N°194 juin 2016

GNU LINUX MAGAZINE / FRANCE

DÉVELOPPEMENT ET ADMINISTRATION SUR SYSTÈMES OPEN SOURCE

ACTUS / POSTGRESQL

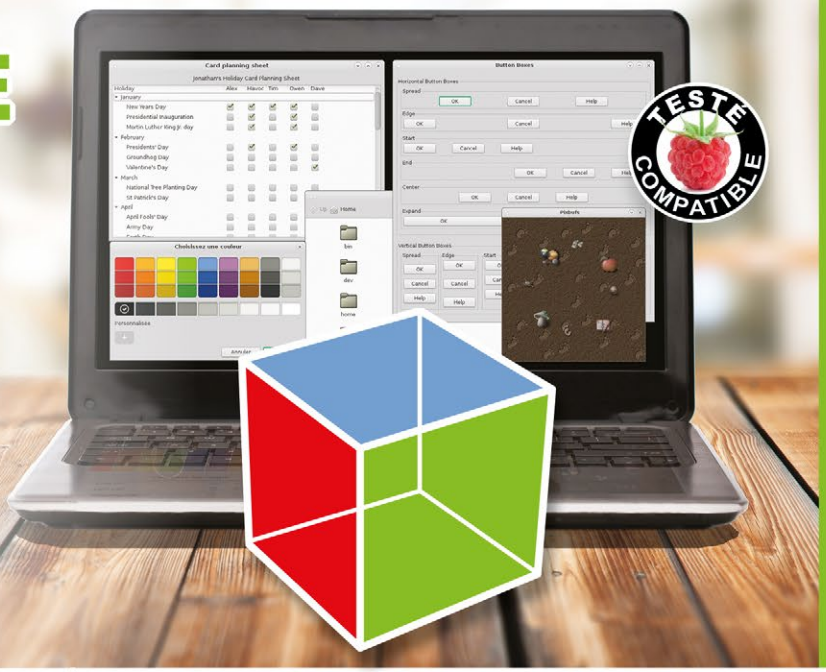
Panorama des nouvelles fonctionnalités SQL de PostgreSQL p.14

HOW-TO / LANGAGE C

CRÉEZ VOTRE PREMIÈRE APPLICATION GRAPHIQUE !

- Faites vos premiers pas avec le toolkit GTK+
- Définissez votre thème
- Internationalisez votre application

p.58



SYSADMIN / GNU GUIX

Gérez enfin vos paquets de manière sûre et flexible p.46

HUMEUR / ANALYSTES

Jean-Pierre Troll est de retour... et il n'est pas content ! p.18

JAVASCRIPT / JQUERY

(Re)Découvrez le framework phare du Web 2.0

p.74

PDO / PHP

Comprenez le comptage de références en PHP

p.52

C++/LISP

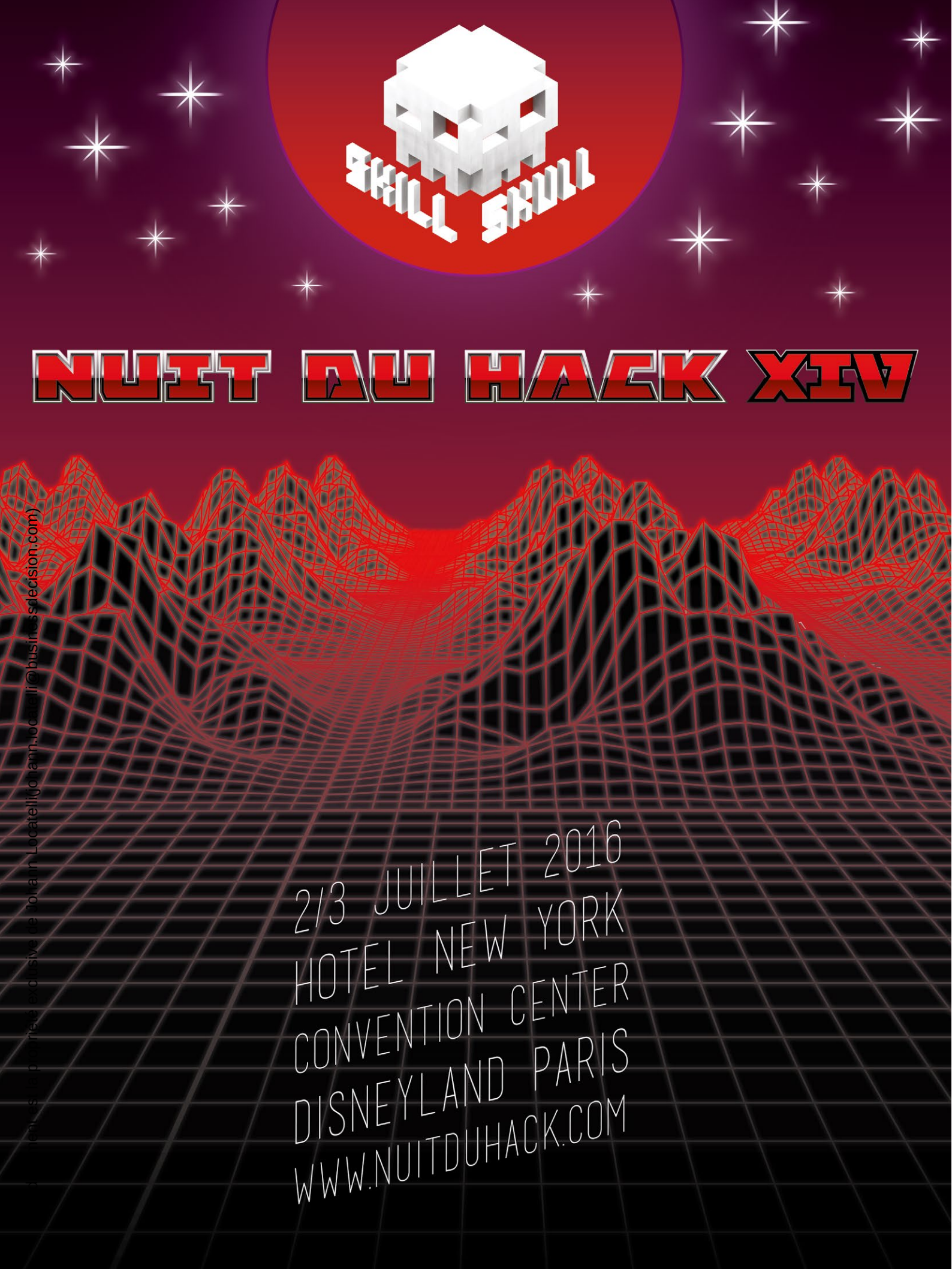
Interprétez les fonctions et les macros Lisp

p.66

ET AUSSI : Calculs sur les réels : les opérations – Perl 6 – Lire des QR Codes

L 19275 - 194 - F: 7,90 € - RD

France MÉTRO. : 7,90 € – CH : 13 CHF – BEL/LUX/PORT.CONT : 8,90 € – DOM/TOM : 8,50 € – CAN : 14 $ CAD



Toute reproduction même partielle sous quelque forme que ce soit est interdite sauf autorisation expresse de Jonathan Locatelli (jonathan.locatelli@businessdecision.com)

GNU/Linux Magazine France
est édité par Les Éditions Diamond

10, Place de la Cathédrale - 68000 Colmar - France
Tél. : 03 67 10 00 20 – **Fax :** 03 67 10 00 21
E-mail : lecteurs@gnulinuxmag.com
Service commercial : abo@gnulinuxmag.com
Sites : www.gnulinuxmag.com – www.ed-diamond.com

Directeur de publication : Arnaud Metzler
Chef des rédactions : Denis Bodor
Rédacteur en chef : Tristan Colombo
Responsable service Infographie : Kathrin Scali
Responsable publicité : Valérie Fréchard,
Tél. : 03 67 10 00 27 – v.frechard@ed-diamond.com
Service abonnement : Tél. : 03 67 10 00 20
Impression : pva, Druck und Medien-Dienstleistungen GmbH, Landau, Allemagne

Distribution France : (uniquement pour les dépositaires de presse)
MLP Réassort : Plate-forme de Saint-Barthélemy-d'Anjou.
Tél. : 02 41 27 53 12
Plate-forme de Saint-Quentin-Fallavier. Tél. : 04 74 82 63 04

IMPRIMÉ en Allemagne - PRINTED in Germany
Dépôt légal : À parution, **N° ISSN :** 1291-78 34
Commission paritaire : K78 976

Périodicité : Mensuel
Prix de vente : 7,90 €

La rédaction n'est pas responsable des textes, illustrations et photos qui lui sont communiqués par leurs auteurs. La reproduction totale ou partielle des articles publiés dans GNU/Linux Magazine France est interdite sans accord écrit de la société Les éditions Diamond. Sauf accord particulier, les manuscrits, photos et dessins adressés à GNU/Linux Magazine France, publiés ou non, ne sont ni rendus, ni renvoyés. Les indications de prix et d'adresses figurant dans les pages rédactionnelles sont données à titre d'information, sans aucun but publicitaire. Toutes les marques citées dans ce numéro sont déposées par leur propriétaire respectif. Tous les logos représentés dans le magazine sont la propriété de leur ayant droit respectif.

SUIVEZ-NOUS SUR :

https://www.facebook.com/editionsdiamond

@gnulinuxmag

LES ABONNEMENTS ET LES ANCIENS NUMÉROS SONT DISPONIBLES !

En version papier et PDF :
www.ed-diamond.com

Codes sources sur https://github.com/glmf

ÉDITORIAL

Faut-il faire table rase du passé ?

Ne vous inquiétez pas, je ne vous invite pas à la rédaction d'un devoir de philosophie, mais à une réflexion beaucoup plus prosaïque sur l'importance à donner à des technologies dites « dépassées ». L'informatique est un domaine où tout évolue très vite, du matériel aux langages. Faut-il pour autant considérer comme inutile tout ce qui a permis d'arriver au point où nous en sommes, de construire les langages que nous utilisons actuellement ? Dernièrement je me suis trouvé confronté à un étudiant qui dénigrait tout ce qui était « ancien » arguant qu'il y avait une « vieille méthode », et une « méthode performante ». Pourtant, dans un contexte d'apprentissage, la « vieille méthode », celle qu'il faudrait mettre au rebut et dont on a honte, permet de comprendre la « nouvelle méthode », celle qui est sous les feux des projecteurs. Parce que pour construire la nouvelle méthode, il a fallu comprendre l'ancienne !

Il n'est pas possible de comprendre le monde dans lequel nous vivons sans connaître un minimum l'Histoire. Il en est de même pour l'informatique. Avant d'utiliser le dernier *framework* à la mode, il est bon de connaître les bases du langage sur lequel il s'appuie. De nos jours on se retrouve confronté à des problèmes de mode qui sont très proches de ce que l'on peut constater dans le domaine vestimentaire. Une année, il « faudra » porter du noir et puis l'année d'après ce sera du fluo, puis des dégradés de bleus, etc. On retrouve le même problème avec les smartphones : pourquoi changer de smartphone tous les 6 mois pour avoir le dernier « I-Sung 4K THX Dolby TrueHD octaconta-core » avec 2Tio de mémoire extensible à 128Pio par ajout d'une carte SuperMicroSD High++, équipé d'une caméra Dual Pixel 120MP ? En dehors de pouvoir frimer pendant deux mois jusqu'à la sortie de l' « I-Sung v2++ », à quoi cela sert-il ? Les commerciaux et publicistes font bien leur travail et du moment qu'il y a des consommateurs, les constructeurs sont heureux. Ce consumérisme se retrouve jusque dans les *frameworks* que nous employons : il faudra absolument utiliser le *framework F1* puis, six mois plus tard *F1* sera totalement *has been*, *F2* sera ce qui se fait de mieux, etc. ! Les arguments seront simples à trouver : « sur Internet, tout le monde utilise *F2* » et si « tout le monde » le fait, alors ils ont forcément raison. On changera donc de *framework* tous les six mois et il faudra réapprendre une logique différente. Pourtant une chose n'aura pas changé : le langage sur lequel s'appuie le *framework*. Si le développeur connaît bien celui-ci, quels que soient les changements il pourra s'adapter rapidement.

Ainsi, il y a deux écoles : soit on accepte d'apprendre les bases de manière à pouvoir évoluer et être capable de s'adapter, soit on se forme à une technologie éphémère de manière à être rentable rapidement et sur une durée très limitée. Tiens, ça rappelle le schéma des « modes » : on prend un développeur pour un an ou deux et après on s'en débarrasse, comme les smartphones. Il existe quand même une différence : pour les smartphones, on commence à les recycler...

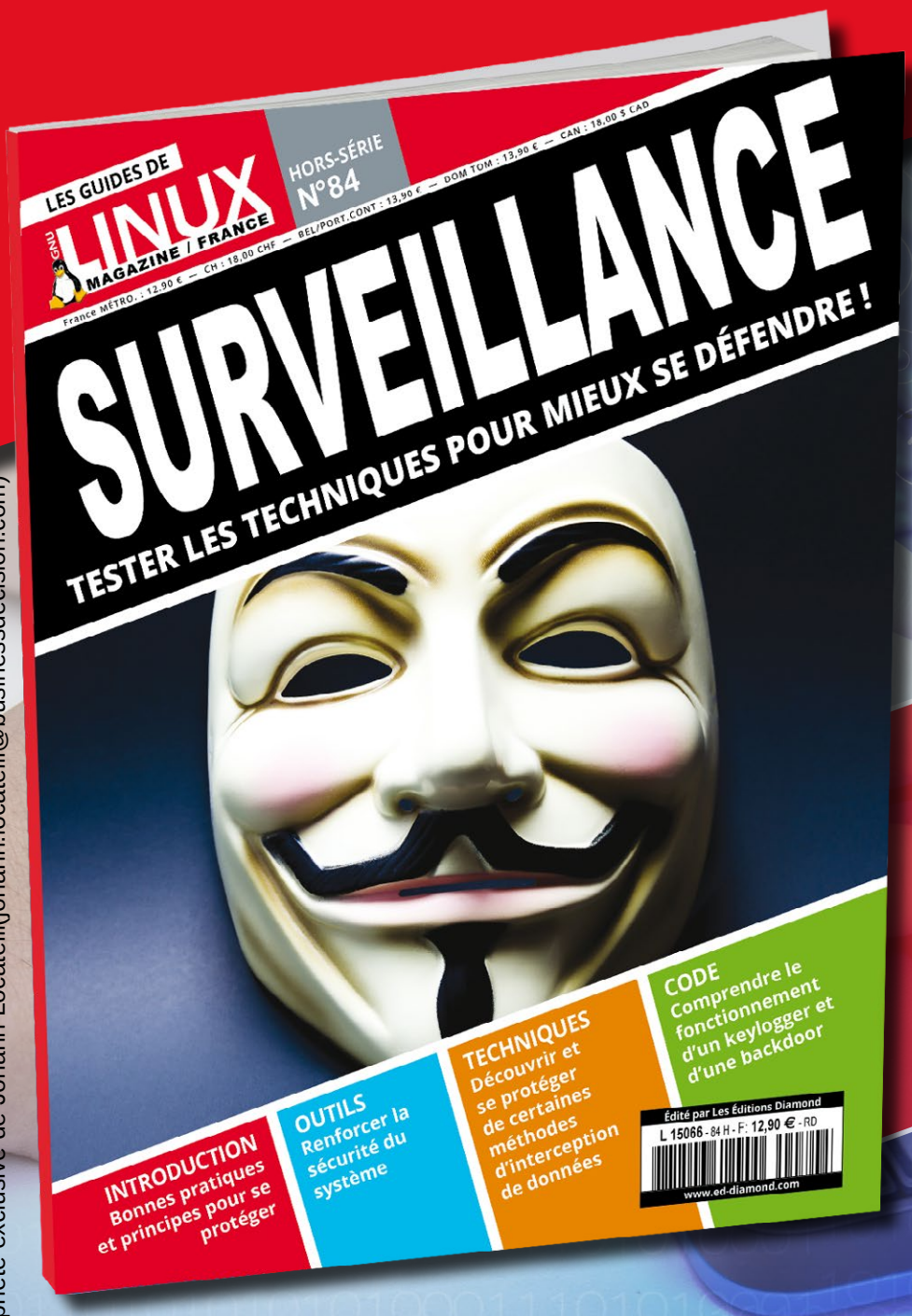
Je n'ai pas pu convaincre mon étudiant, certaines personnes sont trop bornées et je ne pense pas qu'il ouvre jamais un *GNU/Linux Magazine*, pourtant c'est dans ces pages que l'on peut trouver à la fois des notions essentielles et les actualités des derniers *frameworks*.
Je vous souhaite donc une lecture enrichissante !

Tristan Colombo

Ce document est la propriété exclusive de Johann Locatelli(johann.locatelli@businessdecision.com)

ACTUELLEMENT DISPONIBLE

GNU/LINUX MAGAZINE HORS-SÉRIE N°84 !



LE GUIDE POUR
TESTER LES TECHNIQUES POUR MIEUX SE DÉFENDRE !

NE LE MANQUEZ PAS

CHEZ VOTRE MARCHAND DE JOURNAUX ET SUR :

www.ed-diamond.com

Ce document est la propriété exclusive de Johann Locatelli(johann.locatelli@businessdecision.com)

SOMMAIRE

GNU/LINUX MAGAZINE FRANCE N°194

abonnements

27 : offres spéciales professionnels

37/38 : abonnements multi-supports

Ce document est la propriété exclusive de Johann Locatelli(johann.locatelli@businessdecision.com)

LE LANGAGE PERL 6, POURSUITE DU TOUR D'HORIZON

Laurent ROSENFELD [Membre de l'Association des Mongueurs de Perl
et auteur de plusieurs articles et tutoriels sur Perl 5 et Perl 6]

Nous poursuivons ici le tour d'horizon du nouveau langage Perl 6 en abordant les structures de contrôle, les fonctions et la construction de nouveaux opérateurs.

Mots-clés : Conditions, boucles, Fonctions, Fonctions multiples, Opérateurs, Programmation fonctionnelle

Résumé

Dans le premier volet de ce tour d'horizon du nouveau langage de programmation Perl 6, nous avons notamment présenté les différents types de variables et les principaux opérateurs du langage. Le présent article décrit notamment les principales structures de contrôle : les conditions (`if`, `unless`, `given ... when`) et les boucles (`for`, `while`, `until`, `loop`). Il montre ensuite comment créer des fonctions dotées de signatures fortement ou faiblement typées, au choix de l'utilisateur ; on verra notamment comment des signatures différentes permettent d'écrire des fonctions (fonctions dites multiples) ayant le même nom, mais faisant des choses différentes selon le nombre et le type de leurs arguments. Nous verrons enfin combien il est facile de créer dynamiquement de nouveaux opérateurs permettant d'étendre le langage.

Pour commencer ce deuxième article consacré au langage Perl 6, nous voudrions vérifier, cher lecteur, que vous avez bien assimilé le premier article et vous avons réservé à cette fin une petite surprise : nous allons faire un petit examen de contrôle.

- *Exercice 1* : la fonction interne `lcm` renvoie le plus petit multiple commun (PPMC) entre deux nombres. Écrire un programme qui affiche le plus petit nombre positif divisible par tous les nombres de 1 à 20.
- *Exercice 2* : écrire un programme qui calcule la somme des chiffres du nombre factoriel de 100 ;
- *Exercice 3* : trouver la différence entre le carré de la somme des 100 premiers entiers naturels et la somme des carrés des 100 premiers entiers.

Nous n'avons pas encore étudié les conditions (`if`, etc.), ni les boucles (`for`, `while`, etc.), ni les fonctions. Il faut donc trouver dans ce que nous avons déjà appris d'autres solutions pour résoudre ces problèmes. Un indice : relisez tout le chapitre sur les opérateurs. Nous ne relèverons pas les copies, mais essayez vraiment de faire ces exercices.

Lorsqu'au début de mes études d'informatique, j'ai dû faire ce genre d'exercices dans le langage de programmation choisi à l'époque par mes enseignants (Pascal ou C, par exemple), je m'attendais à devoir écrire quelques dizaines de lignes de code pour chacun d'entre eux. Ici, en Perl 6 et avec un modèle de programmation issu de la programmation fonctionnelle, chacun peut se faire en une seule petite ligne de code. Voilà notamment ce que nous voulons dire quand nous disons que Perl 6 est un langage puissant et expressif.

Ce document est la propriété exclusive de Johann Locatelli(johann.locatelli@businessdecision.com)

Vous trouverez des solutions à ces exercices dans l'encadré plus loin.

1 Structures de contrôle : conditions et boucles

1.1 Les conditions simples

La condition **if** / **else** est classique. Sa seule particularité est que, contrairement à beaucoup de langages, elle ne nécessite pas de parenthèses autour de la condition :

```
my $âge = 19;
if $âge >= 18 {
    say "Vous êtes un adulte."
} else {
    say "Vous êtes un jeune."
}
```

À noter que le point-virgule n'est pas nécessaire avant une accolade fermant un bloc. On peut également tester successivement plusieurs conditions avec des clauses **elsif** :

```
if $âge < 12 {
    say "Enfant"
} elsif $âge < 18 {
    say "Adolescent"
} else {
    say "Adulte"
}
```

Une condition **if** peut également se placer après l'instruction (on parle alors d'instruction modifiée) :

```
> say "Bienvenue sur ce site" if $âge >= 18;
Bienvenue sur ce site
```

Cette syntaxe est concise et pratique, mais n'autorise pas de clause **else** ou **elsif**.

La condition **unless** inverse la condition :

```
say  "Désolé : réservé aux adultes !" unless $âge >= 18;
# équivalent à :
say  "Désolé : réservé aux adultes !" if not $âge >= 18;
```

La construction **given ... when** correspond au **switch** d'autres langages, mais permet une formulation beaucoup plus riche et variée des conditions :

```
my Int $âge = 18;
given $âge {
    when *..^0 { say "âge négatif ? Hum..." };
    when 0..2  { say "Bébé" };
    when 3..12 { say "Enfant" };
    when *..17 { say "Adolescent" };
    when   18  { say "tout jeune adulte" };
    default    { say "Adulte"}
}
```

Dès que l'une des conditions est satisfaite, les conditions suivantes ne sont pas évaluées. La condition ***..17**, c'est-à-dire compris entre **0** et **17**, est donc ici correcte et en fait équivalente à **13..17** puisque l'on n'arrive à cette condition que si les trois conditions précédentes ont échoué (ce qui ne veut pas dire qu'il soit recommandé d'écrire une telle bizarrerie contre-intuitive, donnée ici à seul titre d'exemple).

On peut inviter le programme à continuer l'évaluation des conditions suivantes même si une condition est satisfaite avec une instruction **proceed** :

```
my $var = 42;
given $var {
    when 0..50 { say 'Compris entre 0 et 50'; proceed };
    when Int   { say "Un entier"; proceed};
    when /4/   { say "Contient le chiffre 4"; proceed };
    when not .is-prime { say "Non premier";   proceed };
    when 42    { say "réponse à la Grande Question" }
}
```

Ici, grâce aux instructions **proceed**, chacun des cinq messages est affiché l'un après l'autre puisque toutes les conditions sont successivement satisfaites.

