  #:use-module ((guix licenses) #:prefix license:)
  #:use-module (gnu packages)
  #:use-module (guix build-system haskell))

(define-public ghc-html5-entity
  (package
    (name "ghc-html5-entity")
    (version "0.2.0.3")
    (source
      (origin
        (method url-fetch)
        (uri (hackage-uri "html5-entity" version))
        (sha256 (base32 "0bmmzshxanzw5y2y0hvgzz9yw18jqgv535i1xq2a5lf7w8wpj1if"))))
    (build-system haskell-build-system)
    (arguments (list
                    #:tests? #t))
    (home-page "https://github.com/zudov/html5-entity/")
    (synopsis "A library for looking up and validating HTML5 entities")
    (description "This package provides a library for looking up and validating HTML5 entities.
The <http://html.spec.whatwg.org/multipage/entities.json following> document is
used as an authoritative source of the valid entity names and their corresponding
codepoints.  You can think of this library as about bindings to
the data from that file.  For usage see the Text.Html5.Entity module.")
    (license license:bsd-3)))

;; This allows you to run guix shell -f example.scm.
;; Remove this line if you just want to define a package.
ghc-html5-entity
```


Inria

Les fichiers en détail

```
(define-module (guix-packager)
  #:use-module (guix)
  #:use-module ((guix licenses) #:prefix license:)
  #:use-module (gnu packages)
  #:use-module (guix build-system haskell))

(define-public ghc-html5-entity
  (package
    (name "ghc-html5-entity")
    (version "0.2.0.3")
    (source
      (origin
        (method url-fetch)
        (uri (hackage-uri "html5-entity" version))
        (sha256 (base32 "0bmmzshxanzw5y2y0hvgzz9yw18jqgv535i1xq2a5lf7w8wpj1if"))))
    (build-system haskell-build-system)
    (arguments (list
                    #:tests? #t))
    (home-page "https://github.com/zudov/html5-entity/")
    (synopsis "A library for looking up and validating HTML5 entities")
    (description "This package provides a library for looking up and validating HTML5 entities.
The <http://html.spec.whatwg.org/multipage/entities.json following> document is
used as an authoritative source of the valid entity names and their corresponding
codepoints.  You can think of this library as about bindings to
the data from that file.  For usage see the Text.Html5.Entity module.")
    (license license:bsd-3)))

;; This allows you to run guix shell -f example.scm.
;; Remove this line if you just want to define a package.
ghc-html5-entity
```


Inria

Les fichiers en détail

```
(define-module (guix-packager)
  #:use-module (guix)
  #:use-module ((guix licenses) #:prefix license:)
  #:use-module (gnu packages)
  #:use-module (guix build-system haskell))

(define-public ghc-html5-entity
  (package
    (name "ghc-html5-entity")
    (version "0.2.0.3")
    (source
      (origin
        (method url-fetch)
        (uri (hackage-uri "html5-entity" version))
        (sha256 (base32 "0bmmzshxanzw5y2y0hvgzz9yw18jqgv535i1xq2a5lf7w8wpj1if"))))
    (build-system haskell-build-system)
    (arguments (list
                    #:tests? #t))
    (home-page "https://github.com/zudov/html5-entity/")
    (synopsis "A library for looking up and validating HTML5 entities")
    (description "This package provides a library for looking up and validating HTML5 entities.
The <http://html.spec.whatwg.org/multipage/entities.json following> document is
used as an authoritative source of the valid entity names and their corresponding
codepoints.  You can think of this library as about bindings to
the data from that file.  For usage see the Text.Html5.Entity module.")
    (license license:bsd-3)))

;; This allows you to run guix shell -f example.scm.
;; Remove this line if you just want to define a package.
ghc-html5-entity
```


Inria

Les fichiers en détail

```
(define-module (guix-packager)
  #:use-module (guix)
  #:use-module ((guix licenses) #:prefix license:)
  #:use-module (gnu packages)
  #:use-module (guix build-system haskell))

(define-public ghc-html5-entity
  (package
    (name "ghc-html5-entity")
    (version "0.2.0.3")
    (source
      (origin
        (method url-fetch)
        (uri (hackage-uri "html5-entity" version))
        (sha256 (base32 "0bmmzshxanzw5y2y0hvgzz9yw18jqgv535i1xq2a5lf7w8wpj1if"))))
    (build-system haskell-build-system)
    (arguments (list
                    #:tests? #t))
    (home-page "https://github.com/zudov/html5-entity/")
    (synopsis "A library for looking up and validating HTML5 entities")
    (description "This package provides a library for looking up and validating HTML5 entities.
The <http://html.spec.whatwg.org/multipage/entities.json following> document is
used as an authoritative source of the valid entity names and their corresponding
codepoints.  You can think of this library as about bindings to
the data from that file.  For usage see the Text.Html5.Entity module.")
    (license license:bsd-3)))

;; This allows you to run guix shell -f example.scm.
;; Remove this line if you just want to define a package.
ghc-html5-entity
```


Inria

Des valeurs comme les autres

Guile n'est pas (juste) un langage de configuration

- `(define-public ...)`
- `ghc-html5-entity`

Des valeurs comme les autres

Guile n'est pas (juste) un langage de configuration

- `(define-public ...)`
- `ghc-html5-entity`

```
1 ghc_html5_entity = Package(
2   name="ghc_html5_entity",
3   ...)
```


Inria

Des valeurs comme les autres

Guile n'est pas (juste) un langage de configuration

- `(define-public ...)`
- `ghc-html5-entity`

```
1 ghc_html5_entity = Package(
2   name="ghc_html5_entity",
3   ...)
```

```
1 return ghc_html5_entity
```

(retourner la valeur)

Et les modules alors ?

`(define-module ...)`

- ≪ 27,347 packages ≫ `https://packages.guix.gnu.org/`
- → pas un seul package par fichier !

Ne renvoie rien, juste une collection de déclarations

```
1  (define-public a-package
2    (package ...))
3
4  (define-public another
5    (package ...))
6
7  (define-public yet-another
8    (package ...))
```

Correspondance chemin symbolique / système de fichiers

`(define-module (a b c))` ↔ `a/b/c.scm`

Inria

`ghc-html5-entity` vs. `"ghc-html5-entity"`

Le paquet en tant que valeur guile

```
1 ...
2 (propagated-inputs (list sed))
3 (inputs (list ghc-attoparsec
4               ghc-geode
5               ghc-html5-entity
6               ghc-optparse-applicative))
7 ...
```

mais dans le shell

- `guix search...`
- `guix build...`
- `guix shell...`

< la valeur de name >

Inria

En résumé

	type d'objet	avec les imports
Guix packager	module & valeur	oui
`guix import`	valeur	non
(`guix edit`	module	oui)

- module: `-L` ≪ `DOSSIER` ≫ / `GUIX_PACKAGE_PATH`
- valeur: `-f` ≪ `FICHIER` ≫