Augeas: Créer un module lens étape par étape

Table of Contents

Présentation des Lenses

Les lenses sont les blocs de construction de base pour établir la cartographie à partir de fichiers dans l'arbre d'Augeas. On peut considérer une lens comme un enregistrement de trois fonctions get, put et create, où la fonction get prend le contenu d'un fichier texte, l'analyse et produit une partie de l'arbre d'Augeas. Les fonctions put et create prennent un arbre et le transforment en un fichier texte. La différence est que put est utilisé lorsque la partie de l'arbre est transformée, et la fonction create est utilisée quand elle ne l'est pas.

Présentation du fichier de configuration de test

Cette page est un guide pour vous aider à créer un lens Augeas. Dans cet exemple, nous allons créer une lens pour le fichier apt/preferences, qui ressemble à ceci:

```
Explanation: Backport packages are never prioritary
Package: *
Pin: release a=backports
Pin-Priority: 100

Explanation: My packages are the most prioritary
Package: *
Pin: release l=Raphink, version=3.0
Pin-Priority: 700
```

Les principales spécificités de ce format sont:

- Les commentaires ne sont pas autorisés, sauf dans les champs Explication
- Les enregistrements sont séparés par une ou plusieurs lignes vides
- Les champs sont des paires clé / valeur séparées par “:”

Présentation du fichier de test Augeas

L'objectif est de pouvoir analyser correctement un fichier apt/preferences avec un fichier de test. Ce fichier de test sera enregistré sous /usr/share/augeas/lenses/tests/test_aptpreferences.aug. Cela permettra de faire un développement piloté par les tests, dès que notre objectif sera compilé.

Un fichier de test Augeas est un module Augeas qui utilise un lens existant pour le tester par rapport à une chaîne de configuration donnée :

```
module Test_aptpreferences =

let conf ="Explanation: Backport packages are never prioritary
Package: *
Pin: release a=backports
Pin-Priority: 100

Explanation: My packages are the most prioritary
Package: *
Pin: release l=Raphink, v=3.0
Pin-Priority: 700
"

 test AptPreferences.lns get conf =
   { "1"
        { "Explanation"  = "Backport packages are never prioritary" }
        { "Package"      = "*" }
        { "Pin"          = "release"
```

```
        { "a" = "backports" } }
        { "Pin-Priority" = "100" } }
  {}
  { "2"
     { "Explanation"  = "My packages are the most prioritary" }
     { "Package"      = "*" }
     { "Pin"          = "release"
         { "l" = "Raphink" }
         { "v" = "3.0"     } }
     { "Pin-Priority" = "700" } }
```

C'est l'étape par laquelle on décrit la manière dont est traduit votre fichier dans l'arbre Augeas. Dans ce cas, les enregistrements sont représentés en tant que séquences. Ce choix est valable ici parce que le fichier n'a que des enregistrements ou des lignes vides, et rien d'autre, donc les enregistrements peuvent être simplement énumérés numériquement sous le nœud racine du fichier. À l'intérieur de chaque nœud, chaque champ principal sera un nœud avec une valeur. Dans le cas du nœud “Pin”, un ou de plusieurs sous-nœuds pour stocker les tests (par exemple a=backports ou l=Raphink) devront être définis.

Les nouvelles lignes sont importantes dans la chaîne de configuration! Dans cet exemple, il n'y a pas de nouvelle ligne au début du fichier, et une seule nouvelle ligne est permise à la fin.

Le “{}” entre les deux enregistrements; est nécessaire car Augeas prendra en compte la ligne vide, même si elle ne l'affiche pas.

Il faut fermer les parenthèses lorsqu'on spécifie les nœuds!

Si on lanne **augparse** sur un fichier de configuration de test:

```
$ augparse tests/test_aptpreferences.aug
```

```
  tests/test_aptpreferences.aug:14.9-.27:Could not load module
AptPreferences for Aptpreferences.lns
  tests/test_aptpreferences.aug:14.9-.27:Undefined variable
AptPreferences.lns
  tests/test_aptpreferences.aug: error: Loading failed
```

Cela ne marche pas, puisque le lens AptPreferences.aug n'existe pas encore! Passons maintenant au développement du lens

Etape 1: Ecriture du module lens

Définition le squelette du module (layout)