1.2 L'opérateur ternaire

L'opérateur ternaire, issu du langage C, utilise **??** et **!!** :

```
> my ($x, $y) = (1, 2);
(1 2)
> say "Max = ", $x > $y ?? $x !! $y;
Max = 2
```

On peut enchaîner plusieurs de ces opérateurs :

```
my Int $âge = 18;
say $âge <= 2 ?? "Bébé"
   !! $âge <= 12 ?? "Enfant"
   !! $âge < 18 ?? "Ado"
   !! $âge == 18 ?? "Jeune adulte"
   !! "Adulte";
```

Ce qui imprime « Jeune adulte ».

Ce document est la propriété exclusive de Johann Locatelli(johann.locatelli@businessdecision.com)

Ce document est la propriété exclusive de Johann Locatelli(johann.locatelli@businessdecision.com)

Solution des exercices

Voici des solutions possibles aux exercices proposés dans la première page de cet article.

PPMC des nombres de 1 à 20

Le métaopérateur de réduction **[...]** permet d'appliquer un opérateur successivement à tous les éléments d'une liste. Il suffit de l'utiliser avec l'opérateur **lcm** sur la liste des nombres de 1 à 20 :

```
> say [lcm] 1..20;
232792560
```

Somme des chiffres de factorielle 100

Nous utilisons ici deux fois le métaopérateur de réduction **[...]** : une première fois avec la multiplication pour calculer **100!** et une seconde fois pour faire la somme des chiffres du résultat. Ce qui donne :

```
> say [+] split '', [*] 2..100;
648
```

Carré de la somme moins somme des carrés

Perl 6 calcule facilement la somme des **100** premiers nombres avec **[+] 1..100**. L'hyperopérateur **<<...>>** permet de calculer les carrés des cent premiers entiers et le métaopérateur **[...]** de réduire cette liste de carrés à leur somme. Ce qui permet d'écrire :

```
say ([+] 1..100)**2 - [+] (1..100) <<**>> 2;
25164150
```

On pourrait alléger le calcul en remarquant que la somme des **100** premiers entiers naturels est égale à **(100 x 101) / 2 = 5050**, mais le but était ici surtout d'employer les hyperopérateurs et métaopérateurs du langage.

1.3 Les boucles

En Perl 6, la boucle la plus commune est la boucle **for**, qui itère sur une liste de valeurs ou les éléments d'un tableau et dont la syntaxe la plus commune est la suivante :

```
> my @tableau = [0..10];
[0 1 2 3 4 5 6 7 8 9 10]
> for @tableau -> $val { print $val * 2, " "};
0 2 4 6 8 10 12 14 16 18 20 >
```

Cette construction s'appelle un « bloc pointu ». Ici, la variable **$val** est un alias en lecture seule sur les valeurs successives du tableau ; elle n'a pas besoin d'être pré-déclarée et sa portée est naturellement limitée au bloc d'instructions de la boucle **for**. Le lecteur féru de programmation fonctionnelle aura peut-être reconnu dans cette construction de « bloc pointu » à la fois une lambda et une fermeture anonyme.

La boucle **for** ci-dessus itère directement sur les éléments du tableau, ce qui correspond au besoin le plus fréquent. Si l'on a besoin d'itérer sur les indices, il suffit de créer à la volée une liste des indices à l'aide de l'opérateur intervalle **..** et de la fonction ou méthode **end** (retournant l'indice du dernier élément d'un tableau) et d'itérer sur cette liste :

```
my @mois = <none jan fév mar avr mai jun>;
for 1..end @mois -> $num { say "$num \t @mois[$num]" };
# ou: for 1..@mois.end  -> ...
```

On peut également utiliser les itérateurs de listes du langage, par exemple **keys** pour récupérer la liste d'indices :

```
my @nombres = <zéro un deux trois quatre cinq>;
for @nombres.keys -> $indice  { say "$indice \t @
nombres[$indice]" };
```

Ou encore **kv** pour récupérer à la fois les indices et les éléments correspondants :

```
for @nombres.kv -> $indice, $nombre  { say "$indice \t $nombre" };
```

Si l'on a besoin de modifier les valeurs du tableau, il faut un alias en lecture et écriture, ce qui se fait à l'aide d'un bloc « doublement pointu » :

```
my @tableau = [0..10];
for @tableau <-> $val { $val *= 3 }
say @tableau;  # affiche [0 3 6 ... 30]
```

Une syntaxe d'expression modifiée est également possible :

```
> print $_ * 2, " " for [0..10];
0 2 4 6 8 10 12 14 16 18 20 >
```

Une boucle **for** est un itérateur et peut donc travailler de façon « paresseuse » (c'est-à-dire qu'elle ne traite les éléments qu'au fur et à mesure des besoins), même sur une liste « infinie ». Employer une boucle **for** est donc une façon idiomatique d'itérer sur les lignes d'un fichier :

```
for "fichier.txt".IO.lines -> $ligne {
    say $ligne unless $ligne ~~ /^'#'/
}
```

La boucle **while** ressemble à celle des autres langages communs et exécute la boucle tant que la condition est vraie, si ce n'est que les parenthèses ne sont pas nécessaires autour de la condition :

```
> my $var = 0;
0
> while $var < 5 { print 3 * ++$var, " ";}
3 6 9 12 15 >
```

Une boucle **until** exécute le bloc de la boucle tant que la condition est fausse. Voici un exemple utilisant une syntaxe d'instruction modifiée :

```
my $x = 0;
print $x++ until $x > 5; # imprime 012345
```

Vous pouvez enfin utiliser une boucle **loop**, qui correspond à la boucle **for** du langage C et des langages apparentés :

```
loop (my $i=0; $i < 5; $i++) {
  say "Le nombre actuel est : $i"
}
```

Cette syntaxe permet de construire des boucles complexes, mais il est rare d'avoir besoin de ce genre de boucle en Perl 6 : la boucle **for** de Perl 6 est généralement bien plus pratique. C'est la seule construction de boucle pour laquelle les parenthèses restent nécessaires autour des conditions/initialisations. Sa seule utilisation fréquente est la façon idiomatique d'écrire une boucle infinie :

```
loop {...}
```

Les instructions **next** et **last** permettent, respectivement, de recommencer à l'itération suivante et de sortir immédiatement d'une boucle **for**, **while** ou autre :

```
for 1..20 -> $i {
    next if 3 < $i <= 7;
    last if $i == 10;
    print "$i ";
}   # imprime 1 2 3 8 9
```

2 Fonctions ou sous-routines

Les fonctions permettent de regrouper un ensemble de fonctionnalités. On déclare une fonction à l'aide du mot-clef **sub** (pour *subroutine*) suivi de son identifiant (son nom), et l'on définit ce qu'elle fait dans un bloc de code placé entre accolades.

```
saluer();   # imprime "Bonjour ..."
sub saluer {
     say 'Bonjour tout le monde.';
}
```

Contrairement à certains langages, la fonction peut être appelée avant sa définition. L'instruction **saluer();** est l'appel de la fonction. C'est lors de l'exécution de cette ligne de code que la fonction s'exécute.

2.1 Arguments et signatures

La fonction ci-dessus ne prend pas de paramètres en entrée et ne renvoie pas de valeurs de retour, ce qui limite quelque peu son utilité. La plupart des fonctions utiles ont besoin de données en entrée, qui leur sont fournies sous la forme d'arguments. Voici une fonction analogue avec le passage d'un argument :

```
saluer("Maître");   # imprime "Bonjour Maître."
sub saluer ($titre) {
     say "Bonjour $titre.";
}
```

Ici, l'appel de la fonction se fait avec un argument, la chaîne **"Maître"**. La définition de la fonction contient maintenant une *signature* placée entre parenthèses ; cette signature définit la variable **$titre** comme le paramètre (formel) que reçoit la fonction. Dans cet exemple, le paramètre **$titre** prend la valeur de l'argument d'appel, « Maître », et est utilisé ensuite dans le corps de la fonction. Cette signature est réduite à sa plus simple expression, mais elle a déjà pour effet d'obliger le code utilisateur à passer un argument (et un seul) lors de l'appel à la fonction. Si le nombre d'arguments est différent de un, cela génère une erreur.

Ce document est la propriété exclusive de Johann Locatelli(johann.locatelli@businessdecision.com)

Par défaut, les paramètres des fonctions sont des copies en lecture seule des arguments reçus (c'est une forme de passage de paramètre par valeur) et ne peuvent donc pas être modifiés dans la fonction. Mais on peut changer ce comportement grâce à des *traits* (propriétés définies à la compilation) tels que **is rw** (lecture écriture, comme lors d'un passage de paramètres par référence) ou **is copy** (copie locale à la fonction et modifiable) :

```
my Int $valeur = 5;
ajoute-deux($valeur);
say $valeur; # -> 7
sub ajoute-deux (Int $x is rw) { $x += 2}
```

La signature permet non seulement de vérifier le nombre d'arguments, mais aussi leur type. Voici un exemple de fonction ayant une signature un peu plus complexe :

```
sub multiplie (Int $x, Int $y) {
    say "Produit = ", $x * $y
}
multiplie(3, 4); # -> Produit = 12
```

Ici, la signature comporte deux paramètres, **$x** et **$y**. Ce sont des paramètres positionnels : le paramètre **$x** reçoit la valeur du premier argument passé à la fonction et le paramètre **$y** reçoit le second argument. En outre, ces deux paramètres sont de type entier. Le compilateur vérifiera donc que la fonction est bien appelée avec deux arguments de type entier, sinon il émettra une erreur. À noter que si la fonction est comme ici déclarée avant son appel, il est possible d'appeler la fonction en omettant les parenthèses autour des arguments :

```
multiplie 3, 4;  # -> Produit = 12
```

Cette fonction ne peut multiplier que des entiers, ce qui en limite l'utilité. Il est bien sûr possible de définir des signatures sur des types numériques quelconques, par exemple en utilisant le type générique **Numeric**.

Il peut être difficile de se souvenir de l'ordre des paramètres positionnels, surtout quand la fonction accepte de nombreux arguments. Il est dans ce cas possible d'utiliser des paramètres nommés rendant l'ordre des arguments indifférents, avec la syntaxe suivante :

```
sub divise (Numeric :$dividende, Numeric :$diviseur) {
    say $dividende/$diviseur
}
divise(dividende => 12, diviseur => 4);  # -> 3
divise diviseur => 4, dividende => 12 ;  # idem
```

Le second exemple d'appel de la fonction **divise** ci-dessus montre que les parenthèses sont ici encore optionnelles si la fonction a été déclarée avec sa signature avant son appel.

2.2 Fonctions multiples

Il est possible de définir avec le mot-clef **multi** des fonctions spéciales ayant le même nom, mais se distinguant par leur signature (nombre et type des arguments). Perl détermine quel exemplaire de la fonction appeler en fonction de la signature (processus analogue à la résolution des méthodes en programmation orientée objet).

```
multi salue ($nom)         {say "Bonjour $nom"}
multi salue ($nom, $titre) {say "Bonjour $titre $nom"}
salue("George Lucas");    # -> Bonjour George Lucas
salue "Yoda", "Maître";   # -> Bonjour Maître Yoda
```

Ici, c'est le nombre d'arguments de la fonction qui permet à Perl 6 de déterminer quelle version de **salue** appeler. La distinction pourrait aussi se faire sur le type (voire le « sous-type »).

Beaucoup des opérateurs et des fonctions ou méthodes internes de Perl 6 sont définis comme des fonctions multiples. Cela signifie qu'en utilisant une signature n'existant pas en interne, il est possible de redéfinir ou surcharger ces fonctions ou opérateurs.

2.3 Valeurs de retour

Toutes nos fonctions jusqu'ici se contentaient d'afficher quelque chose à l'écran. Très souvent, il est souhaitable qu'une fonction se contente de prendre des paramètres en entrée et de renvoyer des valeurs de retour, sans avoir d'effet de bord. Cela facilite la conception et la mise au point de programmes.

Voici la définition et l'utilisation d'une fonction renvoyant une valeur utile :

```
> sub carré ($x) { return $x * $x }
sub carré ($x) { #`(Sub|183561872) ... }
> say carré 3
9
```

Le mot-clef **return** indique explicitement que la fonction doit se terminer et renvoyer la valeur donnée. Et l'appel **carré 3** renvoie à **say** la valeur **9**. En fait, ce mot-clef n'était pas indispensable ici, car une fonction renvoie implicitement la dernière expression évaluée, si bien que la fonction aurait aussi bien pu s'écrire :

Ce document est la propriété exclusive de Johann Locatelli(johann.locatelli@businessdecision.com)

```
sub carré ($x) { $x ** 2 }
```

Cela suffit pour des fonctions simples n'ayant qu'un seul point de sortie, mais l'utilisation de **return** permet de définir finement le comportement d'une fonction et les valeurs de retour selon les conditions rencontrées.

3 Définir ou redéfinir un opérateur

Les opérateurs de Perl 6 sont en fait des fonctions ou des méthodes ayant souvent un nom un peu inhabituel et définissant quelques propriétés permettant de les utiliser : un opérateur est généralement doté d'une précédence (priorité d'exécution), d'un type de notation syntaxique (préfixée, postfixée, infixée, etc.) et d'une associativité (comment il se comporte quand il y a plusieurs opérateurs de même précédence). Pour créer un nouvel opérateur, il faut au minimum définir son type de notation, et, selon le besoin, éventuellement préciser sa précédence et son associativité.

Nous pourrions par exemple utiliser le symbole euro « **€** » pour définir un opérateur de doublement d'un entier :

```
sub postfix:<Ð> (Int $n) {2*$n}
say 21Ð; # -> 42
```

Ce nouvel opérateur n'est sans doute pas très utile (et son nom n'est pas idéalement choisi pour ce qu'il fait), mais illustre simplement comment il est possible d'enrichir dynamiquement le langage.

À titre d'exemple d'un nom d'opérateur peut-être mieux choisi, utilisons le petit « 2 » en indice du clavier pour définir un opérateur d'élévation au carré :

```
sub postfix:<²>(Numeric $n) {$n**2}
say 5²;        # -> 25
say (3 + 4)²;  # -> 49
```

Perl 6 supportant les caractères unicodes, il est possible d'utiliser les symboles mathématiques (par exemple le symbole racine carrée), les lettres grecques, tibétaines ou d'autres langues, les guillemets japonais 「」, les pictogrammes normalisés, les lettres braille, etc. pour définir des opérateurs.

Un opérateur n'est pas nécessairement un caractère spécial unique, nous pourrions (ou du moins aurions pu, il y a quelques années, ce n'est plus très utile aujourd'hui) définir des opérateurs de conversion de francs en euros et réciproquement :

```
sub prefix:<fÐ> (Numeric $n) {$n/6.55957}
sub prefix:<Ðf> (Numeric $n) {$n*6.55957}
say fÐ 6.56;   # -> 1.0000656
say Ðf 1Ø;     # -> 65.5957
```

Définir un opérateur infixé ne pose pas plus de problème. Par exemple, voici un opérateur de calcul de la moyenne entre deux nombres :

```
sub infix:<moy> (Numeric $n, Numeric $m) {($n+$m)/2}
say 1Ø moy 5; # -> 7.5
```

L'opérateur **!** de négation booléenne est de type préfixé, c'est-à-dire qu'il se place avant le terme auquel il s'applique. Nous pouvons réutiliser le même opérateur **!** pour définir la factorielle, qui utilisera naturellement, comme en mathématiques, une notation postfixée :

```
sub postfix:<!> (Int $n) {
    [*] 2..$n
}
say 2Ø!;      # -> 2432902008176640000
say ! False;  # -> True
```

Voici enfin un exemple dans lequel nous définissons aussi la précédence de l'opérateur. Il existe en Perl 6 un type **Pair** qui définit une paire clef-valeur et se note généralement **clef => valeur**. Nous pourrions vouloir utiliser ce type pour modéliser des couples de valeurs que nous désirons pouvoir additionner membre à membre. Il suffit de définir l'addition de paires et, tant qu'à faire, lui donner la même propriété de précédence que l'addition arithmétique :

```
multi sub infix:<+> (Pair $x, Pair $y) is equiv(&infix:<+>) {
    return $x.key + $y.key => $x.value + $y.value
}
my $a = 4=>3;
my $b = 7=>2;
say $a + $b; # -> 11 => 5
```

Le « trait » **is equiv(&infix:<+>)** précise que cet opérateur a la même précédence que l'opérateur d'addition. Si nous définissons une multiplication membre à membre sur le même modèle en lui donnant la même précédence que la multiplication, nous retrouverons les règles de précédence habituelles entre nos opérations sur les paires.

La création de nouveaux opérateurs est un moyen simple d'enrichir dynamiquement le langage. Pour des enrichissements plus complexes, il est possible de modifier dynamiquement la grammaire même de Perl 6, mais cela nécessiterait de présenter les grammaires de Perl et sortirait du cadre de cette brève présentation.

Ce document est la propriété exclusive de Johann Locatelli(johann.locatelli@businessdecision.com)

Conclusion

Nous avons fait un bref tour d'horizon de la syntaxe de base de Perl 6 et avons aussi essayé de présenter quelques-unes des caractéristiques (fonctions multiples, création de nouveaux opérateurs, etc.) qui le rendent particulièrement puissant et expressif.

Parmi celles que nous n'avons pas pu décrire faute de place (mais pourrons peut-être aborder dans des articles ultérieurs), Perl 6 offre en particulier :

- un nouveau système de programmation orientée objet particulièrement flexible, puissant et expressif, doté de classes, de méthodes et de rôles, la possibilité de créer facilement de nouveaux types, une introspection approfondie et une couche méta-objet permettant de modifier dynamiquement le comportement des objets et des classes ;
- un système d'expressions régulières nettoyé et refondu, rendu plus lisible et modulaire grâce aux regex nommées qui sont en quelque sorte des briques permettant de construire des expressions régulières bien plus puissantes tout en étant plus lisibles, et débouchant sur la création de véritables grammaires permettant l'analyse lexicale et syntaxique non seulement de texte HTML, JSON, XML, etc., mais aussi de langages de programmation : la grammaire utilisée par Perl 6 pour analyser les programmes Perl 6 est elle-même écrite en Perl 6, et il est même possible dans un programme ou un module d'ajouter de nouveaux éléments syntaxiques à la grammaire Perl 6 existante, ce qui rend le langage intrinsèquement malléable et évolutif ;
- un modèle de programmation fonctionnelle très enrichi, avec en natif le support aux listes paresseuses, la programmation en pipe-line, les fonctions d'ordre supérieur, les itérateurs, les hyperopérateurs, les fermetures, les lambdas, la curryfication, etc. ;
- un support exceptionnellement efficace de l'Unicode, probablement sans égal actuellement ;
- un modèle de programmation parallèle, concurrente et asynchrone de haut niveau, bien plus puissant et expressif que les *threads*, sémaphores et verrous, fiable, robuste, facile à utiliser et extrêmement prometteur, s'ajoutant à la possibilité d'utilisation implicite par le compilateur de *threads* utilisant différents *cores* pour traiter des données en parallèle, ainsi qu'un support natif de la programmation évènementielle ;
- un support natif des structures de données multidimensionnelles de plus bas niveau (matrices, etc.) permettant d'envisager du calcul scientifique intensif ;
- un interfaçage particulièrement simple avec des bibliothèques C/C++, Java ou autres, ainsi qu'avec les anciennes versions de Perl, ce qui permet l'utilisation de modules Perl 5 en Perl 6 (ou l'inverse) et ouvre donc la voie à l'utilisation des modules Perl 5 du CPAN, l'une des plus vastes collections de bibliothèques logicielles libres au monde.

La liste ci-dessus pourrait se poursuivre sur plusieurs pages, mais nous nous arrêterons là pour éviter de lasser le lecteur. Tout cela fait de Perl 6 un langage exceptionnellement expressif et intrinsèquement malléable ; même ce qui n'existe pas dans le langage, vous pouvez presque toujours le créer dynamiquement à la demande.

Tout n'est pas encore parfait pour autant. Les performances d'exécution se sont considérablement améliorées depuis un an, mais elles laissent encore un peu à désirer dans certains cas. Les macros avancées ne sont pas encore implémentées et deux ou trois autres fonctionnalités initialement prévues (comme les entrées/sorties non bloquantes) sont encore incomplètement mises en œuvre. Quelques progrès sont donc encore nécessaires (et font l'objet de travaux intensifs), mais Perl 6 offre d'ores et déjà une palette de fonctionnalités que nous pensons très largement inégalée, même parmi les langages récents bénéficiant des ressources incomparablement plus abondantes d'entreprises richissimes comme Google, Apple, Microsoft, Oracle et quelques autres géants du Nasdaq. ■

Pour aller plus loin

- Le site de téléchargement de Rakudo Star/Perl 6 : http://rakudo.org/downloads/star/
- La documentation officielle (en anglais) sur Perl 6 est disponible à l'adresse suivante : doc.perl6.org
- Plusieurs articles et tutoriels en français sont en ligne à l'adresse suivante : http://perl.developpez.com/cours/#TutorielsPerl6
- Le site des Mongueurs de Perl fournit de nombreux liens complémentaires : http://mongueurs.pm/ressources/ref_perl6.html

Ce document est la propriété exclusive de Johann Locatelli(johann.locatelli@businessdecision.com)

DOMAINES | SITES WEB | SERVEURS CLOUD | HÉBERGEMENT

LA PLUS SÛRE DES SÉCURITÉS

TRANSACTIONS 100 % SÉCURISÉES
SSL INCLUS

HÉBERGEMENT SÉCURISÉ ? BIEN SÛR !

1&1 offre le plus haut niveau de protection actuellement disponible. Montrez à vos visiteurs que leur sécurité est votre priorité absolue :

- ✓ Certificat SSL inclus
- ✓ Data centers certifiés
- ✓ Géo-redondance
- ✓ Protection DDoS

CERTIFICAT SSL
SÉCURITÉ MAXIMALE
SEULEMENT CHEZ 1&1 !

1 MOIS POUR ESSAYER

1 CLIC POUR CHANGER DE PACK

1 APPEL UN EXPERT VOUS RÉPOND

0970 808 911
(appel non surtaxé)

1&1

1and1.fr

Conditions détaillées sur 1and1.fr. 1&1 Internet SARL, RCS Sarreguemines B 431 303 775.

Ce document est la propriété exclusive de Johann Locatelli(johann.locatelli@businessdecision.com)

POSTGRESQL 9.5 : LES NOUVELLES FONCTIONNALITÉS SQL

Guillaume LELARGE [Contributeur majeur de PostgreSQL, Consultant Dalibo, auteur du livre « PostgreSQL – Architecture et notions avancées »]

La version 9.5 de PostgreSQL apporte de nombreuses nouveautés. Comme il serait difficile de tout énumérer en un seul article, nous allons aborder ici certaines fonctionnalités relatives au SQL et aux utilisateurs finaux de PostgreSQL. Un gros travail a été effectué par les développeurs pour permettre de gérer de plus gros volumes de données, et cela se voit dans les fonctionnalités que nous allons décrire maintenant.