Pour commencer, définir d'abord le squelette du module lens. Le fichier de lens sera enregistré sous /usr/share/augeas/lenses/aptpreferences.aug:

```
(* Apt/preferences module for Augeas          *)
module AptPreferences =
autoload xfm

let lns =

let filter = incl "/etc/apt/preferences"
           . Util.stdexcl

let xfm = transform lns filter
```

- **module AptPreferences = autoload xfm** L'instruction autoload doit être la toute première instruction à l'intérieur du module.
- **let xfm = transform lns filter**: Pour augtool, il faut également indiquer quels fichiers doivent être transformés avec quel lens. Ceci est fait en utilisant la fonction de transformation qui prend un lens et un filtre de fichier. Quand augtool démarre, il applique toutes les transformations marquées pour le chargement automatique, en recherchant tous les fichiers correspondant au filtre de fichiers et en appliquant le lens correspondant à leur contenu.L'arbre résultant de l'application de la lens au contenu d'un fichier avec chemin absolu `/PATH` est placé dans l'arbre d'Augeas sous `/files/PATH`.
- **let filter = incl "/etc/apt/preferences". Util.stdexcl**: les filtres de fichiers sont construits en concaténant les applications des fonctions intégrées incl et excl, les deux prennent un shell comme unique argument.Ce filtre correspond au seul fichier /etc/apt/preferences
  D'autres syntaxes permettent de concatener plusieurs fichiers, par exemple **let filter = (incl "/etc/pam.d/*") . (excl "*.rpmsave")** correspond à tous les fichiers de /etc/pam.d qui ne se terminent pas par .rpmsave. L'ordre d'incl et d'excl dans le filtre n'a pas d'importance - un fichier est inclus s'il est associé à au moins un glob inclus, et par aucun glob excl. Les globs qui ne contiennent pas de / sont comparés au nom de base d'un fichier, sinon ils sont comparés au nom de chemin complet du fichier.

Les commentaires sont sous la forme de (* quelque chose *).

Choix du sens de développement

Il y a deux façons principales d'apréhender le développement:

- de haut en bas: on peut commencer par décrire comment le fichier est constitué en général, puis entrer dans les détails au fur et à mesure,
- de bas en haut: on peut décrire d'abord toutes les petites primitives du fichier, puis construire les différents types d'enregistrements sur celles-ci jusqu'à constitution du fichier en entier.

Dans cet exemple, on a choisi la seconde option car les enregistrements sont faciles à définir.

Développement du module

Définition de la structure générale

La question à laquelle on doit répondre ici est: “`Comment est constitué le fichier apt/preferences?”. La réponse est assez simple: il est faite de d'enregistrements (record) et de lignes vides (empty), un nombre quelconque de fois:

```
(* Définition de la structure gébérale *)
let lns = ( record | empty )*
```

Cette définition signifie que le fichier peut commencer ou se terminer avec n'importe quel nombre de lignes vides.

Définition des lignes vides

La question à laquelle on doit répondre ici est: “Qu'est-ce qu'une ligne vide?” C'est une ligne qui contient seulement des espaces. Écrivons cela:

```
(* Définir la ligne vide *)
let empty = [ del /[ \t]*\n/ "" ]
```

Cela signifie qu'une ligne vide est constituée de n'importe quel nombre d'espaces, suivi d'une nouvelle ligne. “del” indique que tous ces caractères doivent être ignorés dans l'arbre, et “” indique que la valeur par défaut pour une ligne vide est “”.

Définition des enregistrements

La question à laquelle on doit répondre ici est: “Qu'est-ce qu'un enregistrement?” Un enregistrement est un ensemble d'entrées, rien de plus:

```
(* Définir l'enregistrement *)
let record = [ seq "record" . entries+ ]
```

Définition des entrées

Les entrées peuvent être des entrées de clé / valeur (“Explication”, “Package”, et “Pin-Priority”) ou des entrées spéciales (le cas de “Pin”). Écrivons-le de cette façon pour le moment:

```
let entries = explanation
        | package
        | pin_priority
        | pin
```

Définition des entrées simples

```
let explanation = [ key "Explanation" . Util.del_str ":" . store /[^\n]*/ .
Util.del_str "\n" ]
```

Une entrée explication commence par un mot “Explication”, qui est la clé, suivi d'un deux-points à ignorer, suivi de tout type de caractère sauf d'un saut de ligne, jusqu'à la fin de la ligne, qui doit être ignorée. “[” et “]” indiquent que l'explication est une lens qui va stocker un nœud dans l'arbre d'Augeas.

Toutes le entrées utiliseront 'Util.del*str “:”*','*store /[^\n]*/*' *et* '*Util.del*str “\n”', il est donc judicieux de définir des raccourcis pour ceux-ci.

```
(* Définition des primitives utiles *)
let colon        = Util.del_str ":"
let eol          = Util.del_str "\n"
let value_to_eol = store /[^\n]*/
```

Traitement des espaces: Util.del_str “:” est l'équivalent de del /:/ “:”, mais dans le cas étudié on peut avoir des espaces après le deux-points et on pourrait avoir des espaces entre la valeur et le retour à la ligne:

```
let colon        = del /:[ \t]*/ ": "
let eol          = del /[ \t]*\n/ "\n"
```

value*to*eol pourrait commencer ou se terminer par des espaces, ce qui serait en conflit avec les deux-points et eol. Il faut donc également modifier value*to*eol:

```
let value_to_eol = store /[^ \t\n][^\n]*[^ \t\n]/
```

Cela fonctionnera, mais seulement si ma valeur est d'au moins 2 caractères! Alors, voici une autre façon de l'écrire:

```
let value_to_eol = store ( /[^\n]*/ - /([ \t][^\n]*|[^\n]*[ \t])/ )
```

Ici, on écrit: value*to*eol est tout sauf un retour à la ligne, sauf si elle commence ou se termine par un espace ou une tabulation.

En fait, une bonne pratique ici est d'utiliser Util.eol pour la fin des lignes:

```
let eol = Util.eol
```

POur prendre en charge les nouvelles lignes de style DOS, utiliser Util.doseol:

```
let eol = Util.doseol
```

La description de l'entrée explication avec les primitives :

```
let explanation = [ key "Explanation" . colon . value_to_eol . eol ]
```

La description de l'entrée package avec les primitives :

```
let package = [ key "Package" . colon . value_to_eol . eol ]
```

C'est pareil! C'est là que les fonctions peuvent faciliter la vie. Erire une fonction simple_entry comme suit:

```
let simple_entry (kw:string) = [ key kw . colon . value_to_eol . eol ]
```

(kw: string) est une variable pour la fonction. kw est le nom de la variable et string est le type.

Maintenant la description des entrées explication et package ressemble à ceci:

```
let explanation = simple_entry "Explanation"
let package     = simple_entry "Package"
```

Ce n'est plus vraiment nécessaire de les éclarr individuellement on peut donc simplement redéfinir les entrées comme ceci:

```
let entries = simple_entry "Explanation" | simple_entry "Package" |
simple_entry "Pin-Priority" | pin
```

Définition des entrées spéciales

De quoi est faite l'entrée “Pin”? Elle est constitué d'une valeur, suivie d'un nombre pair quelconque de clé/valeur, séparées par “=” :

```
let pin = [ key "Pin" . colon . value_to_spc . Sep.space . key_value+ . eol
]
```

Sep.space est une définition utile du module Sep, qui est équivalente à del del /[\t]*/ “ ”. Il faut définir les primitives value*to*spc et key_value:

```
let value_to_spc = store Rx.no_spaces
let key_value    = [ key /[^= \/\t\n]+/ . Sep.equal . value_to_spc ]
```

où Rx.no_spaces est une définition du module Rx (regexps), et Sep.equal est tiré du module Sep.

chaque valeur_clé est une lens de sorte qu'il va créer un nouveau sous-nœud à l'intérieur du nœud `'Pin'` .