Mots-clés : PostgreSQL, Nouveautés, Standard SQL, Fonctions OLAP, Sharding

Résumé

PostgreSQL est un moteur de bases de données évoluant constamment. Avec une nouvelle version majeure chaque année, il est souvent difficile de suivre les améliorations apportées. Cet article a pour but de détailler quelques fonctionnalités importantes au niveau SQL proposées par la version 9.5. Il cible principalement les développeurs qui travaillent avec PostgreSQL, mais aussi les utilisateurs et les administrateurs.

Même si PostgreSQL est très proche du standard SQL, il reste un certain nombre de manques que chaque nouvelle version majeure essaie de combler. Pour cette version, les fonctionnalités visent surtout les gros entrepôts de données.

1 Insérer ou mettre à jour en une requête

Depuis sa version SQL:2003, le standard SQL propose une instruction **MERGE** (aussi appelée **UPSERT**) pour insérer une nouvelle ligne ou mettre à jour la ligne si une condition est respectée. Un grand nombre de moteurs de bases de données (comme Oracle, DB2 ou SQL Server) disposent d'une implémentation de cette instruction. Il aura fallu attendre la version 9.5 pour que PostgreSQL propose une commande équivalente.

Cette commande est la commande **INSERT** à qui a été ajoutée une clause optionnelle **ON CONFLICT**. Cette clause permet d'indiquer la marche à suivre si l'insertion ne réussit pas. Deux possibilités s'ouvrent aux utilisateurs : soit ignorer la ligne (auquel cas la ligne n'est ni insérée ni mise à jour... mais tout simplement, la commande ne renvoie pas d'erreur, ce qui évite d'avoir à annuler la transaction en cours), soit mettre à jour la ligne.

```
postgres=# CREATE TABLE t1 (c1 serial primary key, c2 text, c3 boolean);
CREATE TABLE
postgres=# INSERT INTO t1 (c2, c3) SELECT 'Ligne '||i, false FROM
generate_series(1, 10) i;
INSERT 0 10
```

Ce document est la propriété exclusive de Johann Locatelli(johann.locatelli@businessdecision.com)

```
postgres=# INSERT INTO t1 VALUES (1, 'toto', true);
ERROR:  duplicate key value violates unique constraint "t1_pkey"
DETAIL:  Key (c1)=(1) already exists.
```

C'est le cas standard : en cas de violation d'une contrainte, la commande **INSERT** est en erreur. La nouvelle clause **ON CONFLICT** permet de changer ce comportement.

```
postgres=# INSERT INTO t1 VALUES (1, 'toto', true) ON CONFLICT DO NOTHING;
INSERT 0 0
postgres=# SELECT * FROM t1 WHERE c1=1;
 c1 |   c2    | c3
----+---------+----
  1 | Ligne 1 | f
(1 row)
```

Dans cet exemple, **DO NOTHING** permet de ne pas renvoyer d'erreur en cas de conflit avec une contrainte. Il n'y a pas eu d'erreur, et la ligne n'a pas été insérée à cause du conflit sur la clé primaire.

```
postgres=# INSERT INTO t1 VALUES (1, 'toto', true) ON CONFLICT ON
CONSTRAINT t1_pkey DO UPDATE SET c2=excluded.c2;
INSERT 0 1
postgres=# SELECT * FROM t1 WHERE c1=1;
 c1 |  c2  | c3
----+------+----
  1 | toto | f
(1 row)
```

Il y a eu conflit lors de l'insertion, donc la commande **INSERT** a été transformée en commande **UPDATE**, et la colonne **c2** a été modifiée comme indiqué. La pseudo-table **excluded** contient la ligne qui n'a pas pu être insérée et sert donc à récupérer les valeurs de la ligne à insérer. Notez que la colonne **c3** n'a pas été modifiée. Sa valeur était différente dans la commande **INSERT** que celle présente, mais la commande **UPDATE** ne précise pas de mise à jour de cette colonne.

```
postgres=# INSERT INTO t1 (c1, c2, c3) SELECT i, 'Ligne '||i,
false FROM generate_series(9, 12) i ON CONFLICT ON CONSTRAINT
t1_pkey DO UPDATE SET c2=excluded.c2, c3=(excluded.c1%2=1);
INSERT 0 4

postgres=# SELECT * FROM t1 ORDER BY c1;
 c1 |    c2    | c3
----+----------+----
  1 | toto     | f
  2 | Ligne 2  | f
  3 | Ligne 3  | f
  4 | Ligne 4  | f
  5 | Ligne 5  | f
  6 | Ligne 6  | f
  7 | Ligne 7  | f
  8 | Ligne 8  | f
  9 | Ligne 9  | t
 10 | Ligne 10 | f
 11 | Ligne 11 | f
 12 | Ligne 12 | f
(12 rows)
```

Dans cet exemple, deux lignes, les 9 et 10, sont mises à jour, car elles existaient déjà, et les deux suivantes, 11 et 12, sont insérées.

Enfin, il est à noter que la commande **INSERT** est réellement tentée. Donc si une séquence est utilisée, elle est incrémentée que la commande **INSERT** réussisse ou pas.

```
postgres=# INSERT INTO t1 (c2) SELECT 'Ligne '||i FROM generate_
series(1, 10) i ON CONFLICT ON CONSTRAINT t1_pkey DO UPDATE SET
c2=excluded.c2;
INSERT 0 10

postgres=# SELECT * FROM t1 ORDER BY c1;
 c1 |    c2    | c3
----+----------+----
  1 | toto     | f
  2 | Ligne 2  | f
  3 | Ligne 3  | f
  4 | Ligne 4  | f
  5 | Ligne 5  | f
  6 | Ligne 6  | f
  7 | Ligne 7  | f
  8 | Ligne 8  | f
  9 | Ligne 9  | t
 10 | Ligne 10 | f
 11 | Ligne 1  | f
 12 | Ligne 2  | f
 13 | Ligne 3  |
 14 | Ligne 4  |
 15 | Ligne 5  |
 16 | Ligne 6  |
 17 | Ligne 7  |
 18 | Ligne 8  |
 19 | Ligne 9  |
 20 | Ligne 10 |
```

Ce résultat pourrait en étonner plus d'un, donc une petite explication s'impose. La commande **INSERT** va insérer dix lignes, mais elle ne commence pas à partir de la ligne 1 dans la table. Elle commence à partir de la dernière valeur de la séquence. Celle-ci valait **10**. Donc la première ligne que la commande essaie d'insérer est la ligne 11. Cette ligne existant déjà, elle est modifiée au niveau de la colonne **c2**, qui se trouve avoir le texte « Ligne » agrégé à la première valeur de la fonction **generate_series**, donc **1**. La même action arrive pour la ligne 12, déjà existante. Par contre, les lignes 13 à 20 n'existant pas, elles ont été insérées avec le texte « Ligne 3 » à « Ligne 10 ». C'est logique, mais ça surprend sur le coup. Donc attention lors de l'écriture de vos requêtes.

Ce document est la propriété exclusive de Johann Locatelli(johann.locatelli@businessdecision.com)

2 Quelques fonctionnalités OLAP

Même si PostgreSQL fonctionne très bien avec de grosses volumétries, il lui manque certaines fonctionnalités, notamment pour les applications OLAP (*OnLine Analytical Processing*). Prenons la table suivante :

```
postgres=# SELECT * FROM livres;
         livre         |    tag1     |        tag2
-----------------------+-------------+--------------------
 The gods of guilt     | policier    | série Mickey Haller
 Un chapeau de ciel    | fantastique | série Tiphaine
 Le cinquième éléphant | fantastique | série DiscWorld
 The Poet              | policier    | série Jack McEvoy
 The Lincoln Lawyer    | policier    | série Mickey Haller
 Le père porcher       | fantastique | série DiscWorld
 La Vérité             | fantastique | série DiscWorld
(7 rows)
```

La clause **GROUP BY** permet de regrouper les colonnes indiquées suivant leur valeur, et de faire des calculs sur le regroupement effectué :

```
postgres=# SELECT tag1, tag2, count(*) FROM livres GROUP BY tag1, tag2;
    tag1     |        tag2         | count
-------------+---------------------+-------
 policier    | série Mickey Haller |     2
 policier    | série Jack McEvoy   |     1
 fantastique | série Tiphaine      |     1
 fantastique | série DiscWorld     |     3
(4 rows)
```

Nous avons donc le décompte de livres par rapport à un regroupement sur les colonnes **tag1** et **tag2**. Cependant, nous n'avons pas le décompte par **tag1** et **tag2** séparément. La clause **GROUPING SETS** le permet :

```
postgres=# SELECT tag1, tag2, count(*) FROM livres GROUP BY
GROUPING SETS(tag1, tag2);
    tag1     |        tag2         | count
-------------+---------------------+-------
 fantastique |                     |     4
 policier    |                     |     3
             | série DiscWorld     |     3
             | série Jack McEvoy   |     1
             | série Mickey Haller |     2
             | série Tiphaine      |     1
(6 rows)
```

Mais cette fois, il manque le décompte des deux colonnes regroupées. La clause **ROLLUP** permet d'obtenir ce décompte et le décompte sur la première colonne :

```
postgres=# SELECT tag1, tag2, count(*) FROM livres GROUP BY
ROLLUP(tag1, tag2);
    tag1     |        tag2         | count
-------------+---------------------+-------
 fantastique | série DiscWorld     |     3
 fantastique | série Tiphaine      |     1
 fantastique |                     |     4
 policier    | série Jack McEvoy   |     1
 policier    | série Mickey Haller |     2
 policier    |                     |     3
             |                     |     7
(7 rows)
```

Il suffit d'inverser les colonnes dans la clause **ROLLUP** pour avoir le décompte du regroupement et le décompte de la deuxième colonne. Enfin, il est possible d'obtenir le décompte de chaque colonne séparément et des colonnes regroupées en utilisant la clause **CUBE** :

```
postgres=# SELECT tag1, tag2, count(*) FROM livres GROUP BY
CUBE(tag1, tag2);
    tag1     |        tag2         | count
-------------+---------------------+-------
 fantastique | série DiscWorld     |     3
 fantastique | série Tiphaine      |     1
 fantastique |                     |     4
 policier    | série Jack McEvoy   |     1
 policier    | série Mickey Haller |     2
 policier    |                     |     3
             |                     |     7
             | série DiscWorld     |     3
             | série Jack McEvoy   |     1
             | série Mickey Haller |     2
             | série Tiphaine      |     1
(11 rows)
```

Pour connaître le type de regroupement sur une ligne particulière, il est possible d'utiliser la fonction **grouping** :

```
postgres=# SELECT grouping(tag1,tag2), tag1, tag2, count(*) FROM
livres GROUP BY CUBE(tag2, tag1) ORDER BY grouping(tag1,tag2);
 grouping |    tag1     |        tag2         | count
----------+-------------+---------------------+-------
        0 | policier    | série Mickey Haller |     2
        0 | fantastique | série DiscWorld     |     3
        0 | policier    | série Jack McEvoy   |     1
        0 | fantastique | série Tiphaine      |     1
        1 | policier    |                     |     3
        1 | fantastique |                     |     4
        2 |             | série Jack McEvoy   |     1
        2 |             | série Tiphaine      |     1
        2 |             | série DiscWorld     |     3
        2 |             | série Mickey Haller |     2
        3 |             |                     |     7
(11 rows)
```

3 SKIP LOCKED ou comment implémenter simplement une queue de traitement

L'implémentation d'une queue de traitement n'est généralement pas une chose simple en SQL, surtout avec plusieurs consommateurs. Il faut pouvoir lire *n* éléments à traiter, les verrouiller pour qu'un autre consommateur ne travaille pas sur les mêmes que le premier consommateur, tout en s'assurant que l'autre consommateur ne soit pas bloqué. En n'utilisant que du SQL, c'est très difficile.

Ce document est la propriété exclusive de Johann Locatelli(johann.locatelli@businessdecision.com)

En utilisant l'ordre **SELECT ... FOR UPDATE**, il est possible de lire *n* éléments et de les verrouiller. Le problème, c'est que le prochain à exécuter cet ordre va se retrouver bloqué sur au moins une des lignes en cours de traitement par le précédent utilisateur, vu qu'elles sont verrouillées en modification. La clause **NOWAIT** a été ajoutée pour que cet utilisateur ne soit pas bloqué. Ceci étant dit, sa requête tombe en erreur. Il faut qu'il la relance en espérant tomber sur des lignes non verrouillées. L'opération n'est pas impossible, mais elle se complexifie fortement.

```
Session 1:
postgres=# BEGIN;
BEGIN
postgres=# SELECT * FROM jobs ORDER BY
id LIMIT 2 FOR UPDATE;
 id |  tache
----+---------
  1 | Tache 1
  2 | Tache 2
(2 rows)

Session 2:
postgres=# BEGIN;
BEGIN
postgres=# SELECT * FROM jobs ORDER BY
id LIMIT 2 FOR UPDATE;
```

(Oups, bloqué)

```
Session 3:
postgres=# BEGIN;
BEGIN
postgres=# SELECT * FROM jobs ORDER BY id
LIMIT 2 FOR UPDATE NOWAIT;
ERROR:  could not obtain lock on row in
relation "jobs"
```

(il va falloir récupérer les deux éléments suivants... possible avec une clause **WHERE id>2**, mais il n'est pas dit que les lignes 3 et supérieures ne sont pas elles aussi bloquées)

Arrive la clause **SKIP LOCKED** avec la version 9.5 de PostgreSQL. Cette clause permet de lire les *n* premières lignes non verrouillées.

```
Session 4:
postgres=# BEGIN;
BEGIN
postgres=# SELECT * FROM jobs ORDER BY id
LIMIT 2 FOR UPDATE SKIP LOCKED;
 id |  tache
----+---------
  3 | Tache 3
  4 | Tache 4
(2 rows)
```

Cette nouvelle clause devrait être rapidement adoptée par un bon nombre de développeurs, surtout qu'elle existe dans **Oracle** depuis la version 11g.

4 Procédures stockées

Lors de l'utilisation de types de données standards en SQL, tout langage est capable de gérer le type de données. Par contre, quand il s'agit d'un type un peu complexe ou structuré, il existe fréquemment un type avancé dans certains langages comme **Python** et **Perl**. L'ajout des objets transformations va permettre d'aller plus loin. Ils sont déjà utilisés par le module **hstore** qui transforme un objet SQL de type hstore en un objet Python de type dictionnaire, et inversement. Perl est aussi concerné : un objet hash est initialisé pour un objet SQL hstore.

Un autre changement est la possibilité d'utiliser l'opérateur **=>** au lieu de l'opérateur **=** pour indiquer les paramètres nommés et leurs valeurs dans un appel de fonction.

Le langage **PL/pgsql** dispose maintenant d'une instruction **ASSERT** permettant de débugguer certaines fonctions lorsque des valeurs invalides sont fournies en tant qu'argument.

5 Divers

D'autres améliorations SQL ont été apportées pour faciliter la vie aux utilisateurs et aux administrateurs.

Par exemple, il était impossible de modifier le statut *template* d'une base de données ainsi que le statut d'autorisation de connexions sans modifier directement le catalogue système **pg_database**. Maintenant, les ordres SQL **CREATE DATABASE** et **ALTER DATABASE** disposent des clauses **IS_TEMPLATE** et **ALLOW_CONNECTIONS**. Ce n'est qu'une petite amélioration, mais elle facilitera la vie d'un bon nombre d'administrateurs.

Autre exemple, la revue des instructions de création d'objet pour leur ajouter la clause **IF NOT EXISTS**. Cette clause est enfin disponible pour les index, séquences, vues matérialisées et contraintes. Cette clause permet de ne pas retourner d'erreur si un objet de même nom existe déjà. Il est à noter que seul le nom est vérifié. Le reste de la définition de l'objet n'est pas pris en compte.

Les *triggers* sur événement se voient ajouter la possibilité d'être exécutés suite à une modification de la définition d'une table (commande **ALTER TABLE**). De même, ils sont exécutés sur les commandes **COMMENT**, **SECURITY LABEL** et **GRANT**/**REVOKE** pour les objets bases de données.

Conclusion

La version 9.5 de PostgreSQL améliore encore fortement sa facilité d'utilisation et d'administration. Elle apporte des fonctionnalités attendues depuis longtemps comme le **UPSERT**, des fonctionnalités qui n'avaient pas d'équivalent comme le **SKIP LOCKED**, et des fonctionnalités qui permettent de gérer de gros volumes de données, notamment avec les nouveautés OLAP. Le prochain article parlera aussi de **TABLESAMPLE** ainsi que de quelques fonctionnalités qui devraient faciliter la venue du *sharding* avec tout le travail effectué sur la norme SQL/MED. Et enfin, on abordera quelques nouveautés pour les DBA. ■

Ce document est la propriété exclusive de Johann Locatelli(johann.locatelli@businessdecision.com)

JE SUIS PAS ANALYSTE !

Jean-Pierre TROLL [« Humeuriste »]

J'ai plus d'abonnements au Gartner, Forrester et autres cabinets d'analystes. Enfin, ma boîte me donne plus les accès à cette si substantifique moelle de réflexion, d'analyse, de synthèse, de futurologie ! Ça tombe bien, car ça commençait à fortement me soûler toute cette logorrhée marketing reprise la bouche en cœur par mes collègues, mes managers, mes clients, ma copine, mon chat... (je n'ose ni n'arrive à mettre d'ordre dans cette liste !)

Mots-clés : Humeur, Jean-Pierre Troll, Coup de gueule, Analyste

Résumé

Le retour de la revanche de Jean-Pierre Troll. Après une mise au vert, la démangeaison du coup de gueule me reprend. Un peu comme un éternuement que j'essayais de contenir... Pour ce retour, je propose une analyse alternative de la transformation digitale. Plutôt que les technologies, je propose de se concentrer sur les usages. Et la convergence et l'interrelation de quatre changements de paradigmes majeurs dans les usages donnent ce que je baptise le « Plexus » : hyperconnexion, consumérisation, commoditisation et intermédiation. Ok, « *Beam me Mr Spock !* ».

Veni, Vidi et Re-Veni !

Pourquoi ce retour à la romaine? Déjà j'aime pas le concept de *Vici* : César et tout plein d'autres empereurs ne me feront pas mentir. Je préfère m'en tenir aux postures liées à l'observation. La victoire ? De quoi ? Sur qui ? Pff...

Plus sérieusement, j'avais besoin de me mettre « au ver » (non c'est pas du surf labellisé COP21 !), de me distancier par rapport à ce monde en chamboulement et surtout retrouver mes racines et mes envies. Bon c'est fait ! Y'en a marre du Cantal, des chèvres et du tralala !!! Je veux me défouler, comme tout le monde.

Alors faut avouer que Tristan fait du bon boulot avec sa rubrique « Humeur ». Respect ! Étant p'tet un peu à l'origine de ladite rubrique, je vous propose - oh contradicteurs adorés - d'y ajouter (de temps en temps = modèle open source c'est prêt quand c'est prêt ! = une courgette est meilleure poussée dans le sol) ma pointe de cynisme.

Et, pour conclure, y'en a plus marre d'en avoir marre !!!

Je dis : on va bien se marrer !! Car y'a de quoi faire avec ce que j'observe : beaucoup de grand n'importe quoi, du pré-digéré, du détruit au micro-onde marketing...

Bref une année 2016 qui s'annonce plutôt caustique !

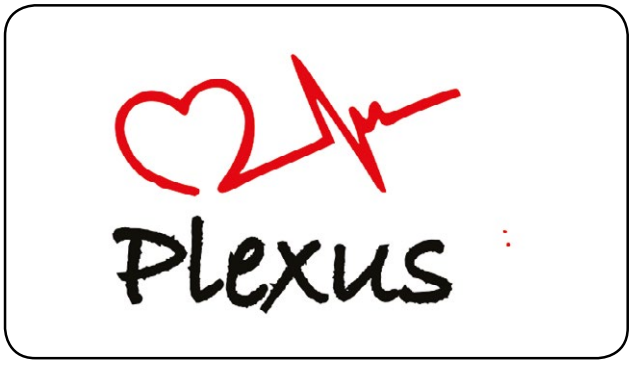
Mon retour n'est pas lié au départ de mon maître Jean-Pierre Coffe. Un grand respect à toi Jean-Pierre pour m'avoir inspiré. Mieux vaut vivre sa vie avec goût, sinon c'est juste de la merde ! Merci pour tout.

Ce document est la propriété exclusive de Johann Locatelli(johann.locatelli@businessdecision.com)

J'ai encore en tête le Nexus du Gartner... c'est probablement dépassé. Souvenez-vous y'a encore deux ans c'était : « le Nexus des Forces » est la convergence et le renforcement mutuel du social, de la mobilité, du *cloud* et de l'information, au service de nouveaux scénarios métiers... (bon y'a aussi plein d'autres Nexus, comme le smartphone de Google, le dépôt d'artéfacts de Sonatype, voire même le livre d'Henry Miller...)

Ça c'était il y a plusieurs années, désormais les analystes futurologues sont sûrement désormais au niveau des voitures en mode « Machine Learning » ou la réalité augmentée au service de la guerre propre... qu'en sais-je ? Rien ! J'ai plus d'abonnement. Fichtre ! Il me faut donc réfléchir par moi-même...

Mon souci c'est que beaucoup parlent de technologies disruptives (et donc futur *mainstream*), mais ils oublient les usages, les tendances qu'elles potentialisent. Donc aujourd'hui à mon tour de donner mon avis sur le monde qui m'entoure, celui que je connais le mieux : celui de l'IT ! Je sors des cabinets (d'analystes) et dis : y'en a marre d'avoir du prêt-à-penser !!! Nous aussi on peut réfléchir, même sans être rémunérés à chaque fois qu'on sort un *buzzword* (celui-là au scrabble c'est bingo !).



D'où mon Nexus à moi. Tadam... voici le Plexus :

- Hyperconnexion ;
- Consumérisation ;
- Commoditisation ;
- Intermédiation.

Bon il m'en fallait quatre également. J'ai pas trouvé un mot bien franglais pour le dernier, pardonnez-moi. HCCI ça sonne pas terrible, Hyperus ça sonne trop Dan Simmons, bref Plexus c'est ce que j'ai de mieux pour le moment. Bon après si vous avez vraiment des élans lyriques on pourra toujours se monter une plateforme de *SMartKeting* (ou SMK pour les gens « in »). Vous connaissez pas ? Normal, je viens de créer le mot. L'idée derrière ? Pfff ! Qui s'en fiche ?

Je vais expliciter ma vision de chacun de ces termes : ils guideront certains de mes prochains coups de gueule. C'est pour vous allécher et en même temps me motiver. Ben oui j'ai besoin d'un minimum de motivation ! Après des années à m'écorcher la voix dans le vide (le vrai, là où y'a pas de son, pas celui de Star Wars) ma fille me traite désormais de « vieux dinosaure » : « dino » pourquoi pas, mais vieux... j'abandonne.

1 Hyperconnexion

C'est à la mode de parler d'objets connectés. Ben oui, connectez-moi mon four, ma fourchette, ma montre...

Alors oui c'est vrai que retrouver sa voiture, son chien ou sa famille par géolocalisation c'est pratique, mais bon, se faire poser des implants pour mieux être connectés c'est pas un peu du grand n'importe quoi quand même ?! Vous voulez pas les commander à distance non plus ? Elle est où la bonne relation avec son jouet, son ami, les siens ? À travers les serveurs de Google ! Et quand ils (mettez qui vous voulez derrière je m'en fiche) décideront de vous déconnecter qu'allez vous faire, devenir, redevenir ?

Ben parce que ceux qui vous connectent soi-disant, ils vous connectent pas à votre vie en la rendant plus cool et facile, mais à eux ! Ils vous vampirisent vos données, vos relations, bref... votre vie !

Alors, viendrons peut-être - ben oui c'est prêt que quand c'est bon ! - des coups de gueule sur le mobile, l'instantané, l'e-ce-que-vous-voulez !

2 Consumérisation

Je ne sais pas si ce mot est (déjà) dans le dictionnaire. En français y'a bien « consumérisme », mais la consumérisation est pour moi la dérivée de cela. C'est un autre ordre de grandeur. Le passage à l'hyperconsommateur est concrétisé grâce à l'effet de masse couplé à l'hyperconnection. Et voilà très vite le consommateur consommé ! Le marketing est aujourd'hui roi sur les masses hyperconnectées. Marketing économique, politique ou idéologique. Vive le ciblage et le décisionnel de nos supermarchés (ah... c'est vrai faut dire Big Data aujourd'hui sinon on est trop naze...).

Mais j'observe aussi la subversion subtile des nouveaux modes de consommation de masse envers le modèle client-

Ce document est la propriété exclusive de Johann Locatelli(johann.locatelli@businessdecision.com)

fournisseur : « On veut du comme à la maison quoi ! Du sexy, du simple, du qui en jette ! Oui... même si c'est pour faire tourner une centrale nuclééaaiiire ! Quel dinosaure ce Jean-Pierre, j'te jure ! »

Bref nous passons réellement d'utilisateurs à consommateurs, le tout sous les sourires des marketeux et autres propagandistes roués aux techniques de manipulation et de la création de nouveaux besoins (ben oui, pourquoi manger ? Je préfère passer au niveau 26 de « NoQuest », quitte à mourir IRL !).

Y'a tant à gueuler sur ce sujet que je me réserve pour d'autres billets ! Je m'énerve tout seul rien qu'à y penser...

3 Commoditisation

Bon OK à l'ère des révolutions agricoles ou industrielles la démocratisation des biens de production et de consommation a donné par exemple des outils tels que la tronçonneuse, la machine à laver ou encore la voiture. Mais dans le domaine de l'IT c'est vraiment la faute des logiciels libres appuyés plus tard par l'open source ! C'est rigolo cette histoire, car après avoir tonitrué et traité les libristes de communistes doux rêveurs et barbus, les cols blancs bien pensants nous ont généralisé l'open à toutes les sauces ! Open Data (filez-nous vos données, car nous au moins on sait comment gagner de l'argent avec !), Open Innovation (filez-nous vos idées, car nous au moins on sait comment gagner de l'argent avec !)... Bref, des déclinaisons plus récentes de l'open source (filez-nous votre code, car nous au moins on sait comment gagner de l'argent avec !).

Et quand y'a plus d'argent à gagner, ben cela devient une commodité ! Ben oui madame, on a essoré tout ce qu'on pouvait avec les OS propriétaires, les serveurs d'applications J2E super chers (et bientôt les SGBD obsolètes...).

Je détaillerai (peut-être) le phénomène et ce que cela peut permettre : ben déjà le software, Internet, l'élastique dans notre métier voire le *Cloud* (et non le *[Clood]* comme disent ceux qui n'ont jamais que lu le mot sans comprendre ce qu'il tente de dissimuler, probablement dans des analyses payées Ruby sur l'ongle <jpt>ok, celle-là était facile, mais bon les analystes me hérissent le poil</jpt>).

4 Intermédiation

Mon préféré, car c'est un rayon du Plexus vieux comme le monde : plutôt que de produire ou de consommer pour produire, je vais aider les producteurs et les consommateurs à échanger, sans trop rien produire à part cette intermédiation. On a créé comme ça l'argent, les banques, les transactions bancaires, les agences immobilières, etc. Et maintenant on a Uber ou AirBnB. L'idée est là même : disrupter – c'est-à-dire parasiter - le flux de consommation.

Et au passage je ne suis pas trop d'accord avec le concept de NATU (Netflix / AirBnB / Tesla Motors / Uber) qui seraient les putschistes des GAFA (Google / Apple / Facebook / Amazon). Apparemment, les analystes ne savent compter que jusqu'à quatre... Bref ! Netflix déploie son Plexus en ajoutant à l'intermédiation de DVD puis de *Pay-Per-View*, de la consumérisation via la production de contenus originaux à consommer en mode boulimique (genre tous les épisodes de la série d'un coup, c'est pas grave demain j'ai juste un comité de projet !). Et Tesla fabrique quand même des voitures... Alors qu'Uber ou AirBnB se positionnent bien en parasites : déployant plutôt le Plexus sur l'hyperconnexion pour le premier (un VTC vite si vous êtes connectés) ou la commoditisation pour le dernier (une location ben c'est juste une location, pourquoi y ajouter des services chers et inutiles comme la sécurité, le confort garanti ou autres trucs *has been* d'hôtel !).

L'intermédiation peut également être mafieuse. Avec toutes sortes de services achetables dans le Dark Web si vous voulez espionner ou nuire à une personne ou une organisation. Je ne m'étendrai pas plus ici sur cet aspect pour me réserver sur un futur article (ou pas) sur les mouvements de masses que les plateformes d'intermédiation favorisent (crowd-ce-que-vous-voulez).

Conclusion

Bref... En relisant mon coup de gueule, je me suis surpris à hurler « Y'en a marre des papiers écrits à la va-vite, des analyses sponsorisées à la mord moi le nœud ! » et à me dire que j'avais écrit le même genre de délires, mais sans revendiquer le statut de leader d'opinion. J'ai failli jeter mes notes dans les cabinets (pas d'analystes ceux-là... encore que) et tirer la chasse.

Puis je me suis ressaisi lorsque Clémentine (@clem_Troll) m'a dit de sa voix orangée : « Chéri, tu veux pas qu'on passe un week-end à Malaga ? J'ai trouvé une super maison sur AirBnB avec connexion fibre et Netflix déjà payé ! En plus y'a même un service de VTC qui vient te chercher en voiture électrique avec des bonbons plein les portes et de la Cristalline gratuite !!! ». ■

Ce document est la propriété exclusive de Johann Locatelli(johann.locatelli@businessdecision.com)

ACTUELLEMENT DISPONIBLE

HACKABLE N°12 !

CRÉEZ VOTRE BORNE D'ARCADE !

...MINIATURE À BASE DE RASPBERRY PI

NE LE MANQUEZ PAS

CHEZ VOTRE MARCHAND DE JOURNAUX ET SUR :

www.ed-diamond.com

Ce document est la propriété exclusive de Johann Locatelli(johann.locatelli@businessdecision.com)

PEUT-ON VRAIMENT CALCULER AVEC UN ORDINATEUR : LES OPÉRATIONS

Florent LANGROGNET [Ingénieur de Recherche CNRS - Laboratoire de Mathématiques de Besançon]

Nous avons vu le mois dernier comment étaient représentés les réels sur ordinateur. Il est temps maintenant d'utiliser ces réels (les flottants) pour calculer. Quels sont les mécanismes mis en œuvre lors de chaque opération ? À quelle(s) étape(s) des erreurs de précision peuvent-elles apparaître ? L'ordre des calculs a-t-il une importance sur la qualité des résultats ? Vous trouverez les réponses à toutes ces questions (et bien d'autres) dans les pages de cet article.

Mots-clés : Calcul, Réels, Flottants, Précision, Arithmétique, Norme

Résumé

Après avoir expliqué comment les réels étaient représentés sur ordinateur, nous allons montrer dans les détails comment se déroulent deux opérations (l'addition et la soustraction). Ainsi, nous verrons les causes des erreurs de précision et quelles sont les pistes pour les contenir. Nous regarderons ensuite ce qui se passe avec les autres opérations et conclurons.

La lecture de la première partie de cet article vous a convaincu (si besoin était) que les résultats des calculs sur ordinateur n'étaient que des approximations (plus ou moins bonnes) des vraies valeurs. Maintenant que nous savons comment sont stockés ces réels, nous allons pouvoir les utiliser pour calculer !

Calculer sur ordinateur, c'est d'abord arrondir, c'est-à-dire choisir un représentant (dans le monde des flottants) pour chaque réel. Je présenterai, au début de cet article, les outils permettant d'arrondir puis je détaillerai les mécanismes de deux opérations (l'addition et la soustraction) en indiquant les situations dans lesquelles le résultat est inexact, voire aberrant. Je montrerai aussi comment, en réordonnant l'ordre des opérations, la précision peut être améliorée. Enfin, j'évoquerai plus globalement le comportement de toutes les opérations et la norme IEEE-754 qui a permis de limiter les dérives.

1 Le besoin d'arrondir

Nous avons vu que tous les nombres réels n'étaient pas représentables en machine. Lors de chaque opération (ou presque) on devra arrondir la vraie valeur par un nombre représentable proche, appelé l'arrondi. Les réels représentables sur ordinateur sont appelés réels flottants (ou simplement les flottants).

La première opération qui est concernée est l'affectation. En effet, il est probable que, dès l'affectation, on soit obligé

Ce document est la propriété exclusive de Johann Locatelli(johann.locatelli@businessdecision.com)

de choisir un flottant proche de la vraie valeur pour la représenter, et ainsi, de commettre une première erreur de précision. La figure 1 représente les choix qui s'offrent pour stocker un réel (non représentable exactement sur ordinateur).

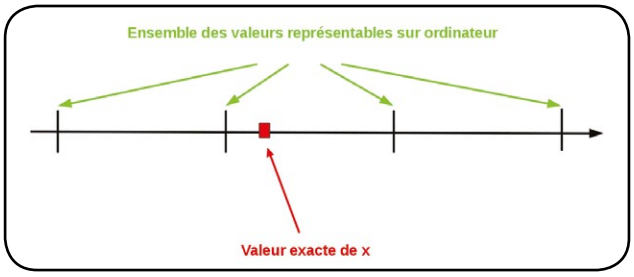


Fig. 1 : Comment choisir le meilleur représentant pour une valeur **x**, *non représentable sur ordinateur ?*

Pour effectuer ce choix, la norme IEEE-754 propose quatre modes d'arrondi :

- vers $+\infty$

  (noté RU pour *Round Up*) permet de choisir le flottant le plus proche en se dirigeant vers $+\infty$

- vers $-\infty$

  (RD : **Round Down**) permet de choisir le flottant le plus proche en se dirigeant vers $-\infty$

- vers **0** (RZ : *Round towards Zero*) permet de choisir le flottant le plus proche en se dirigeant vers **0**.

- au plus proche (RN : *Round to Nearest*) permet de choisir le flottant le plus proche. C'est le mode d'arrondi par défaut.

Ces quatre modes d'arrondis sont schématisés par la figure 2.

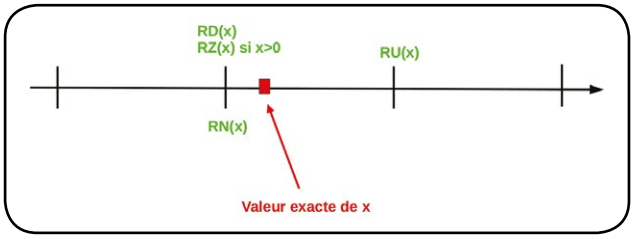


Fig. 2 : Les quatre modes d'arrondis pour représenter **x**, *réel positif.*

On dispose ainsi d'un premier outil pour choisir comment on peut arrondir un réel lors de n'importe quelle opération. À titre d'exemple, le programme **arrondi_affectation.cpp** (ci-dessous) permet de vérifier l'effet d'un changement de mode d'arrondi pour l'affectation.

```
#include <iomanip>
#include <iostream>
#include <fenv.h>

void affectation(){
  float a;
  std::cout<<"Entrer la valeur de a"<<std::endl;
  std::cin>>a;
  std::cout<<" a : "<<a<<std::endl;
}

int main(){
  std::cout<<std::setprecision(10);

  std::cout<<"mode d'arrondi RD"<<std::endl;
  std::cout<<"----------------"<<std::endl;
  fesetround(FE_DOWNWARD);
  affectation();

  std::cout<<std::endl<<"mode d'arrondi RU"<<std::endl;
  std::cout<<"----------------"<<std::endl;
  fesetround(FE_UPWARD);
  affectation();

  std::cout<<std::endl<<"mode RN"<<std::endl;
  std::cout<<"-------"<<std::endl;
  fesetround(FE_TONEAREST);
  affectation();

  return 0;
}
```

Note

Le changement du mode d'arrondi s'effectue avec l'appel à **std::fesetround()**.

L'exécution de ce programme avec la valeur **a=0,1** donne les résultats ci-dessous, conformes aux attentes, compte tenu des représentations possibles pour cette valeur sur ordinateur (schématisées par la figure 3, page suivante).

```
mode d'arrondi RD
----------------
Entrer la valeur de a
0.1
 a : 0.09999999403

mode d'arrondi RU
----------------
Entrer la valeur de a
0.1
 a : 0.1000000015

mode RN
-------
Entrer la valeur de a
0.1
 a : 0.1000000015
```

Ce document est la propriété exclusive de Johann Locatelli(johann.locatelli@businessdecision.com)

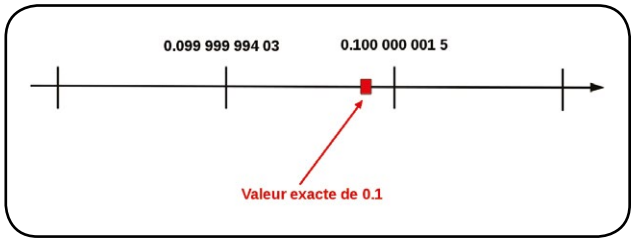


Fig. 3 : Arrondis pour représenter 0,1 sur ordinateur.

2 Additionner des réels

2.1 Principes

L'addition de deux réels **A** et **B** (dont le résultat est un réel **S**) se décompose en 3 étapes :

- alignement des mantisses si les exposants de **A** et **B** sont différents ;
- addition des mantisses ainsi alignées ;
- renormalisation de la mantisse de **S** si elle n'est pas normalisée (et gestion du bit implicite).

2.2 Exemple : 9,5 + 1,75

Nous allons maintenant détailler le mécanisme d'addition de deux réels représentables sur ordinateur : **9,5** et **1,75**. Ces nombres sont représentés en simple précision selon la figure 4.

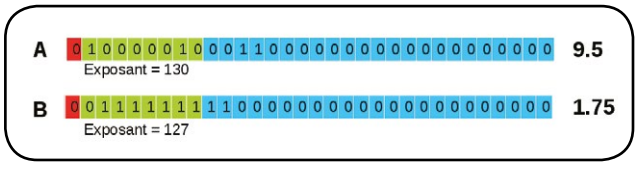


Fig. 4 : Représentation (simple précision) des nombres 9,5 et 1,75 sur ordinateur avec le bit de signe (en rouge), les bits représentant l'exposant décalé (en vert) et la mantisse (en bleu).

La première étape consiste à aligner les mantisses pour que les exposants soient les mêmes. On réécrit donc le nombre le plus petit (ici **1,75**) pour que son exposant soit le même que celui du plus grand (**130**). Il faut donc ajouter **3** à l'exposant et décaler vers la droite la mantisse de **1,75**. Lors de ce décalage (représenté sur la figure 5) :

- il ne faut pas oublier de réintroduire le bit implicite (qui a pour valeur **1**) ;
- après avoir réintroduit le **1** (bit implicite), on introduit des **0**.

Ce décalage a pour conséquence de faire « sortir » des bits de la mantisse. Ces bits sont perdus ; on peut donc perdre de l'information à ce stade.

Il faut bien noter que, sous cette écriture, **1,75** n'est plus sous sa forme normalisée.

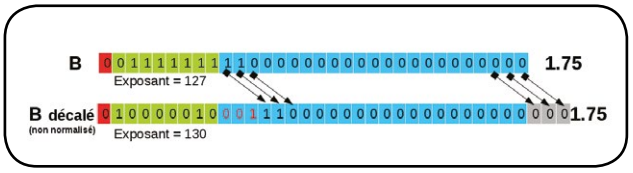


Fig. 5 : Décalage lors de l'alignement des mantisses. Les bits qui sortent (et qui sont donc perdus) sont représentés en gris.

Ainsi réécrit, l'exposant de **1,75** est le même que celui de **9,5**. Il ne nous reste plus qu'à additionner les mantisses. Il s'agit simplement d'une addition en binaire schématisée par la figure 6.

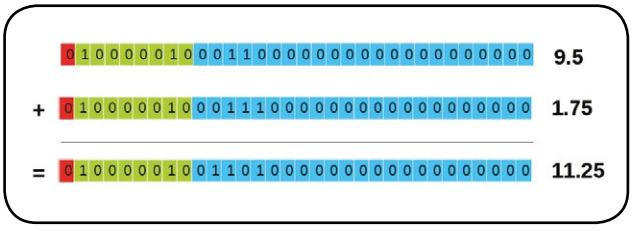


Fig. 6 : Somme des mantisses.

Ce calcul mène donc à l'écriture **0 10000010 01101000000000000000000**, qui correspond bien à **11,25**. Ce calcul est donc juste...

Note

Le bit implicite est géré dans le cas de l'addition de **9,5+1,75** de manière transparente : le bit implicite de la somme est la somme des bits implicites (**1** pour **9,5** et **0** pour **1,75** après décalage) ; il vaut donc bien **1** et le résultat est bien écrit sous sa forme normalisée avec un bit implicite (qui vaut **1**). Ce n'est pas le cas lors de l'addition de nombres proches qui nécessite alors un traitement supplémentaire.

2.3 Perte de précisions

Lors de l'alignement des mantisses, nous avons vu que des bits « sortent » (on ne stocke que 23 bits de mantisse en simple précision). Dans le cas précédent, c'est sans consé-

Ce document est la propriété exclusive de Johann Locatelli(johann.locatelli@businessdecision.com)

quence, car tous ces bits valaient **0**. En revanche, si certains de ces bits n'étaient pas nuls, ce décalage provoquerait une perte de précision.

Pour s'en convaincre, nous allons effectuer l'addition de **9,5** avec le nombre juste supérieur à **1,75**, soit **1,7500001192092896** (dont le bit de poids faible de la mantisse est **1**) représenté par la figure 7.

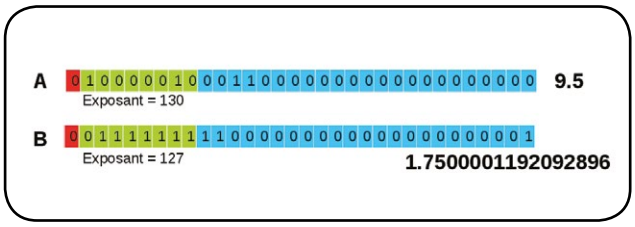


Fig. 7 : Représentation des nombres 9,5 et 1,7500001192092896.

En alignant les mantisses (voir figure 8), la valeur du bit de poids faible (**1**) est perdue. Le nombre ainsi représenté (sous une forme non normalisée) correspond à **1,75** et non plus à **1,7500001192092896** !

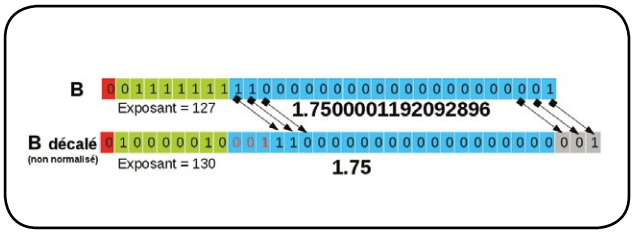


Fig. 8 : Décalage lors de l'alignement des mantisses : 1,7500001192092896 devient 1,75 !

L'addition se termine comme celle de **9,5** et de **1,75** et l'on obtient : **9,5 + 1,7500001192092896 = 11,25**.

Il est important toutefois de noter que **11,25** est la meilleure (la plus proche) représentation de la vraie valeur (**11,2500001192092896**) qui n'est pas représentable sur ordinateur.

Voici donc le premier exemple de perte de précision lors d'une addition. On peut alors facilement comprendre que cette perte peut être beaucoup plus importante si de nombreux bits valant **1** sortent lors du décalage. Cette situation peut se rencontrer lorsque l'on additionne deux réels d'ordres de grandeur très différents. On peut même arriver au cas extrême où toute l'information du plus petit des deux nombres est perdue : c'est l'absorption que nous allons illustrer sur l'addition de **1 000 000** et de **0,01171875** (représentés sur la figure 9).

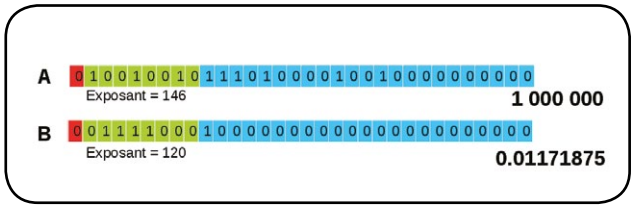


Fig. 9 : Représentation des nombres 1 000 000 et de 0,01171875.

Lors de l'alignement des mantisses (décalage de 26 bits vers la droite), tous les bits de la mantisse sortent et toute l'information est perdue comme le montre la figure 10.

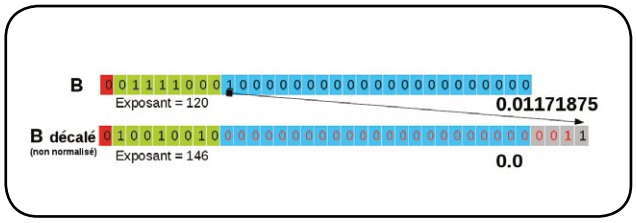


Fig. 10 : Décalage lors de l'alignement des mantisses : 0,01171875 devient 0 !

La conséquence est alors prévisible : on réalise (voir figure 11) l'addition de **1 000 000** avec **0**.

On obtient donc : **1 000 000 + 0,01171875 = 1 000 000** !