Présentation du module assemblé

Le module final ressemble à cala :

```
(* Apt/preferences module for Augeas          *)

module AptPreferences =
autoload xfm


(* Define useful primitives *)
let colon        = del /:[ \t]*/ ": "
let eol          = Util.eol
let value_to_eol = store ( /[^\n]*/ - /([ \t][^\n]*|[^\n]*[ \t])/ )
let value_to_spc = store Rx.no_spaces
let key_value    = [ key /[^= \t\n]+/ . Sep.equal . value_to_spc ]

(* Define empty *)
let empty = Util.empty

(* Define record *)

let simple_entry (kw:string) = [ key kw . colon . value_to_eol . eol ]
let pin = [ key "Pin" . colon . value_to_spc . Sep.space . key_value+ . eol
]

let entries = simple_entry "Explanation"
            | simple_entry "Package"
            | simple_entry "Pin-Priority"
            | pin

let record = [ seq "record" . entries+ ]

(* Define lens *)
let lns = ( record | empty )*

let filter = incl "/etc/apt/preferences"
           . Util.stdexcl
```

```
let xfm = transform lns filter
```

Etape 2: Déboggage du module

Compiler le module

Il y a de fortes chances qu'une première compilation échoue

```
 $ augparse aptpreferences.aug
aptpreferences.aug:18.3-.76:Failed to compile key_value
aptpreferences.aug:18.24-.40:exception: The key regexp /[^=
]+/ matches a '/'

aptpreferences.aug:28.3-31.13:Failed to compile entries
aptpreferences.aug:25.36-.71:exception: ambiguous concatenation
      'Explanation: \n' can be split into
      'Explanation:|=| \n'

     and
      'Explanation: |=|\n'

    First lens: aptpreferences.aug:25.36-.65
    Second lens: aptpreferences.aug:15.22-.41

aptpreferences.aug: error: Loading failed
```

Comprendre et réparer les erreurs

Retouche de la clé pin

La première erreur concerne la clé dans key_value, qui n'est pas assez précis. Un moyen de contourner est de le spécifier, car nous connaissons les valeurs qu'elle pourrait prendre:

```
let key_value (kw:string)    = [ key kw . Sep.equal . value_to_spc ]
let pin_key = key_value "a"
            | key_value "c"
            | key_value "l"
            | key_value "o"
            | key_value "v"

let pin = [ key "Pin" . colon . value_to_spc . Sep.space . pin_key+ . eol ]
```

De cette façon, on s'assure que la clé est l'une des valeurs spécifiées.

value_to_eol

La deuxième erreur. Augeas dit qu'un conffile avec 'Explication: \n' pourrait être interprété de deux façons, en utilisant les lens sur les lignes 15 ou 25.

La ligne 15 est:

```
 let eol = Util.eol
```

ce qui est équivalent à del /[\t]*\n/ “\n”.

La ligne 25 est:

```
let simple_entry (kw:string) = [ key kw . colon . value_to_eol . eol ]
```

L'erreur suggère que value*to*eol pourrait être un simple espace. Dans ce cas, Augeas ne saurait pas si l'espace appartient à value*to*eol ou à eol. Il semble que la définition de value*to*eol ne fonctionne pas vraiment. Un moyen facile serait de considérer que eol est après tout Util.del_str “\ n”, mais le format permettrait d'avoir des espaces en fin de ligne, donc ça pourrait arriver, et on ne veut pas avoir d'espace à l'intérieur des valeurs.

Au lieu de cela, on peut réécrire value*to*eol comme ceci :

```
let value_to_eol = store /([^ \t\n].*[^ \t\n]|[^ \t\n])/
```

ou utiliser la définition Rx.space_in, qui est équivalente:

```
let value_to_eol = store Rx.space_in
```

C'est le même [^\n], dans cette nouvelle expression rationnelle, on accepte soit une chaîne de 2 caractères minimum sans qu'un saut de ligne ne commence ou ne se termine par un espace ou une tabulation, ou un seul caractère qui n'est pas un espace, une tabulation ou un saut de ligne.

pin_key spaces

Si on réexécute l'analyseur, on obtiens une nouvelle erreur:

```
aptpreferences.aug:33.68-.77:exception: ambiguous iteration
      'v=\002\001\001' can be split into
      'v=\002\001|=|\001'

     and
      'v=\002\001|=|\001'
```

```
    Iterated lens: aptpreferences.aug:27.18-31.24
```

On a fait une erreur dans l'entrée pin à nouveau!

```
let pin = [ key "Pin" . colon . value_to_spc . Sep.space . pin_key+ . eol ]
```

Où sont les séparateurs entre les différentes entrées possibles de pin*key? Le format permet d'avoir plusieurs paires clé / valeur pin*key, séparées par une virgule. Alors il faut les prendre en compte :

```
let pin = [ key "Pin" . colon . value_to_spc . Sep.space . pin_key . ( del
/,[ \t]*/ ", " . pin_key )*. eol ]
```