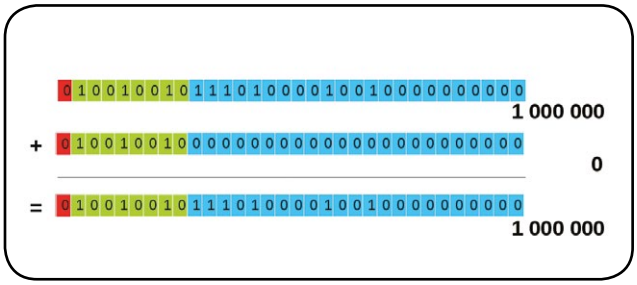


Fig. 11 : Addition de 1 000 000 et 0.

3 Soustraire des réels

3.1 Principes et exemple

Pour réaliser une soustraction entre deux réels, le principe est identique à celui de l'addition : alignement des mantisses, soustraction des mantisses et éventuellement renormalisation du nombre ainsi obtenu.

Prenons l'exemple de **9,875 – 1,75**. Ces deux nombres sont représentés selon la figure 12.

Ce document est la propriété exclusive de Johann Locatelli(johann.locatelli@businessdecision.com)

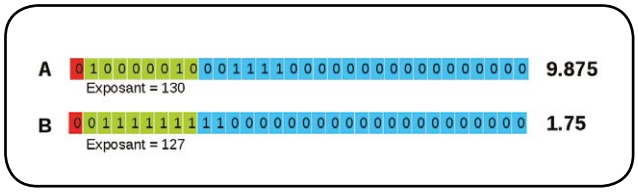


Fig. 12 : Représentation des nombres 9,875 et 1,75.

La première étape (alignement des mantisses) conduit à réécrire **1,75** pour que son exposant soit le même que celui de **9,875**, soit **130** (voir figure 13).

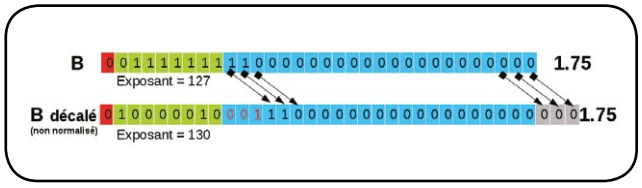


Fig. 13 : Décalage pour aligner les mantisses.

On réalise ensuite la soustraction des mantisses (voir figure 14).

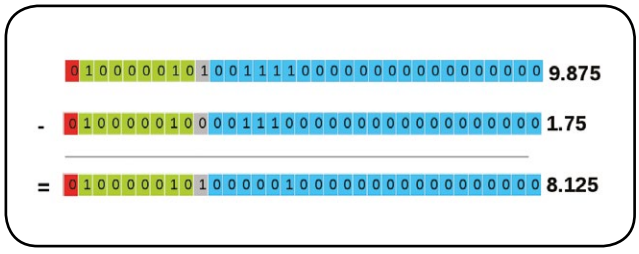


Fig. 14 : Soustraction : 9,875 − 1,75.

Dans ce cas (sans perte de précision) on obtient la valeur exacte lors du calcul soit **9,875 − 1,75 = 8,125**.

3.2 Perte de précisions

La soustraction, comme l'addition, est susceptible de provoquer des pertes de précision. C'est le cas lorsque l'on soustrait deux nombres proches. Nous allons tout d'abord traiter un exemple (**9,75 - 9,5**) pour lequel il est inutile d'aligner les mantisses puisque les exposants sont identiques (voir figure 15).

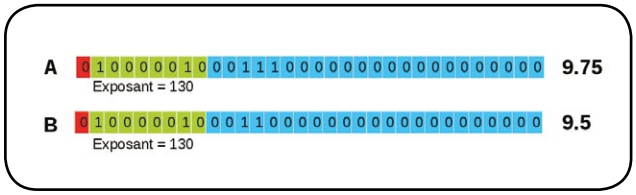


Fig. 15 : Représentation des nombres 9,75 et 9,5.

Dans ce cas, il suffit de soustraire les mantisses pour obtenir le résultat comme le montre la figure 16. Cette figure fait également apparaître la gestion du bit implicite : contrairement au cas précédent où cette gestion était transparente (on avait **1** pour le bit implicite du plus grand nombre et **0** pour le plus petit), ici le bit implicite est **1** pour les deux nombres (puisque l'on n'a pas réalisé de décalage). Le bit implicite du résultat est donc égal à **0**.

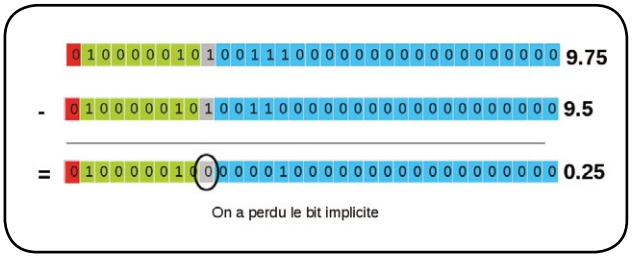


Fig. 16 : Soustraction des mantisses laissant apparaître un 0 à la place du bit implicite.

Le résultat obtenu est juste, mais il n'est pas sous son écriture normalisée (avec un bit implicite égal à **0** !). Il convient donc de le normaliser pour faire réapparaître en gris le bit implicite (**1**) en décalant la mantisse vers la gauche pour que le premier bit non nul prenne la place du bit implicite. Lors de ce décalage, on a besoin d'introduire des bits (par la droite) pour remplacer ceux qui sont partis à gauche. La valeur de ces bits est inconnue, et arbitrairement considérée comme étant nulle.

Dans notre exemple (voir figure 17), on a besoin de décaler la mantisse de 5 bits vers la gauche et donc de soustraire **5** à la valeur de l'exposant pour avoir un nombre à l'écriture normalisée.

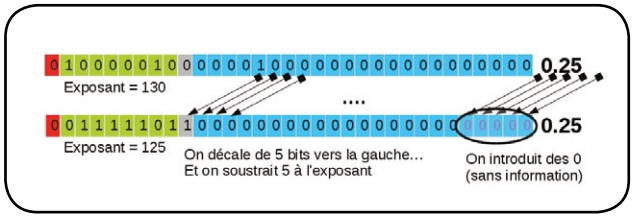


Fig. 17 : Renormalisation avec introduction de valeurs arbitraires.

Lors de ce calcul, on obtient bien : **9,75 − 9,5 = 0,25**. Mais alors d'où peuvent venir les pertes de précision ?

Des pertes de précision peuvent apparaître lorsque le vrai résultat n'est pas représentable sur ordinateur. En effet, rappelons que la soustraction (comme l'addition) de deux nombres représentables sur ordinateur peut être un nombre non représentable sur ordinateur.

Ce document est la propriété exclusive de Johann Locatelli(johann.locatelli@businessdecision.com)

PROFESSIONNELS DES TICE, COLLECTIVITÉS, ÉCOLES D'INGÉNIEURS, UNIVERSITÉS, R & D, ENSEIGNANTS, ...

LINUX MAGAZINE / FRANCE

PROFESSIONNELS !

DÉCOUVREZ NOS OFFRES D'ABONNEMENTS ...

...EN VOUS CONNECTANT À L'ESPACE DÉDIÉ AUX PROFESSIONNELS SUR :

www.ed-diamond.com

PDF COLLECTIFS PRO

Prix TTC en Euros / France Métropolitaine

OFFRE	ABONNEMENT	1 - 5 lecteurs Réf	1 - 5 lecteurs Tarif TTC	6 - 10 lecteurs Réf	6 - 10 lecteurs Tarif TTC	11 - 25 lecteurs Réf	11 - 25 lecteurs Tarif TTC
PROLM2	11n° GLMF	☐ PRO LM2/5	260,-	☐ PRO LM2/10	520,-	☐ PRO LM2/25	1040,-
PROLM+2	11n° GLMF + 6n° HS	☐ PRO LM+2/5	472,-	☐ PRO LM+2/10	944,-	☐ PRO LM+2/25	1888,-

PROFESSIONNELS :
N'HÉSITEZ PAS À NOUS CONTACTER POUR UN DEVIS PERSONNALISÉ PAR E-MAIL :
abopro@ed-diamond.com
OU PAR TÉLÉPHONE :
03 67 10 00 20

ACCÈS COLLECTIFS BASE DOCUMENTAIRE PRO

Prix TTC en Euros / France Métropolitaine

OFFRE	ABONNEMENT	1 - 5 connexion(s) Réf	1 - 5 connexion(s) Tarif TTC	6 - 10 connexions Réf	6 - 10 connexions Tarif TTC	11 - 25 connexions Réf	11 - 25 connexions Tarif TTC
PROLM+3	GLMF + HS	☐ PRO LM+3/5	267,-	☐ PRO LM+3/10	534,-	☐ PRO LM+3/25	1068,-
PROA+3	GLMF + HS + LP + HS	☐ PRO A+3/5	297,-	☐ PRO A+3/10	594,-	☐ PRO A+3/25	1188,-
PROH+3	GLMF + HS + LP + HS + MISC + HS + OS	☐ PRO H+3/5	447,-	☐ PRO H+3/10	894,-	☐ PRO H+3/25	1788,-

Les abréviations des offres sont les suivantes : LM = GNU/Linux Magazine France HS = Hors-Série LP = Linux Pratique OS = Open Silicium

SÉLECTIONNEZ VOTRE OFFRE DANS LA GRILLE CI-DESSUS ET RENVOYEZ CE DOCUMENT COMPLET À L'ADRESSE CI-DESSOUS !

Voici mes coordonnées postales :	
Société :	
Nom :	
Prénom :	
Adresse :	
Code Postal :	
Ville :	
Pays :	
Téléphone :	
E-mail :	

☐ Je souhaite recevoir les offres promotionnelles et newsletters des Éditions Diamond.
☐ Je souhaite recevoir les offres promotionnelles des partenaires des Éditions Diamond.

En envoyant ce bon de commande, je reconnais avoir pris connaissance des conditions générales de vente des Éditions Diamond à l'adresse internet suivante : **boutique.ed-diamond.com/content/3-conditions-generales-de-ventes** et reconnais que ces conditions de vente me sont opposables.

LINUX MAGAZINE / FRANCE

Les Éditions Diamond
Service des Abonnements
10, Place de la Cathédrale
68000 Colmar – France
Tél. : + 33 (0) 3 67 10 00 20
Fax : + 33 (0) 3 67 10 00 21

Vos remarques :

.....

.....

Ce document est la propriété exclusive de Johann Locatelli(johann.locatelli@businessdecision.com)

Pour bien comprendre ce cas de figure, nous allons prendre un nouvel exemple avec la soustraction de deux « voisins » (deux nombres qui se suivent dans le monde des flottants) : **9,5 - 9,500000953674316**. La représentation de ces deux nombres ne diffère que sur la valeur du bit de poids faible de la mantisse comme illustré sur la figure 18.

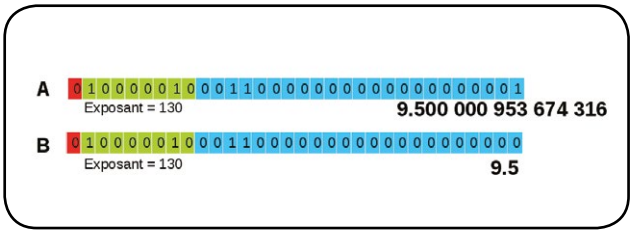


Fig. 18 : Représentation des nombres 9,5 et 9,500000953674316.

Lors de la soustraction des mantisses (voir figure 19), comme dans l'exemple précédent, on a perdu le bit implicite. De plus, la valeur représentée n'est pas le résultat exact de la soustraction, car celui-ci n'est pas représentable. Il est cependant le meilleur représentant (le plus proche).

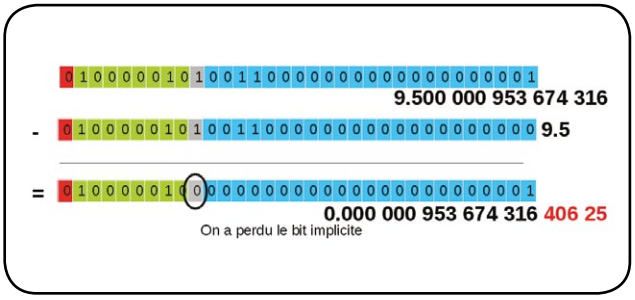


Fig. 19 : Soustraction des mantisses, perte du bit implicite. Les chiffres en rouge indiquent l'erreur de précision lors de cette opération.

La renormalisation de ce nombre (figure 20) produit un décalage de 23 bits vers la gauche.

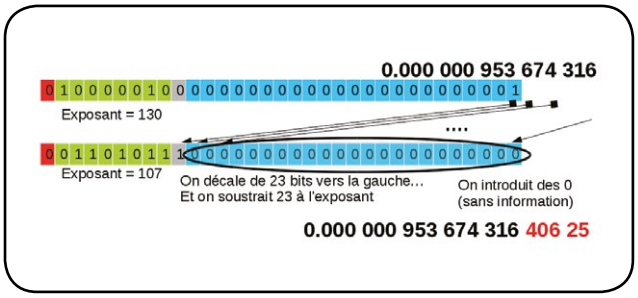


Fig. 20 : Renormalisation.

La perte de précision lors de la soustraction de deux nombres proches est appelée *cancellation*.

4 Cancellation et absorption : le duo explosif

Lors d'une *cancellation*, on introduit arbitrairement des **0** dans la mantisse (car on ne dispose pas de plus de précision sur les opérandes). Si les opérandes sont issus d'un calcul précédent, ils ont pu subir une perte de précision (en cas d'absorption par exemple). Cette perte de précision va alors être réintégrée et amplifiée. Dans ce cas, les bits arbitrairement introduits (des **0**) ne sont pas les bons.

Ainsi une absorption suivie d'une *cancellation* peut conduire à des pertes de précision très importantes et à des résultats très surprenants (et faux).

On se propose de calculer : **(1 000 000 + 0,01171875) – 1 000 000**.

On commence donc par l'addition de **1 000 000** et de **0,01171875** représentés selon la figure 21.

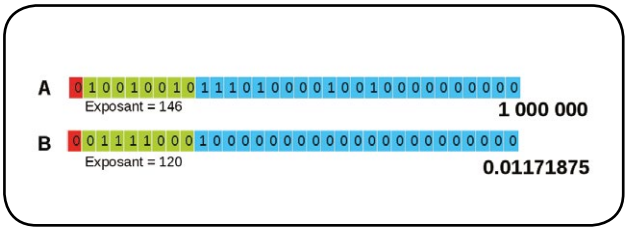


Fig. 21 : Représentation des nombres 1 000 000 et 0,01171875.

Lors de l'alignement des mantisses (voir figure 22), toute l'information de la mantisse de **0,01171875** est perdue. Le nombre ainsi représenté n'est plus **0,01171875** mais **0** !

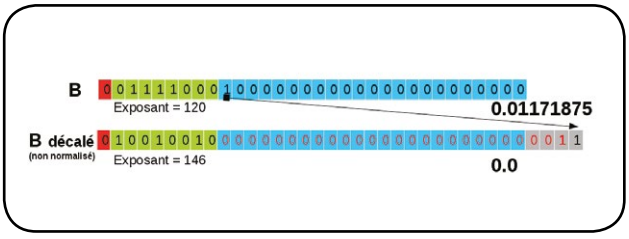


Fig. 22 : Alignement des mantisses et perte de toute l'information.

L'addition des mantisses (figure 23) conduit donc naturellement au résultat **1 000 000 + 0,01171875 = 1 000 000**. Sur cette figure on a également représenté en gris les bits qui ont été perdus lors de l'étape précédente. On remarque donc que si l'on disposait de quatre bits supplémentaires pour la mantisse, le calcul serait exact.

Ce document est la propriété exclusive de Johann Locatelli(johann.locatelli@businessdecision.com)

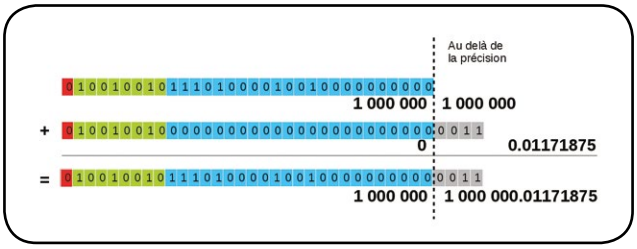


Fig. 23 : Addition des mantisses.

Ensuite, nous procédons à la soustraction de **1 000 000** (figure 24) puis à la renormalisation du nombre représenté (figure 25).

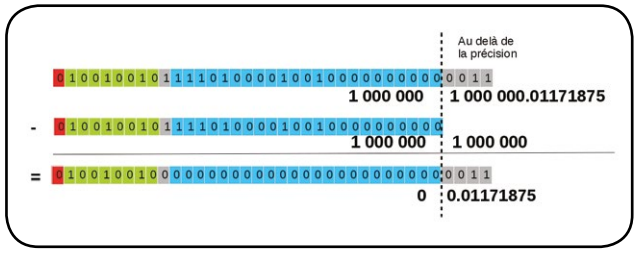


Fig. 24 : Soustraction des mantisses.

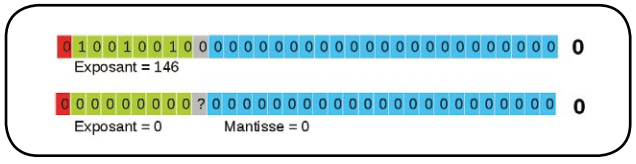


Fig. 25 : Renormalisation.

Au final, le nombre obtenu est **0** alors qu'avec quatre bits supplémentaires le calcul aurait été juste.

On arrive donc à ce résultat étonnant : **(1 000 000 + 0,01171875) – 1 000 000 = 0** !

Ce résultat est bien évidemment faux mais, conduit différemment, ce calcul aurait pu donner le bon résultat. En effet, le calcul **(1 000 000 - 1 000 000) + 0,01171875** donne bien **0,01171875**.

5 Propriétés algébriques, impact de l'ordre des opérations

5.1 Propriétés algébriques

L'exemple précédent montre que certaines propriétés algébriques ne sont pas assurées en arithmétique flottante :

- L'associativité n'est pas respectée (en général) ni pour l'addition, ni pour la multiplication :

$$(a+b)+c \stackrel{?}{=} a+(b+c)$$

$$(a*b)*c \stackrel{?}{=} a*(b*c)$$

- La distributivité n'est pas respectée (en général) entre la multiplication et l'addition :

$$a(b+c) \stackrel{?}{=} ab+ac$$

Seule la commutativité est respectée pour l'addition et la multiplication :

$$a+b=b+a$$

$$a*b=b*a$$

Cette différence essentielle entre l'arithmétique sur des (vrais) réels et l'arithmétique flottante est une des causes d'importantes pertes de précisions. Il faut donc garder à l'esprit que le simple fait de modifier l'ordre des opérations n'est pas sans conséquence. Mais le développeur n'est pas le seul à vouloir changer l'ordre des opérations. Le compilateur, dès qu'on lui demande d'optimiser le code (avec les options **-Ox** pour gcc par exemple), s'autorise à réordonner certains calculs. Bien que mathématiquement équivalents, ces changements peuvent avoir (ont) des conséquences sur la précision de calcul. Dès lors, un compromis doit être trouvé entre précision et rapidité d'exécution.

5.2 Impact de l'ordre des opérations

Pour illustrer l'impact de l'ordre des opérations lors d'un calcul sur ordinateur, prenons l'exemple de la somme des inverses d'entiers calculés de deux façons :

1. Somme de **1** à **N** :

$$\sum_{i=1}^{N} \frac{1}{i}$$

2. Somme de **N** à **1** :

$$\sum_{i=N}^{1} \frac{1}{i}$$

L'exécution en simple précision de ce programme (**somme_des_inverses.cpp**) donne les résultats suivants :

Ce document est la propriété exclusive de Johann Locatelli(johann.locatelli@businessdecision.com)

```
#include <stdio.h>
#include <iomanip>
#include <iostream>

float somme(int N){
  float res = 0.0;
    for (int i=1;i<=N;i++){
    res += (1.0/i);
  }
  std::cout<<"somme de 1 à "<<N<<" : "<<std::setprecision(7)
<<res<<std::endl;

  res = 0.0;
    for (int i=N;i>=1;i--){
    res += (1.0/i);
  }
  std::cout<<"somme de "<<N<<" à 1 :"<<std::setprecision(7)
<<res<<std::endl<<std::endl;
}

int main(){
  somme(100000);
  somme(1000000);
  somme(10000000);
  somme(100000000);
  return 0;
}
```

```
$ ./somme_des_inverses
somme de 1 à 100000 : 12.09085
somme de 100000 à 1 :12.09015

somme de 1 à 1000000 : 14.35736
somme de 1000000 à 1 :14.39265

somme de 1 à 10000000 : 15.40368
somme de 10000000 à 1 :16.68603

somme de 1 à 100000000 : 15.40368
somme de 100000000 à 1 :18.80792
```

Reportés dans le tableau suivant, les résultats obtenus (avec en gras, les chiffres exacts) montrent qu'en effectuant le calcul de **N** à **1**, le calcul est beaucoup plus juste (moins inexact) qu'en l'effectuant de **1** à **N**. Ceci s'explique par le fait que dans le premier cas, pour **N** grand, on finira par additionner des réels d'ordres de grandeur très différents, conduisant à de nombreuses absorptions qui provoquent des pertes de précisions. Dans le deuxième cas, ce phénomène arrive moins souvent et la précision est meilleure.

6 La norme IEEE-754

Dans les années 1980 (et même avant), les constructeurs et les compilateurs étaient déjà confrontés à ces questions : comment stocker les réels, comment calculer ? Mais sans concertation, les réponses de ces acteurs ont été, bien logiquement, différentes. Les calculs menés dans des contextes différents (architecture, compilateur) n'avaient alors aucune chance d'aboutir au même résultat. Un travail de normalisation a alors été engagé et a abouti à l'adoption de la norme IEEE-754 en 1985, révisée en 2008. Cette norme définit un cadre pour, par exemple, le format simple et double précision pour les réels (sur 32 et 64 bits). Il impose également l'arrondi correct pour cinq opérations (**+**, **-**, *****, **/** et $\sqrt{\cdot}$).

Sans entrer dans les détails, l'objectif de l'arrondi correct est d'éviter la propagation des erreurs d'arrondis à chaque opération. Lorsque l'arrondi correct est respecté, le résultat est le même que si on effectuait le calcul en précision infinie (sur de vrais réels) puis on arrondissait ce résultat. Cette norme a, en revanche, laissé une certaine liberté pour toutes les autres opérations ; l'arrondi correct étant simplement recommandé. Ce choix, motivé par des considérations de performances, est à l'origine d'implémentations souvent différentes. Les résultats de calcul (à partir d'un code identique) sont donc encore aujourd'hui dépendants de l'architecture, du compilateur (et de ses options)...

Conclusion

Vous avez désormais les connaissances nécessaires pour comprendre d'où viennent les pertes de précision lors des calculs sur ordinateur. Rappelons tout d'abord que les nombres stockés ne sont qu'un sous-ensemble des nombres réels. Ainsi, chaque opération doit choisir un représentant dans l'ensemble des flottants, avec une perte de précision inévitable. De plus, certaines opérations ne sont pas tenues de choisir le meilleur représentant (le plus proche par exemple). Enfin, certaines optimisations du compilateur engendrent (en réordonnant les calculs) d'autres sources d'imprécision.

N	10^5	10^6	10^7	10^8
Calcul de **1** à **N**	**12,09085**	**14,35736**	**15,40368**	**15,40368**
Calcul de **N** à **1**	**12,09015**	**14,39265**	**16,68603**	**18,80792**
Valeur exacte	**12,09015**	**14,39273**	**16,69531**	**18,99790**

Ce document est la propriété exclusive de Johann Locatelli(johann.locatelli@businessdecision.com)

On comprend ainsi que certains calculs mènent à des résultats bien différents des vraies valeurs et que, dans tous les cas, il convient de s'interroger sur la qualité, la précision des résultats et leur reproductibilité.

Par ailleurs, il est possible d'améliorer la précision de calcul (en utilisant des bibliothèques de calcul ad hoc) ou d'estimer cette précision en utilisant d'autres arithmétiques. L'arithmétique stochastique permet d'estimer par exemple le nombre de chiffres significatifs d'un résultat alors que l'arithmétique par intervalle permet de donner un encadrement de ce résultat. Ces aspects feront peut-être l'objet d'un prochain article.

En attendant, j'espère que celui-ci a atteint un objectif : ne plus considérer le résultat d'un calcul sur ordinateur comme une valeur exacte ! Garder un regard critique... ■

Pour aller plus loin

Les sources d'inspiration de cet article sont multiples et complémentaires :

- La page personnelle de l'auteur (https://lmb.univ-fcomte.fr/Florent-Langrognet) contient quelques exposés et cours sur ce sujet.
- L'article de F. Langrognet (et contributeurs), « *JDEV 2013 - Développer pour calculer : des outils pour calculer avec précision & Comment calculer avec des intervalles* » (*HPC Magazine*, novembre 2013, pp,-51-65) revient sur le traitement de cette thématique lors des JDEV (Journées du DÉVeloppement logiciel) 2013, organisées par le réseau métier de l'enseignement supérieur et de la recherche DevLOG (Développement LOGiciel).
- L'école thématique CNRS « *Précision et reproductibilité en calcul numérique* » (http://calcul.math.cnrs.fr/spip.php?rubrique98) organisée par le réseau métier de l'enseignement supérieur et de la recherche Calcul en 2013.
- Le livre « *Handbook of Floating-Point Arithmetic* » de J.M. Muller, N. Brisebarre, F. de Dinechin, C.-P. Jeannerod, V. Lefèvre, G. Melquiond, N. Revol, D. Stehlé, et S. Torres (Birkhauser, 2010) est un livre de référence pour comprendre en profondeur l'arithmétique flottante.

Ce document est la propriété exclusive de Johann Locatelli(johann.locatelli@businessdecision.com)

LINAGORA SOUTIENT

Linux Professional Institute

la vraie

CERTIFICATION INDÉPENDANTE

Maximisez vos chances de réussite

(taux de satisfaction 95%)

NOS SESSIONS JUILLET 2016

PARIS

LPI 201	4 au 7
LPI 202	18 au 21
Administration TOMCAT	20 au 22

TOULOUSE

LPI 201	4 au 7
LPI 202	18 au 21

formation.LINAGORA.com

DÉCODER UN CODE QR

Nicolas PATROIS [Enseignant dans le secondaire]

Une fois que l'on sait déchiffrer un code QR [1], il faut le décoder... et il y a encore du travail avant de pouvoir lire son contenu.

Mots-clés : Python, Code QR, Programmation orientée objet, Algorithme

Résumé

Nous allons décortiquer le code QR petit à petit pour en extraire les informations utiles à la lecture de son contenu. Le code et les fichiers utiles sont sur GitHub [2].

Je me suis servi des informations données sur *Wikiversity* **[3]** pour la rédaction de cet article.

1 Formats

Nous allons enfin pouvoir lire le contenu du message caché dans la bouillie de pixels de la figure 1. Pour cela, il faut lire le format (niveau de correction et masque) dans la petite zone dédiée. Ensuite, une fois l'image démasquée, on en extrait le message brut : les données en clair et de quoi le corriger. La partie en clair contient en son début les informations précises nécessaires à son interprétation.

Fig. 1 : Le code QR à décoder.

1.1 Lecture du format

On commence par lire (voir figure 2) les deux zones de bits verts dans `form[0]` (celle en haut à gauche) et `form[1]` (celle coupée en deux morceaux en bas et à droite).

Note

La couleur rouge caractérise les cibles d'alignement, le bleu les pointillés, en vert les deux zones de format et en gris les données.

Fig. 2 : Les quatre différentes zones d'un code QR.

Je renuméroterai le code à partir de 01 à chaque section.

```
01:   def formats(self):
02:     if self.esttourne is None:
03:       self.placeim()
04:     self.form=[None,None]
05:     self.form[0]=self.qr[8][:6]+self.qr[8][7:9]+[self.qr[7][8]]
06:     for i in range(5,-1,-1):
07:       self.form[0].append(self.qr[i][8])
08:     self.form[0]=[i^j for (i,j) in zip(self.form[0],masquef)]
09:     self.form[1]=[self.qr[-i-1][8] for i in range(7)]
10:     self.form[1]=self.form[1]+self.qr[8][-8:]
11:     self.form[1]=[i^j for (i,j) in zip(self.form[1],masquef)]
```

Les formats, après lecture, valent `111110110101010`, qui n'est pas interprétable directement. On a besoin du masque binaire `masquef` (qui est dans `qrcodestandard.py`) pour les décoder bit à bit avec un simple `^` (ou exclusif).

```
masquef=[int(i) for i in "101010000010010"]
```

Ce document est la propriété exclusive de Johann Locatelli(johann.locatelli@businessdecision.com)

En effet :

```
 111110110101010
^101010000010010
---------------
 010100110111000
```

Notre zone de format est donc **010100110111000**.

Note

masquef est créé à partir de la chaîne que j'avais la flemme de convertir à la main en une liste. La fonction **zip** de la ligne 11 regroupe en parallèle les éléments de types itérables. Par exemple, **zip([1,2],[3,4],(5,6))** retourne **[(1, 3, 5), (2, 4, 6)]**. Mathématiquement, **zip** retourne la matrice transposée de celle dont les lignes sont en argument et permet de démasquer bit à bit la zone de format. J'aurais pu convertir en entiers chaînes et listes pour utiliser l'opérateur pythonique **^** qu'on emploiera plus loin.

1.2 Vérification du format

On se contente pour le moment de vérifier que **form[0]** et **form[1]** sont identiques, la vérification complète et son code correcteur seront détaillés dans un article ultérieur.

```
01:   def verifformat(self):
02:     if self.form is None:
03:       self.formats()
04:     if self.form[0]!=self.form[1]:
05:       print("Formats différents.",file=stderr)
06:       exit(1)
07:
08:     self.formatok=True
```

Et on dit qu'on est content, chantons sous la pluie...

Dans les images suivantes, la zone verte sera démasquée.

1.3 Démasquage des données

1.3.1 Masque binaire

La partie grise et blanche ne contient pas tout à fait les données brutes, il faut composer avec un masque binaire pour les obtenir.

```
01:   def demasque(self):
02:     if self.formatok is None:
03:       self.verifformat()
04:
05:     self.nivcor=niveaucorrection[tuple(self.form[0][:2])]
06:     self.masque=masques[tuple(self.form[0][2:5])]
```

Les deux premiers bits du format donnent le niveau de correction des erreurs, du niveau L qui corrige le moins d'erreurs au niveau H qui en corrige le plus (les lettres signifient *Low, Medium, Quality* et *High* **[4]**). Ici, notre image utilise le niveau L. **niveaucorrection** et **masques** sont dans **qrcodestandard.py** :

```
niveaucorrection={(0,1):"L",(0,0):"M",(1,1):"Q",(1,0):"H"}
masques={(0,0,0):lambda i,j:(i+j)%2==0,
         (0,0,1):lambda i,j:i%2==0,
         (0,1,0):lambda i,j:j%3==0,
         (0,1,1):lambda i,j:(i+j)%3==0,
         (1,0,0):lambda i,j:(i//2+j//3)%2==0,
         (1,0,1):lambda i,j:(i*j)%2==0 and (i*j)%3==0,
         (1,1,0):lambda i,j:((i*j)%3+i*j)%2==0,
         (1,1,1):lambda i,j:((i*j)%3+i+j)%2==0
         }
```

Le **masque** est donné par les trois bits suivants. **masques** est un dictionnaire où les clés sont des tuples (les bits de rang 2 à 4 de la zone de format) et les valeurs des fonctions anonymes créées *ad hoc*. Oui, Python permet ça tout naturellement. Dans notre cas, **masque** est le troisième de la liste (voir le premier article de cette série **[5]** pour le dessin des huit masques ou la fin de cet article pour le nôtre), il inverse les valeurs des colonnes d'abscisse divisibles par 3, c'est une fonction à deux variables qui retourne un booléen ; **True** s'il faut inverser le bit à la position idoine et **False** sinon.

```
08:     self.dim=len(self.qr)
09:     self.version=(self.dim-17)//4
```

On calcule le nombre de modules en hauteur (et donc en largeur), on en déduit la **version**. Il s'agit bien de la **version** du code qui commence à **1** s'il contient 21 modules et augmente de **1** à chaque quadruplet de modules supplémentaires en hauteur et en largeur. Il n'y a donc pas de code QR de dimension paire ou de 23×23 modules.

1.3.2 Création de la zone à masquer

```
10:     self.gris=griser(self.dim,self.version)
```

La fonction **griser** est dans **qrcodestandard.py** ; elle retourne un tableau de mêmes dimensions que le code QR qui indique la zone dont on doit démasquer le contenu :

Ce document est la propriété exclusive de Johann Locatelli(johann.locatelli@businessdecision.com)

```
def griser(dimension,version):
  tablegris=[[False for _ in range(9)]+[True for _ in
range(dimension-17)]+\
    [False for _ in range(8)] for _ in range(6)]
  tablegris=tablegris+[[False for _ in range(dimension)]]
  tablegris=tablegris+[[False for _ in range(9)]+[True for _ in
range(dimension-17)]+\
    [False for _ in range(8)] for _ in range(2)]
  tablegris=tablegris+[[True for _ in range(6)]+[False]+\
    [True for _ in range(dimension-7)] for _ in
range(dimension-17)]
  tablegris=tablegris+[[False for _ in range(9)]+\
    [True for _ in range(dimension-9)] for _ in range(8)]
```

Les trois premières lignes ajoutent les deux cibles du haut et la zone entre elles deux (la deuxième est la zone de pointillés horizontale). La suivante construit les lignes entre les cibles et la dernière les lignes inférieures avec la cible inférieure à gauche.

Quand **gris[i][j]** vaut **True**, on est dans la zone de données en blanc et gris (et non entre gris clair et gris foncé...). On a commencé par les grandes cibles, les pointillés et les zones de format. Le code aurait été moins lourd en utilisant la multiplication des listes par un scalaire (où **2*[[4,1]]** vaut **[[4,1],[4,1]]**) ; sauf que des copies comme précédemment ou dans **[[]]*2=[[],[]]** sont comme des liens *unix*, modifier l'une des listes internes modifie l'autre ce qu'on veut éviter s'il y a des mini-cibles.

```
  for ci in minicibles[version]:
    for cj in minicibles[version]:
      if (ci,cj) not in {(6,6),(6,max(minicibles[version])),(max
(minicibles[version]),6)}:
        for i in range(ci-2,ci+3):
          tablegris[i][cj-2:cj+3]=[False]*5
```

Pour les petites cibles de 5 modules de côté, le dictionnaire **minicibles** contient les abscisses et les ordonnées de leurs centres en fonction de la version du code QR **[6]**. Il faut veiller à ne pas faire se chevaucher les mini-cibles et les grandes d'où le test au milieu de ce code.

Voici le début du dictionnaire **minicibles**, lui aussi dans **qrcodestandard.py** :

```
minicibles={1:[],
            2:[18],3:[22],4:[26],5:[30],6:[34],
            7:[6,22,38],8:[6,24,42],9:[6,26,46],10:[6,28,50],11:
[6,30,54],12:[6,32,58],13:[6,34,62],
            ...
}
```

Et ainsi de suite jusqu'à 40, vous comprenez pourquoi je coupe, le chef et la PAO aussi.

La suite de la fonction **griser** :

```
if version>=7:
  for i in range(6):
    tablegris[i][-11:-8]=[False]*3
  for i in range(3):
    tablegris[-11+i][0:6]=[False]*6
```

Il s'agit des deux rectangles de 3×6 modules placés entre les pointillés, à gauche contre la cible de droite et au-dessus contre la cible du bas. Ils n'existent que pour les codes QR de **version** supérieure ou égale à 7.

```
return tablegris
```

1.3.3 Démasquage

```
12:     for i in range(self.dim):
13:       for j in range(self.dim):
14:         if self.gris[i][j]:
15:           self.qr[i][j]^=self.masque(i,j)
```

Et enfin, on démasque bit à bit, le résultat est dans la figure 3 (la partie verte est démasquée, observez en bas et en haut **01010** c'est-à-dire **01** suivi de **010**, voir section 1.1).

Fig. 3 : La partie grisée est démasquée.

Vous vous dites qu'on pourrait démasquer tout le tableau sans se soucier du **gris** et vous avez raison. Seulement, le **gris** est nécessaire lors de l'extraction des données brutes puisqu'on ne doit pas lire de bit dans les zones colorées et autant anticiper s'il faut déboguer.

2 Lecture des données

2.1 Extraction des données binaires

2.1.1 L'extraction

Il est temps d'extraire de la zone grise le message binaire et sa correction.

```
01:   def messagebrut(self):
02:     if None in [self.version,self.masque,self.dim,self.
gris,self.nivcor]:
03:       self.demasque()
04:     i,j=self.dim-1,self.dim-1
```

Ce document est la propriété exclusive de Johann Locatelli(johann.locatelli@businessdecision.com)

```
05:     self.code=[]
06:     direction=-1
```

La suite de bits, en-tête, message et correction des deux, est dans **code**, on le découpera plus tard, car ils y sont mélangés.

```
08:     limite=(16*(4+self.version*(8+self.version))-25*(self.
version>=2+self.version//7)**2-(self.version>=7)*(36+self.
version//7*40))//8*8
```

limite est le nombre maximum de bits (message et correction) que peut contenir un code QR de la **version** donnée. Dans notre cas, il peut contenir au maximum 560 bits, soit 70 octets. La formule vient du complément **[7]** et tient compte de la taille des pointillés, du nombre de mini-cibles et de la présence éventuelle des zones de format. Si **v** est la **version**, la voici plus lisible (les crochets incomplets en haut signifient qu'on ne conserve que la partie entière du quotient comme dans **version//7** en Python 3) :

$$\left\lfloor \frac{16(v^2+8v+4)-25\left((v\geqslant 2)+\left\lfloor\frac{v}{7}\right\rfloor\right)^2-(v\geqslant 7)\left(36+40\left\lfloor\frac{v}{7}\right\rfloor\right)}{8} \right\rfloor \times 8$$

Comme d'habitude en Python, un test vaut **1** s'il est vrai et **0** s'il est faux. Vous pouvez vous amuser à retrouver la raison de chaque terme dans cette formule, vous avez tout ce qu'il faut ici.

```
09:     while len(self.code)<limite:
10:       if self.gris[i][j]:
11:         self.code.append(self.qr[i][j])
12:       i,j,direction=suivant(i,j,direction,self.dim)
```

On stocke le bit s'il est bien dans le **gris**. La fonction **suivant**, dans **qrcodestandard.py**, donne les coordonnées du bit suivant celui de coordonnées **i,j** selon la **direction** et la **dimension** du code QR et la direction (utile si elle a changé) :

Et voici la partie délicate. On commence par le bit tout en bas à droite puis on remonte de droite à gauche une colonne de deux bits de largeur.

```
def suivant(i,j,direction,dimension):
  j-=1
```

On commence par reculer d'un module vers la gauche.

```
  if j%2!=(dimension%2+(j<6))%2:
    j+=2
    i+=direction
```

Note

Bon, je vais vous aider à démontrer cette formule, appelons **v** la **version**.

On a au total **4v + 17** modules en longueur et en hauteur donc $(4v+17)^2$ modules en tout, ce à quoi il faut soustraire les trois cibles de **8 × 8 = 64** modules, les deux zones de format de **15** modules chacune, le module toujours noir et les pointillés bleus. On a donc, sans compter les mini-cibles et les zones complémentaires de **3 × 6** :

$(4v+17)^2-3\times64-2\times15-1-2\times(4v+17-2\times8)=16v^2+128v+64=16(v^2+8v+4)$.

La seconde expression compte le nombre de modules pris par les mini-cibles qui ne rencontrent pas les pointillés. Elles sont de **5 × 5**, apparaissent à partir de la **version** 2 et on en ajoute à chaque **version** divisant 7.

Enfin, **36** compte les modules pris dans les deux rectangles de **3 × 6**, **40** vient des mini-cibles qui chevauchent les pointillés bleus, donc qui prennent **2 × (25 − 5) = 40** modules supplémentaires.

On compte le nombre d'octets, on divise par **8** (euclidiennement) et j'ai remultiplié par **8** pour obtenir le nombre de bits.

Remarquez que dans notre cas, il reste **7** modules puisque la formule donne bien **560** alors que $29\times29-3\times64-2\times15-1-2\times13-25=567$.

Si on a reculé deux fois, on change de ligne et on avance de deux modules pour rester dans la colonne.

```
  if i==-1:
    j-=2
    i=0
    direction=1
```

Une fois arrivé en haut, on redescend la colonne suivante de deux modules de largeur, toujours de droite à gauche.

```
  if i==dimension:
    j-=2
    i=dimension-1
    direction=-1
```

Et arrivé en bas, on remonte.

```
  if j==6:
    j=5
return i,j,direction
```

Ce document est la propriété exclusive de Johann Locatelli(johann.locatelli@businessdecision.com)

À un détail près : on saute la septième colonne (d'indice 6) qui contient le segment pointillé bleu vertical. La variable **direction** indique si on monte (**-1**) ou si on descend (**+1**). Il suffit de suivre le labyrinthe de la figure 4 en partant du coin inférieur droit.

On peut remarquer que la fonction **suivant** ne se préoccupe pas du **gris** sauf pour la colonne bleue.

On lira donc la suite de bits suivante pour la première colonne, en montant : **01000010010100110001101001011100110110**. La deuxième, descendante, commence par **0101011110**.

Fig. 4 : Comment lire les données brutes.

2.1.2 L'entrelacement des blocs

Pour qu'un petit dessin n'empêche pas la lecture du message d'un code QR, les données sont en fait entrelacées dès qu'un code QR est assez grand selon le tableau suivant :

Niveau de correction	Low	Medium	Quality	High
Entrelacement à partir de la version	**6**	**4**	**3**	**3**

On le voit par exemple plus bas dans le dictionnaire **tableau** de **qrcodestandard.py** où **tableau[5]["L"]** vaut **"(134,108)"** alors que **tableau[6]["L"]** vaut **"2×(86,68)"**.

Les octets des blocs en clair puis correcteurs sont mélangés et donc distribués partout dans l'image. Dans notre cas, on a un seul bloc de 55 octets de données suivis de 15 octets correcteurs. Dans le cas de la **version** 5 et niveau de correction H (**"2×(33,11),2×(34,12)"**), on a deux blocs de données de 11 octets suivis de deux de douze octets suivis de quatre blocs de correction de 22 octets. On écrit les octets des quatre blocs dans un tableau comme ci-dessous.

Le bloc n°**0** contient les 11 premiers octets soit les octets **0** à **10**, une case vide signifie qu'il faut la sauter et passer à la suivante. La partie en clair est construite orthogonalement : on lit le tableau de haut en bas d'abord puis de gauche à droite, il s'agit en quelque sorte du tableau transposé de celui-ci en sautant les cases vides. Ainsi, la partie en clair entrelacée contient les octets **0**, **11**, **22**, **34**, **1**, **12**, **23**, ..., **9**, **20**, **31**, **43**, **10**, **21**, **32**, **44**, **33**, **45**.

On fait la même chose pour les octets des blocs correcteurs qui ont tous la même taille.

```
14:     posclair,lonredondant,lonclair,nbbloc=court2vf(tableau[s
elf.version][self.nivcor])
```

On récupère, dans le **tableau** (qui est un dictionnaire de dictionnaires), la chaîne qui nous intéresse **"(70,55)"** et on la transforme en son équivalent qui ne contient que des coordonnées absolues (plus un zéro initial). Les données en clair et leurs corrections sont regroupés en blocs pas trop grands pour des raisons de temps de calcul. Par exemple, pour le niveau de correction Q de la **version** 3, les données sont en deux blocs de 35 octets dont les 17 premiers contiennent la partie en clair (il reste donc 18 octets pour la correction dans chacun).

Voici le début du contenu de **tableau**, stocké dans **qrcodestandard.py** :

```
tableau={1:{"L":"(26,19)","M":"(26,16)","Q":"(26,13)","H":"(26,9)"},
         2:{"L":"(44,34)","M":"(44,28)","Q":"(44,22)","H":"(44,16)"},
         3:{"L":"(70,55)","M":"(70,44)","Q":"2×(35,17)","H":"2×(35,13)"},
         4:{"L":"(100,80)","M":"2×(50,32)","Q":"2×(50,24)","H":"4×(25,9)"},
         5:{"L":"(134,108)","M":"2×(67,43)","Q":"2×(33,15),2×(34,16)",
"H":"2×(33,11),2×(34,12)"},
         6:{"L":"2×(86,68)","M":"4×(43,27)","Q":"4×(43,19)","H":"4×(43,15)"},
```

Et ainsi de suite jusqu'à 40. J'ai graissé les trois niveaux de correction illustrés dans cette section.

2.1.3 Le code qui détricote

Plutôt que de transformer à la main les informations données dans **[8]**, j'ai préféré créer des fonctions *ad hoc* qui le font à ma place, elles aussi dans **qrcodestandard.py** :

```
def blocs(l):
  return list(eval(l.replace("x","*").replace(")",")+")))
```

Bloc	Octets											
n°0	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	
n°1	**11**	**12**	**13**	**14**	**15**	**16**	**17**	**18**	**19**	**20**	**21**	
n°2	**22**	**23**	**24**	**25**	**26**	**27**	**28**	**29**	**30**	**31**	**32**	**33**
n°3	**34**	**35**	**36**	**37**	**38**	**39**	**40**	**41**	**42**	**43**	**44**	**45**

Ce document est la propriété exclusive de Johann Locatelli(johann.locatelli@businessdecision.com)

LINUX MAGAZINE / FRANCE

DÉCOUVREZ NOS OFFRES D'ABONNEMENTS !