Cependant l'erreur est toujours là, mais semble juste un peu différent:

```
aptpreferences.aug:27.79-.113:exception: ambiguous iteration
      ',v=\002\001\001\001' can be split into
      ',v=\002\001|=|\001\001'

     and
      ',v=\002\001|=|\001\001'

    Iterated lens: aptpreferences.aug:27.81-.110
```

Il est maintenant possible pour un value*to*spc de contenir un “,” donc Augeas ne sait pas si le “,” fait partie de value*to*spc ou s'il sépare les entrées pin*key, donc il faut modifier value*to_spc en:

```
let value_to_spc = store /[^, \t\n]+/
```

Maintenant on obtient:

```
aptpreferences.aug:37.13-.32:exception: ambiguous iteration
      'Pin-Priority:\001\nPin:\001\001\001\n\001' can be split into
      'Pin-Priority:\001\n|=|Pin:\001\001\001\n\001'

     and
      'Pin-Priority:\001\nPin:\001|=|\001\001\n\001'
```

On a oublié un autre point important! Les enregistrements sont séparés par des retours à la ligne. Un enregistrement est en fait:

```
let record = [ seq "record" . entries+ . eol ]
```

Etape 3:Test du module lens

Maintenant que le lens se compile sans erreur, on peut le tester.

Le module lens débogué

```
(* Apt/preferences module for Augeas          *)

module AptPreferences =
autoload xfm
(* Define useful primitives *)
let colon        = del /:[ \t]*/ ": "
let eol          = Util.eol
let value_to_eol = store Rx.space_in
let value_to_spc = store /[^, \t\n]+/

(* Define empty *)
let empty = Util.empty

(* Define record *)
let simple_entry (kw:string) = [ key kw . colon . value_to_eol . eol ]

let key_value (kw:string)    = [ key kw . Sep.equal . value_to_spc ]
let pin_key = key_value "a"
            | key_value "c"
            | key_value "l"
            | key_value "o"
            | key_value "v"

let pin = [ key "Pin" . colon . value_to_spc . Sep.space . pin_key . ( del
/,[ \t]*/ ", " . pin_key )*. eol ]

let entries = simple_entry "Explanation"
            | simple_entry "Package"
            | simple_entry "Pin-Priority"
            | pin

let record = [ seq "record" . entries+ . eol ]

(* Define lens *)
let lns = ( record | empty )*

let filter = incl "/etc/apt/preferences"
           . Util.stdexcl

let xfm = transform lns filter
```

Tester le module lens

```
$ augparse tests/test_aptpreferences.aug
Test run encountered exception:
tests/test_aptpreferences.aug:14.9-.36:exception: Short iteration
```

```
    Error encountered here (107 characters into string)
    <ackports\nPin-Priority: 100\n\n|=|Explanation: My packages are>

    Tree generated so far:
    /1
/1/Explanation = "Backport packages are never prioritary"
/1/Package = "*"
/1/Pin = "release"
/1/Pin/a = "backports"
/1/Pin-Priority = "100"


tests/test_aptpreferences.aug: error: Loading failed
```

Il y a encore quelque chose qui ne va pas! Si on reviens à la dernière correction faite, on notera qu'on a ajouté **eol** à la fin d'un enregistrement. Cela a fixé le lens, mais cela ne reflète pas la réalité. En réalité, le dernier enregistrement ne se termine pas nécessairement par une nouvelle ligne. De plus, la ligne qui sépare deux enregistrements n'est pas vraiment une eol, mais une ligne vide à la place.

Il faut donc prendre en compte ces règles :

```
let record = [ seq "record" . entries+ ]
let lns = empty* . ( record . empty )* . record?
```

L'enregistrement (record) lui-même ne se termine plus avec une eol. Au lieu de cela, le lns prend soin de la séparation. Un lns est maintenant un nombre optionnel de lignes vides au début, suivi d'une série optionnelle d'enregistrements et de lignes vides, éventuellement suivie d'un seul enregistrement, dans le cas où il n'y a pas de ligne vide à la fin du fichier.

Vérifier à nouveau la compilation:

```
$ augparse aptpreferences.aug
```

Et maintenant l'exécution:

```
$ augparse tests/test_aptpreferences.aug
```

Cela fonctionne!