PRO OU PARTICULIER = CONNECTEZ-VOUS SUR :

www.ed-diamond.com

LES COUPLAGES PAR SUPPORT :

VERSION PAPIER

Retrouvez votre magazine favori en papier dans votre boîte à lettres !

VERSION PDF

Envie de lire votre magazine sur votre tablette ou votre ordinateur ?

ACCÈS À LA BASE DOCUMENTAIRE

Effectuez des recherches dans la majorité des articles parus, qui seront disponibles avec un décalage de 6 mois après leur parution en magazine.

SÉLECTIONNEZ VOTRE OFFRE DANS LA GRILLE AU VERSO ET RENVOYEZ CE DOCUMENT COMPLET À L'ADRESSE CI-DESSOUS !

Voici mes coordonnées postales :	
Société :	
Nom :	
Prénom :	
Adresse :	
Code Postal :	
Ville :	
Pays :	
Téléphone :	
E-mail :	

❑ Je souhaite recevoir les offres promotionnelles et newsletters des Éditions Diamond.

❑ Je souhaite recevoir les offres promotionnelles des partenaires des Éditions Diamond.

En envoyant ce bon de commande, je reconnais avoir pris connaissance des conditions générales de vente des Éditions Diamond à l'adresse internet suivante : **boutique.ed-diamond.com/content/3-conditions-generales-de-ventes** et reconnais que ces conditions de vente me sont opposables.

LINUX MAGAZINE / FRANCE

Les Éditions Diamond
Service des Abonnements
10, Place de la Cathédrale
68000 Colmar – France
Tél. : + 33 (0) 3 67 10 00 20
Fax : + 33 (0) 3 67 10 00 21

Vos remarques :

.....

.....

Ce document est la propriété exclusive de Johann Locatelli (johann.locatelli@businessdecision.com)

VOICI TOUTES LES OFFRES COUPLÉES AVEC GNU/LINUX MAGAZINE !

POUR LE PARTICULIER ET LE PROFESSIONNEL ...

Prix TTC en Euros / France Métropolitaine

CHOISISSEZ VOTRE OFFRE !

SUPPORT — Prix en Euros / France Métropolitaine

Offre	ABONNEMENT	PAPIER Réf	PAPIER Tarif TTC	PAPIER + PDF (PDF 1 lecteur) Réf	PAPIER + PDF Tarif TTC	PAPIER + BASE DOCUMENTAIRE (1 connexion BD) Réf	PAPIER + BASE DOCUMENTAIRE Tarif TTC	PAPIER + PDF + BASE DOCUMENTAIRE (PDF 1 lecteur + 1 connexion BD) Réf	PAPIER + PDF + BASE DOCUMENTAIRE Tarif TTC
LM	11 n° GLMF	LM1	65,-	LM12	95,-	LM13	149,-	LM123	174,-
LM+	11 n° GLMF + 6 n° HS	LM+1	118,-	LM+12	177,-	LM+13	197,-	LM+123	256,-
	LES COUPLAGES « LINUX »								
A	11 n° GLMF + 6 n° LP	A1	95,-	A12	140,-	A13	218,-	A123	263,-
A+	11 n° GLMF + 6 n° HS + 6 n° LP + 3 n° HS	A+1	182,-	A+12	263,-	A+13	300,-	A+123	386,-
B	11 n° GLMF + 6 n° MISC	B1	100,-	B12	147,-	B13	233,-	B123	280,-
B+	11 n° GLMF + 6 n° HS + 6 n° MISC + 2 n° HS	B+1	172,-	B+12	248,-	B+13	300,-	B+123	381,-
C	11 n° GLMF + 6 n° LP + 6 n° MISC	C1	135,-	C12	197,-	C13	312,-	C123	374,-
C+	11 n° GLMF + 6 n° HS + 6 n° LP + 3 n° HS + 6 n° MISC + 2 n° HS	C+1	236,-	C+12	339,-	C+13	403,-	C+123	516,-
	LES COUPLAGES « EMBARQUÉ »								
F	11 n° GLMF + 6 n° HK* + 4 n° OS	F1	125,-	F12	188,-	F13	229,-*	F123	292,-*
F+	11 n° GLMF + 6 n° HS + 6 n° HK* + 4 n° OS	F+1	183,-	F+12	275,-	F+13	287,-*	F+123	379,-*
	LES COUPLAGES « GÉNÉRAUX »								
H	11 n° GLMF + 6 n° HK* + 6 n° LP + 6 n° MISC + 4 n° OS	H1	200,-	H12	300,-	H13	402,-*	H123	499,-*
H+	11 n° GLMF + 6 n° HS + 6 n° HK* + 6 n° LP + 3 n° HS + 6 n° MISC + 2 n° HS + 4 n° OS	H+1	301,-	H+12	452,-	H+13	493,-*	H+123	639,-*

Les abréviations des offres sont les suivantes : LM = GNU/Linux Magazine France | HS = Hors-Série | LP = Linux Pratique | OS = Open Silicium | HC = Hackable

* HK : Attention : La base Documentaire de Hackable n'est pas incluse dans l'offre.

N'hésitez pas à consulter les détails des offres ci-dessus sur : www.ed-diamond.com !

Ce document est la propriété exclusive de Johann Locatelli(johann.locatelli@businessdecision.com)

On remplace les **×** par ***** et les séparateurs **),** par **)+** pour que Python transforme tout ça en une seule liste *via* **eval**. Dans notre cas, **blocs** retourne **[70,55]** et pour le niveau de correction Q, elle retourne **[35,17,35,17]**. Oui, la multiplication est toujours prioritaire sur l'addition, même pour les listes.

Comme la manière initiale, relative, n'est pas très utilisable, je l'ai convertie, à l'aide de la fonction **court2vf** en des **True**/**False** dans le tableau transposé :

```
def court2vf(l):
  l1=blocs(l)
  clair=l1[1::2]
  M=max(clair)
  m=min(clair)
  s1=M*len(l1)//2
  s2=sum(clair)
  l2=[True]*m*len(clair)
  l2+=[False]*(M-m)*l1.count(m)
  l2+=[True]*(M-m)*l1.count(M)
  l3=sum(l1[2*i]-l1[2*i+1] for i in range(len(clair)))
  return l2,l3,s1,len(l1)//2
```

Cette fonction retourne, dans le niveau de correction L de la **version** 3, **[True]*55,15,55,1** et pour le niveau de correction Q, **[True]*34,36,34,2**. Dans le cas où les blocs en clair n'ont pas tous la même longueur comme dans la **version** 5 et niveau de correction H (**"2×(33,11),2×(34,12)"**), elle retourne **[True]*11*4+[False]*2+[True]*2,46,88,4**.

```
15:     self.clair=[[] for _ in range(nbbloc)]
16:     self.redondant=[[] for _ in self.clair]
```

Pour le moment, **clair** et **redondant** contiennent autant de listes vides que de blocs de données dans le code QR.

```
17:     j=0
18:     for i in range(lonclair):
19:       if posclair[i]:
20:         self.clair[i%nbbloc]+=self.code[j*8:j*8+8]
21:         j+=1
22:     for i in range(lonredondant):
23:       self.redondant[i%nbbloc]+=self.code[(i+j)*8:(i+j+1)*8]
```

posclair[i] vaut **True** s'il faut stocker le bit. **i** est le numéro du bit dans le tableau transposé remis à plat, sans tenir compte des longueurs éventuellement différentes des blocs. **j** est le numéro du bit, mais en en tenant compte, il sert aussi à déterminer le début des blocs correcteurs.

Dans notre cas (**version** 3 et niveau de correction L), les données ne sont pas coupées et on a au maximum 70 octets de données dont 55 de message.

Voici le contenu hexadécimal de la partie en clair **4254c6973657a20474e552f4c696e7578206d6167617a696e65204672616e63652e20f09f98830ec11ec11ec11ec11ec11ec11ec11ec11** et de la correction **0b1622eb6e2c152a08d34a2dd8c788**. Observez le bourrage à la fin, on le voit sur la figure 5 juste après la pièce de Tetris blanche au milieu de la zone haute ; et le peu d'erreurs que l'on pourra corriger. *Akala jingle bells* toutes ces couleurs.

Fig. 5 : Le découpage du message.

Les barres noires délimitent les zones importantes : l'entête, les données (avec bourrage), la correction et le supplément inutilisé. Les octets ou quadruplets sont délimités par les barres magenta.

2.2 Vérification de la correction du message

On va se contenter de dire que tout va bien, le développement de la correction d'erreurs se fera plus loin après des préliminaires de rigueur.

```
01:   def messagecorr(self):
02:     if None in [self.clair,self.redondant]:
03:       self.messagebrut()
04:     compte=0
05:     if self.corrige:
06:       for i in range(len(self.clair)):
07:         self.clair[i],self.redondant[i],c=corrige(self.
clair[i],self.redondant[i])
08:         compte+=c
09:     clair=[]
10:     for l in self.clair:
11:       clair+=l
12:     self.clair=clair
13:     redondant=[]
14:     for l in self.redondant:
15:       redondant+=l
16:     self.redondant=redondant
17:     if compte:
18:       self.messageok=compte
19:     else:
20:       self.messakeok=True
```

Ligne 04, la variable **compte** décompte le nombre total d'erreurs corrigées, qu'on corrige si le drapeau des options de ligne de commandes **corrige** est égal à **1**. Ensuite, on regroupe les blocs de la partie en clair puis ceux de la partie redondante.

2.3 Décodage final

Enfin ! On va pouvoir savoir ce que contient notre exemple de code QR.

Ce document est la propriété exclusive de Johann Locatelli(johann.locatelli@businessdecision.com)

```
01:   def decodeqr(self):
02:     if self.messageok is None:
03:       self.messagecorr()

05:     self.mode=3-self.code[:4].index(1)
06:     self.longueur=longbin(self.version,self.mode)
```

La sélection du **mode** se fait selon la position du premier **1** dans les quatre premiers bits (qui sont **0100**, observer le carré en bas à droite de la figure 5). Ici, on utilise le mode Byte, le **1** est en effet en deuxième position (donc d'indice 1) et **mode** vaut **2**.

Voici la fonction **longbin**, présente dans **qrcodestandard.py** :

```
def longbin(version,mode):
  if version<=9:
    return {0:10,1:9,2:8,3:8}[mode]
  elif version<=26:
    return {0:12,1:11,2:16,3:10}[mode]
  return {0:14,1:13,2:16,3:12}[mode]
```

Selon le **mode** et la **version**, varie le nombre de bits du champ qui donne le nombre exact de caractères du message final et non le nombre d'octets, la nuance est importante pour les formats numérique, alphanumérique et kanji. Celui-ci est écrit en base 2, le champ de **longueur** contient 8 bits dans notre cas, c'est le rectangle de **2 × 4** sous le trait noir.

La fonction suivante, **bin2dec** stockée dans **qrcodeoutils.py**, calcule la valeur décimale d'une suite de **0** et de **1** stockée dans une liste.

```
def bin2dec(n):
  s=0
  for b in n:
    s*=2
    s+=b
  return s
```

Cette fonction utilise l'algorithme de Horner (William George, pas Yvette), rapide et simple à mettre en œuvre.

En effet, si $n=\overline{b_7b_6b_5b_4b_3b_2b_1b_0}_2=\sum_{i=0}^{7} b_i 2^i$ (c'est l'écriture de **n** en base 2, chaque b_i vaut **0** ou **1**), on peut écrire que $n=b_7 2^7+b_6 2^6+b_5 2^5+b_4 2^4+b_3 2^3+b_2 2^2+b_1 2^1+b_0 2^0=(((((((0\times2+b_7)\times2+b_6)\times2+b_5)\times2+b_4)\times2+b_3)\times2+b_2)\times2+b_1)\times2+b_0$.

Pourquoi diable ? Pour économiser le nombre de multiplications effectuées. On a 29 multiplications par le calcul naïf en comptant le calcul des puissances (en $O(n^2)$, en gros proportionnel au carré du nombre de bits) et 8 par la méthode de Horner (en O(n), proportionnel au nombre de bits). La technique fonctionne aussi si on veut évaluer un polynôme (on remplace 2 par **x**), on la réutilisera plus loin.

```
20:     self.longclair=bin2dec(self.clair[4:4+self.longueur])
```

On calcule donc la longueur exacte du message qui est, dans notre cas, de 37 caractères puisque les huit bits du champ sont **00100101** soit **32 + 4 + 1**.

Suivent les fonctions des quatre modes. Le message est lu immédiatement après les bits de **longueur**, donc au treizième bit dans notre cas.

```
22:     def numeric():
23:       n=""
24:       fin=self.longclair%3
25:       self.longclair-=fin
26:       i=4+self.longueur
27:       while len(n)<self.longclair:
28:         n=n+str(bin2dec(self.clair[i:i+10]))
29:         i+=10
30:       if fin==1:
31:         n=n+str(bin2dec(self.clair[i:i+4]))
32:       elif fin==2:
33:         n=n+str(bin2dec(self.clair[i:i+7]))
34:       return int(n)
```

Le format numérique ne permet de stocker que des nombres entiers positifs, par exemple de très grands nombres premiers. Comme **$999<2^{10}=1024$**, il suffit de dix bits pour stocker trois chiffres en base 10, ce qui est bien mieux que 24 bits en ASCII. Que se passe-t-il si le nombre de chiffres n'est pas un multiple de 3 ? Si le nombre contient **3n + 1** chiffres, le dernier est codé sur 4 bits ligne 31 (car **$2^3<9<2^4$**) et s'il en contient **3n + 2**, les deux derniers sont codés sur 7 bits ligne 33 (car **$2^6<99<2^7$**).

```
36:     def alphanumeric():
44:       ch=""
45:       fin=self.longclair%2
46:       self.longclair-=fin
47:       i=4+self.longueur
48:       while len(ch)<self.longclair:
49:         n=bin2dec(self.clair[i:i+11])
50:         q,r=divmod(n,45)
51:         ch=ch+alphanum[q]+alphanum[r]
52:         i+=11
53:       if fin==1:
54:         ch=ch+alphanum[bin2dec(self.clair[i:i+6])]
55:       return ch
```

Le format alphanumérique permet d'écrire des URL ou du texte simple. Il contient 45 symboles : de **0** à **9**, les chiffres, de **10** à **35**, les lettres et enfin neuf symboles supplémentaires de **36** à **44** : **alphanum** (stocké dans **qrcodestandard.py**) est la chaîne de caractères **"0123456789ABCDEFGHIJKLMNOPQRSTUVWXYZ $%*+-./:"** avec un espace après le **Z** et pas de lettre minuscule.

En fait, deux symboles sont codés sur 11 bits : le quotient et le reste de la division par **45** puisque **$2^{10}<45^2=2025<2^{11}$**.

Ce document est la propriété exclusive de Johann Locatelli(johann.locatelli@businessdecision.com)

Si le nombre de symboles est impair, le dernier est codé sur 6 bits ligne 54 (car $2^5<45<2^6$).

```
57:     def byte():
58:       return bytes([bin2dec(self.clair[i:i+8]) for i in
range(4+self.longueur,4+self.longueur+self.longclair*8,8)]).
decode("utf-8")
```

Là, c'est simple : chaque octet est lu tel quel, mais nous supposons ici que le message est encodé en utf-8, ce qui n'est pas le cas en général, les codes QR acceptent d'autres encodages. Notre message commence par **0x4c 0x69 0x73 0x65** ce qui donne **Lise**. J'ai utilisé le type **bytes**, qui est la manière élégante en Python 3 d'effectuer des manipulations sur un flux d'octets. Il s'agit en gros d'une chaîne de caractères (d'octets en fait, immuable) sans encodage. Ainsi, afficher un byte affiche la liste des valeurs décimales de ses éléments, c'est le format idéal pour la manipulation de flux de caractères. La méthode **decode** permet de le traduire en une chaîne de caractères encodée, ici en utf-8. Mon précédent code essayait de sortir le message sans se soucier de son encodage.

```
63:     def kanji():
64:       raise NotImplementedError
```

Je ne connais rien aux kanjis donc j'ai préféré m'abstenir d'écrire un code faux même si j'ai déjà joué avec une Casio FX-850P.

```
66:     self.message={0:numeric,1:alphanumeric,2:byt,3:kanji}[self.mode]()
```

Python permet ce genre de syntaxe concise et efficace. Le numéro du **mode** est remplacé par la fonction idoine, qui n'a pas d'argument.

2.4 Affichage

Testons notre travail :

```
if __name__=="__main__":
  essai=qrdecode()
  essai.decodeqr()
  print(essai)
```

On initialise **essai**, on applique la méthode qui décode qui, par poupées russes, fait tout le travail et voici la sortie de la méthode **__str__** au grand complet :

```
> ./qrdecode.py glmf3.png -a 1
Fichier : glmf3.png
Niveau de correction : Low
Masque :
```

```
Dimensions : 29×29
Version : 3
Mode : Byte
Longueur du message : 37
Message :
Lisez GNU/Linux magazine France. ☺
```

Et voilà, avec le beau *sourilaid* bien interprété, certains lecteurs transforment les *sourilaids* en une image.

Conclusion

Nous pouvons maintenant lire n'importe quel code QR bien carré, mais pas ceux munis d'un logo. Pour cela, il faut un détour par les mathématiques des corps finis et ça, ce sera pour un prochain article. ■

Références

[1] PATROIS N., « *Déchiffrer un code QR* », *GNU/Linux Magazine n°193*, p. 28 à 31.

[2] Le code et les exemples sont présents ici : https://github.com/GLMF/GLMF194

[3] « *Reed–Solomon codes for coders* » https://en.wikiversity.org/wiki/Reed%E2%80%93Solomon_codes_for_coders et le complément plus précis https://en.wikiversity.org/wiki/Reed%E2%80%93Solomon_codes_for_coders/Additional_information. Le code QR utilisé dans cet article a été créé sur le site https://www.the-qrcode-generator.com/, car le code de création n'existait pas à ce moment.

[4] http://www-igm.univ-mlv.fr/~dr/XPOSE2011/QRCode/workflows.html.

[5] PATROIS N., « *Expliquer un code QR* », *GNU/Linux Magazine n°193*, p. 22 à 26.

[6] https://en.wikiversity.org/wiki/Reed%E2%80%93Solomon_codes_for_coders/Additional_information#Alignment_pattern.

[7] https://en.wikiversity.org/wiki/Reed%E2%80%93Solomon_codes_for_coders/Additional_information#Symbol_size.

[8] https://en.wikiversity.org/wiki/Reed%E2%80%93Solomon_codes_for_coders/Additional_information#RS_block_size.

Ce document est la propriété exclusive de Johann Locatelli(johann.locatelli@businessdecision.com)

CHIFFRER UNE PARTITION AVEC LVM ET LUKS

Tristan COLOMBO

Un nouvel ordinateur portable ? Il est essentiel de chiffrer votre disque ! Pour cela, nous allons utiliser un conteneur LVM (Logical Volume Manager) pour les données et les chiffrer avec un conteneur LUKS (Linux Unified Key Setup).

Chiffrement LVM Debian Partition LUKS

L'OBJECTIF

Lors d'une nouvelle installation, chiffrer la partition `/home` et la partition d'échanges swap.

LES OUTILS

- une Debian 8.3

PHASE 1

Début de l'installation

Je ne vous ferai pas l'offense de décrire l'installation précédant le partitionnement... Je commencerai donc mes explications à partir du partitionnement manuel.

PHASE 2

Partitionnement manuel : la partition /

Pour `/`, nous allons créer une partition primaire et la formater en `ext4`. Suivant vos habitudes, dimensionnez (voire surdimensionnez votre partition). Pour ma part, pour avoir longtemps voulu jouer à dimensionner au plus juste mes partitions racines et avoir dû modifier leur taille de nombreuses fois (en craignant de perdre toutes les données), j'en suis venu à systématiquement surdimensionner ces partitions. Vu la taille des disques de nos jours, on peut se le permettre et cela évite pas mal de tracas...

PHASE 3

Partitionnement manuel : la partition de swap

Suivant la règle admise, la partition d'échange (swap) doit avoir une taille qui soit le double de la RAM. Nous créons cette partition en tant que partition logique et sélectionnons « swap » dans le menu de paramétrage du système de fichiers.

PHASE 4

Partitionnement manuel : la partition de /home

Avec l'espace restant, nous allons créer une partition logique et sélectionner dans le menu « système de fichiers » : ***Volume physique pour le chiffrement*** (*physical volume for encryption*).

Ce document est la propriété exclusive de Johann Locatelli(johann.locatelli@businessdecision.com)

PHASE 5

Configurer les partitions chiffrées

Il faut maintenant configurer les partitions chiffrées... grâce à l'entrée du menu portant le même nom : ***Configurer les volumes chiffrés***. Vous pourrez voir un message vous avertissant que le schéma de partitionnement que vous avez défini doit être appliqué avant de pouvoir poursuivre. Si des données étaient présentes, elles vont être perdues... Vous devez donc valider les modifications en sélectionnant le choix « Oui ».

Sélectionnez ensuite ***Créer des volumes chiffrés*** (*Create encrypted volume*) et indiquez que vous souhaitez chiffrer la partition définie précédemment (elle possède le drapeau « chiffré » ou « crypto ») ainsi que la swap. Notez qu'après ce choix vous retournerez à l'écran permettant la création des volumes chiffrés : vos choix ont quand même été pris en compte. Sélectionnez simplement ***Terminer*** (« *Finish* ») puis effacez (ou non) les données présentes sur le disque. Pour une installation propre, j'ai choisi un effacement.

Pour finir, vous devez donner une phrase secrète pour le chiffrement des partitions (je vous recommande la même phrase pour **/home** et la swap de manière à minimiser les risques d'erreur...).

PHASE 6

Créer la partition LVM

Sélectionnez ***Configurer le gestionnaire de volumes logiques (LVM)*** (*Configure the Logical Volume Manager*) et confirmez l'écriture des changements opérés précédemment sur les partitions.

Vous devez ensuite créer un groupe de volumes en sélectionnant l'entrée appropriée du menu. Il faut donner un nom à ce groupe et sélectionner les partitions qui feront partie du groupe (les partitions notées **sd..._crypt**).

PHASE 7

Créer les volumes logiques home et swap

Pour créer le volume logique **home**, sélectionnez ***Créer un volume logique*** (*Create logical volume*) puis le groupe de volumes créé précédemment, appelez votre volume **home** (sans le slash) et indiquez sa taille (ici ce sera toute la taille disponible moins la taille de la swap).

Renouvelez l'opération pour créer un volume **swap** de la taille restante et sélectionnez ensuite ***Terminer*** (*Finish*) pour valider vos choix.

PHASE 8

Monter le volume home sur /home et la swap

Sélectionnez le volume logique **home** et choisissez **ext4** comme système de fichiers et **/home** comme point de montage. La ligne indiquant la partition que vous recherchez doit ressembler à :

```
Groupe de volumes LVM Nom_Groupe, volume logique home - XXX GB
Linux device-mapper (linear)
```

Effectuez la même opération pour le volume **swap** en sélectionnant cette fois-ci « swap » dans le menu « système de fichiers ».

Achevez les opérations en sélectionnant ***Terminer le partitionnement et appliquer les changements*** (*Finish partitioning and write changes to disk*).

PHASE 9

Poursuite de l'installation

L'installation se déroule ensuite de manière classique, nous ne nous y attarderons donc pas.

PHASE 10

Premier lancement

Au démarrage de votre système, vous allez avoir une bonne surprise :

```
Loading, please wait...
  Volume group "Nom_Groupe" not found
  Skipping volume group Nom_Groupe
Unable to find LVM volume Nom_Groupe/swap
Please unlock disk sd..._crypt :
```

Ce document est la propriété exclusive de Johann Locatelli(johann.locatelli@businessdecision.com)

Et la même chose pour la partition **home**... vous devrez donc taper deux fois votre *passphrase* et ensuite vous connecter avec votre identifiant et votre mot de passe. C'est un peu lourd, non ?

PHASE 11

Montage et déchiffrement automatiques

En tant que root, nous allons créer le fichier **/root/cryptkey** :

```
$ su
# cd
# dd if=/dev/urandom of=cryptkey bs=512 count=1
1+0 enregistrements lus
1+0 enregistrements écrits
512 octets (512 B) copiés, 0,000664417 s, 771 kB/s
# chmod 700 cryptkey
```

Éditez ensuite le fichier **/etc/crypttab** avec **vi** par exemple et remplacez les occurrences de **none** par **/root/cryptkey** (la commande magique est **:%s/none/\/root\/cryptkey/**) :

```
sd..._crypt UUID=... /root/cryptkey luks
sd..._crypt UUID=... /root/cryptkey luks
```

Il va falloir maintenant enregistrer le mot de passe pour chacune des partitions. Les commandes suivantes seront donc à renouveler deux fois (par exemple, chez moi il a fallu les exécuter sur **/dev/sda5** et **/dev/sda6**) :

```
# cryptsetup luksAddKey /dev/sd... /root/cryptkey
Entrez une phrase de passe :
```

Remplacez bien sûr **/dev/sd...** par votre partition.

Vous pouvez vérifier que tout s'est correctement déroulé par :

```
# cryptsetup luksDump /dev/sd...
...
Key Slot 1: ENABLED
  Iterations:  285711
  Salt:  5c 22 24 31 ...
       ...
  Key material offset:512
  AF stripes:  4000
Key Slot 2: DISABLED
...
```

LE RÉSULTAT

Lancez la commande suivante avant de rebooter :

```
# update-initramfs -u -k all
```

Votre partition **/home** est chiffrée et vous n'avez qu'à taper votre identifiant et votre mot de passe pour vous connecter. ■

Pour en savoir plus...

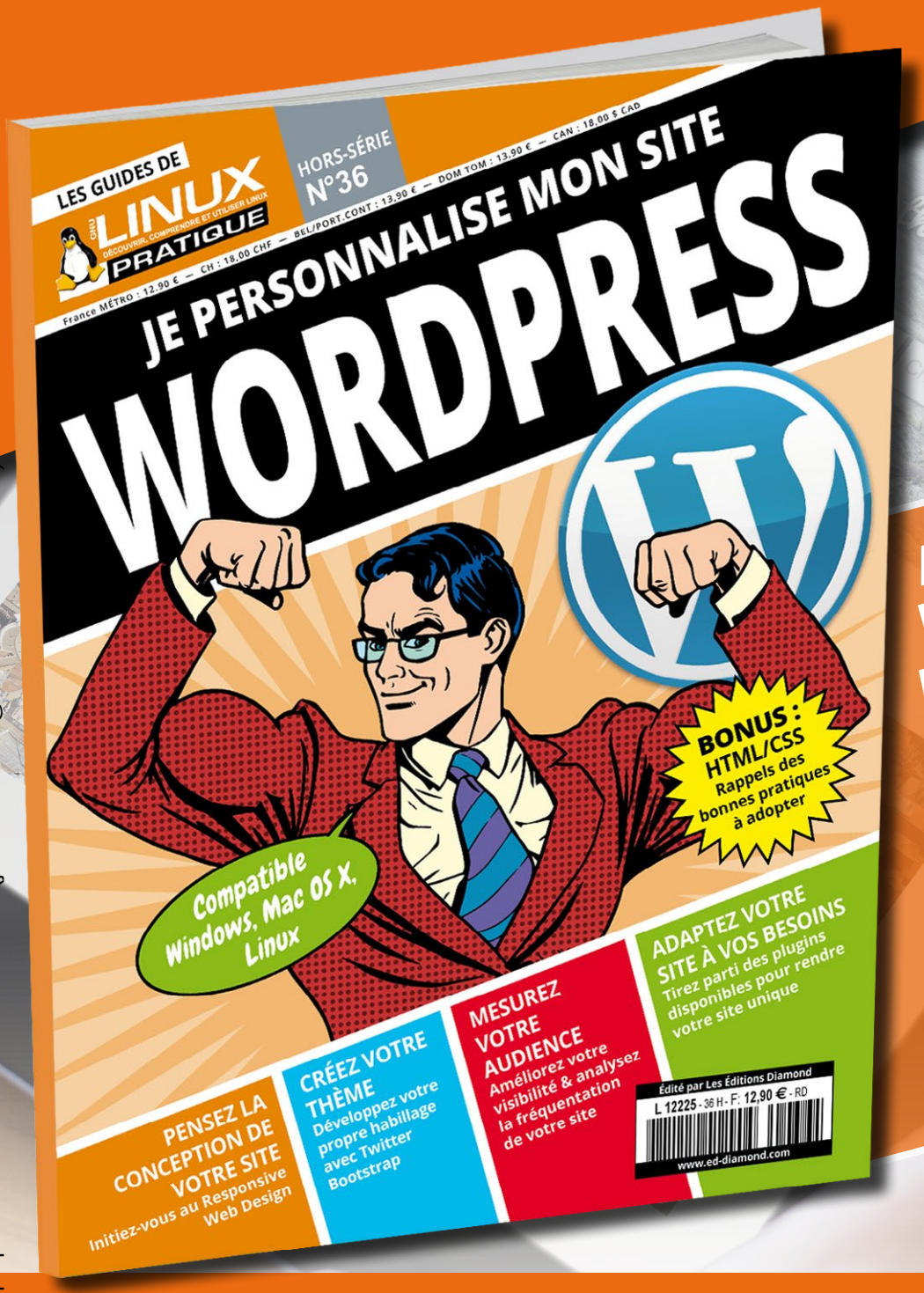
Vous voulez en savoir un peu plus sur tout ce qui touche à la confidentialité des données ? *GNU/Linux Magazine Hors-série n°84* fournit un guide complet sur la surveillance réseau.

On y traite des différents outils et techniques à connaître et des bonnes pratiques à respecter pour se protéger efficacement.. Un numéro à découvrir en kiosque et sur www.ed-diamond.com/.

Ce document est la propriété exclusive de Johann Locatelli(johann.locatelli@businessdecision.com)

DISPONIBLE EN JUIN !

LINUX PRATIQUE HORS-SÉRIE N°36 !



PERSONNALISEZ VOTRE SITE WORDPRESS

DISPONIBLE EN JUIN !

CHEZ VOTRE MARCHAND DE JOURNAUX ET SUR :

www.ed-diamond.com

Ce document est la propriété exclusive de Johann Locatelli(johann.locatelli@businessdecision.com)

GESTION DE PAQUETS SÛRE ET FLEXIBLE AVEC GNU GUIX

Ludovic COURTÈS [Développeur GNU ; ingénieur de recherche Inria]

Les distributions GNU/Linux classiques sont mal aimées. Cet article revient sur leurs limites et donne un aperçu de la solution que propose le projet GNU avec Guix, un gestionnaire de paquets transactionnel, flexible et personnalisable.

Mots-clés : Gestion de paquets, Distribution, Programmation fonctionnelle, GNU

Résumé

***Linux Weekly News* dénombre plus de 500 distributions GNU/Linux [1] et y voit une « célébration de la diversité ». Avec ça, le problème de la distribution de logiciels a été largement exploré, et probablement largement résolu, se dit-on. Et pourtant ! Cet article traite d'un nouveau gestionnaire de paquets, GNU Guix [2], qui fournit mises à jour transactionnelles, retours en arrière, et est extensible et personnalisable à souhait.**

Les distributions « classiques » ont certaines limitations, comme l'impossibilité de revenir en arrière après une mise à jour ou la difficulté de reproduire un environnement logiciel exact. La profusion de gestionnaires de paquets annexes ajoute à la confusion, et l'utilisation de **Docker** contourne ces difficultés sans les corriger. GNU Guix met en œuvre une gestion des paquets *fonctionnelle* qui entend résoudre certains de ces problèmes.

1 Le problème

Avec une distribution GNU/Linux « classique », une mise à jour est toujours quitte ou double : il ne vaut mieux pas qu'une coupure électrique intervienne pendant la mise à jour (on risque de se retrouver avec un système inutilisable), ou qu'un des nouveaux paquets ne fonctionne pas (difficile de revenir en arrière).

Une fois que l'on a une machine avec une distribution classique qui fonctionne bien, on aimerait pouvoir reproduire son état, c'est-à-dire l'ensemble des paquets installés et la configuration associée, sur une autre machine par exemple. C'est chose difficile, car on n'est jamais sûr de pouvoir réinstaller les mêmes paquets aux mêmes versions, et parce qu'une partie de la configuration du système échappe complètement au gestionnaire de paquets. Le fait qu'il soit devenu courant de combiner beaucoup de gestionnaires de paquets (**Bower**, **Cabal**, **CPAN**, **npm**, **pip**, etc.) rend le contrôle de l'environnement logiciel plus difficile encore.

Docker permet de contourner la difficulté en permettant de « figer » une image de l'état du système. Pour y parvenir, on va stocker dans un `Dockerfile` la séquence de commandes permettant *a priori* d'atteindre l'état souhaité.

Cette approche risque de ne pas être reproductible, puisque l'effet des commandes dépend de l'état des dépôts de code source des gestionnaires de paquets utilisés dans l'image Docker. Elle favorise un empilement de couches sans donner une vision globale dans la composition des paquets. Enfin, elle a d'autres inconvénients comme l'utilisation inefficace du stockage et la difficulté de s'assurer que chaque image contient les mises à jour de sécurité critiques.

Ce document est la propriété exclusive de Johann Locatelli(johann.locatelli@businessdecision.com)

2 La gestion de paquets « fonctionnelle »

Aux alentours de 2004, Eelco Dolstra a commencé son travail de thèse sur **Nix [3]**, un gestionnaire de paquets *fonctionnel*. Ici « fonctionnel » fait référence non pas au fait qu'il fonctionne (bien qu'il fonctionne), mais au paradigme de gestion de paquets, qui s'inspire de la programmation fonctionnelle telle que mise en œuvre par des langages comme **OCaml**, **Haskell** ou **Scheme**.

L'idée est de voir chaque paquet comme une valeur immuable, résultat de l'application d'une fonction de compilation à un ensemble d'arguments. Par exemple, le binaire du logiciel **GIMP** est vu comme le résultat d'appliquer une fonction qui lance `./configure && make && make install` à un ensemble d'entrées : le source de GIMP, **GCC**, la bibliothèque standard du C (libc), la bibliothèque **GTK+**, etc. À son tour, GTK+ est le résultat de cette fonction appliquée à d'autres arguments. De cette façon, on exprime un graphe de dépendances tel que celui de la figure 1.

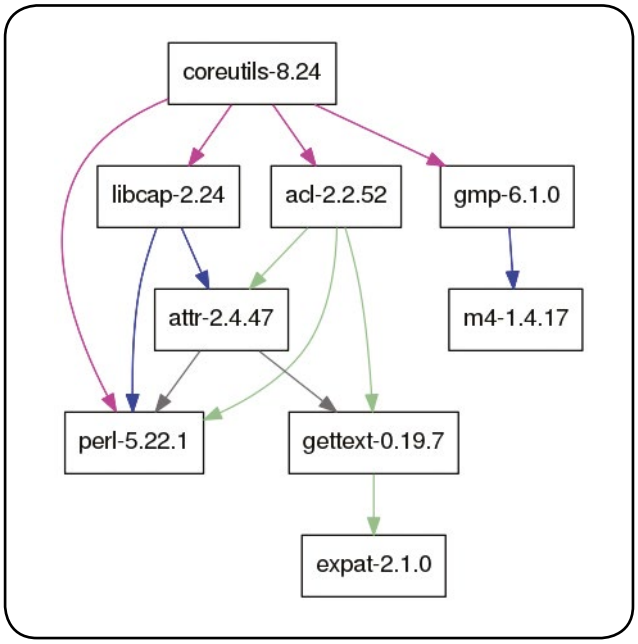


Fig. 1 : Graphe de dépendances à la compilation des outils de base GNU (Coreutils) produit par la commande guix graph coreutils.

GNU Guix a été créé en 2013 pour reprendre les fondements de Nix, mais en fournissant une interface de programmation unifiée et embarquée dans le langage Scheme, un langage de programmation fonctionnelle générique, mis en œuvre par **GNU Guile [3]**. De cette façon, Guix a accès à tout Guile - compilateur, déverminer, bibliothèques, environnement de développement, etc. - et tout Guix est accessible en Scheme.

Autrement dit, Guix est une bibliothèque Scheme comme une autre, et la distribution toute entière est une bibliothèque où chaque paquet est un objet Scheme. Le but recherché, et atteint, est que l'on puisse simplement écrire des fonctions qui manipulent des paquets, écrire des interfaces utilisateurs et autres applications qui se servent de Guix. Guix fournit ainsi un langage d'empaquetage universel (s'applique à des paquets C, mais aussi **PyPI**, **RubyGems**, **ELPA**, etc.), plusieurs interfaces utilisateurs (ligne de commandes, **Emacs**, et Web), un outil de gestion d'environnements de développement (`guix environment`, comme **Virtualenv**, **rvm**, etc., mais pour tout type de paquet), un outil de vérification des paquets (`guix lint`), un outil de mise à jour des recettes de paquets (`guix refresh`), et d'autres.

Mais l'enjeu de cette approche est, plus généralement, de fournir un système transparent et bidouillable que les usagers puissent s'approprier plus facilement, pour exercer la liberté n°1 que le logiciel libre leur donne **[4]**.

3 Installation

Il est possible d'installer Guix comme un gestionnaire de paquets supplémentaire (un de plus !) sur votre distribution GNU/Linux. Il cohabitera pacifiquement avec la distribution et sans interférence : Guix installe ses affaires dans `/gnu/store` comme nous le verrons plus bas, et vous pouvez à tout moment ajouter ou enlever les paquets installés avec Guix de `PATH` et autres variables.

L'installation peut se faire de plusieurs façons comme détaillé dans le manuel de Guix **[4]**, le plus simple étant d'utiliser les binaires précompilés fournis par le projet.

Le démon `guix-daemon` prend en charge la compilation des paquets et/ou le téléchargement de binaires précompilés provenant de sources autorisées. Pour que la compilation de paquets puisse être vue comme une fonction « pure » et soit reproductible, ce démon s'assure que la compilation est effectuée dans un environnement isolé, un *conteneur*.

4 En avant !

Une fois `guix-daemon` démarré, on peut déjà lancer une construction de paquets :

Ce document est la propriété exclusive de Johann Locatelli(johann.locatelli@businessdecision.com)

```
$ guix build hello
Le fichier suivant sera téléchargé:
   /gnu/store/zby49aqfbd9w9br4l52mvb3y6f9vfv22-hello-2.10

Found valid signature for /gnu/store/
zby49aqfbd9w9br4l52mvb3y6f9vfv22-hello-2.10
From https://mirror.hydra.gnu.org/nar/...-hello-2.10
Downloading zby49a...-hello-2.10 (170KiB installed)...
 https://mirror.hydra.gnu.org/nar/...-hello-2.10 737KiB/s 00:00 |
49KiB transferred
/gnu/store/zby49aqfbd9w9br4l52mvb3y6f9vfv22-hello-2.10
$ /gnu/store/zby49aqfbd9w9br4l52mvb3y6f9vfv22-hello-2.10/bin/hello
Bonjour, le monde !
```

Ce qu'on voit ici, c'est qu'à une compilation du paquet GNU Hello, **guix build** a *substitué* un binaire pré-compilé téléchargé directement depuis **mirror.hydra.gnu.org**. Le résultat est ce long nom de répertoire en **/gnu/store** qui contient effectivement la commande **hello**.

Tout ce que produit Guix arrive dans le répertoire **/gnu/store**, qu'on appelle *l'entrepôt* (le *store* en anglais). Par exemple, **hello** a notamment une dépendance à l'exécution sur la bibliothèque standard du langage C (**libc**), qui elle est aussi dans l'entrepôt, comme le montre cette commande qui liste les dépendances à l'exécution :

```
$ guix gc --references /gnu/store/
zby49aqfbd9w9br4l52mvb3y6f9vfv22-hello-2.10
/gnu/store/8m00x5x8ykmar27s9248cmhnkdb2n54a-glibc-2.22
/gnu/store/v39bh3ln3ncnzhyw0kd12d46kww9747v-gcc-4.9.3-lib
/gnu/store/zby49aqfbd9w9br4l52mvb3y6f9vfv22-hello-2.10
```

Cette longue chaîne en base32 dans les chemins ci-dessus est en fait le condensé SHA256 de *toutes* les dépendances utilisées à la compilation pour produire ce résultat. Dans le cas de Hello, les dépendances à la compilation sont : le source de Hello, le script de compilation, mais aussi la libc, le compilateur, **Bash**, **coreutils**, **sed**, **grep**, **awk**, etc. Vraiment toutes les dépendances ! De cette manière, on a vraiment une correspondance directe entre le source, y compris les outils de compilation, et le binaire produit — c'est une formalisation de la notion de *Corresponding Source* telle que décrite dans la GNU GPL.

Cette correspondance source/binaire est cruciale. Elle signifie que les usagers n'ont pas à faire confiance aveuglément à un fournisseur de binaire : les usagers peuvent à tout moment compiler localement et vérifier qu'ils obtiennent le même résultat, à l'octet près, que le fournisseur de binaires. C'est exactement ce que vérifie la commande **guix challenge**.

5 Les profils

Évidemment, on n'a pas vraiment envie de taper ces chemins à la main. La commande **guix package** permet à chaque usager (pas besoin d'être **root**) de maintenir des *profils* où sont installés des paquets. Il suffit de rajouter un profil dans **PATH** et ses paquets deviennent disponibles. Par exemple, pour installer Emacs et **Vim** (on ne sait jamais) dans son profil par défaut, **~/.guix-profile**, on fait simplement :

```
$ guix package --install emacs vim
Les paquets suivants seront installés:
   vim  7.4     /gnu/store/...-vim-7.4
   emacs        24.5    /gnu/store/...-emacs-24.5

Les dérivations suivantes seront compilées:
   /gnu/store/...-profile.drv
   /gnu/store/...-gtk-icon-themes.drv
   /gnu/store/...-ca-certificate-bundle.drv
   /gnu/store/...-info-dir.drv
Le fichier suivant sera téléchargé:
   /gnu/store/...-vim-7.4

[...]

2 paquets dans le profil
Il pourrait être nécessaire de définir les variables
d'environnement suivantes:
   export PATH="/home/alice/.guix-profile/bin"
   export INFOPATH="/home/alice/.guix-profile/share/info"
$ guix package --list-installed
emacs   24.5    out     /gnu/store/...-emacs-24.5
vim     7.4     out     /gnu/store/...-vim-7.4
```

Guix est attentionné et nous indique même les variables d'environnement à définir pour pouvoir utiliser les paquets installés. Pour que ces variables soient automatiquement définies, on peut rajouter cette ligne dans **~/.bash_profile** :

```
GUIX_PROFILE="$HOME/.guix-profile" . "$GUIX_PROFILE/etc/profile"
```

Cette opération est *transactionnelle* : on peut taper **<Ctrl>** + **<C>** à tout moment, et soit à la fois Emacs et Vim seront installés, soit aucun ne le sera. C'est aussi le cas pour des transactions plus complexes :

```
$ guix package -r vim -i nano
Le paquet suivant sera supprimé:
   vim  7.4     /gnu/store/...-vim-7.4

Le paquet suivant sera installé:
   nano 2.5.3   /gnu/store/...-nano-2.5.3

[...]

2 paquets dans le profil
```

Ce document est la propriété exclusive de Johann Locatelli(johann.locatelli@businessdecision.com)

Chaque transaction donne lieu à une nouvelle *génération* du profil :

```
$ guix package --list-generations
Génération 1    Mar 30 2016 14:29:05
  emacs 24.5    out     /gnu/store/...-emacs-24.5
  vim   7.4     out     /gnu/store/...-vim-7.4

Génération 2    Mar 30 2016 14:39:02    (actuel)
  emacs 24.5    out     /gnu/store/...-emacs-24.5
  nano  2.5.3   out     /gnu/store/...-nano-2.5.3
```

On peut à tout moment basculer vers une autre génération, la précédente par exemple :

```
$ guix package --roll-back
switched from generation 2 to 1
```

Ce mécanisme vaut aussi pour les mises à jour (avec **`--upgrade`**)... et c'est très rassurant !

6 Maîtriser ses environnements logiciels

Un profil n'est rien d'autre qu'une forêt de liens symboliques :

```
$ readlink -f ~/.guix-profile
/gnu/store/...-profile
$ readlink ~/.guix-profile/bin/emacs
/gnu/store/...-emacs-24.5/bin/emacs
```

On peut donc en créer autant qu'on veut, et Guix saura nous dire quelles sont les variables d'environnement qui vont bien :

```
$ guix package -p ~/dev-python -i python@2.7 python2-numpy
Les paquets suivants seront installés:
   python2-numpy       1.10.4  /gnu/store/...-python2-numpy-1.10.4
   python      2.7.10  /gnu/store/...-python-2.7.10

2 paquets dans le profil
Il pourrait être nécessaire de définir les variables d'environnement
suivantes:
   export PATH="/home/ludo/dev-python/bin"
   export PYTHONPATH="/home/ludo/dev-python/lib/python2.7/site-packages"
$ guix package -p ~/dev-python --search-paths
export PATH="/home/ludo/dev-python/bin"
export PYTHONPATH="/home/ludo/dev-python/lib/python2.7/site-packages"
$ eval `guix package -p ~/dev-python --search-paths`
$ python -c "import numpy; print(numpy.version.version)"
1.10.4
```

L'outil **`guix environment`** permet de créer des environnements de développement temporaires, à la volée. Par exemple, pour un environnement Python 2.x avec Numpy comme ci-dessus, on pourrait simplement faire :

```
$ guix environment --ad-hoc python@2 python2-numpy -- \
    python -c "import numpy; print(numpy.version.version)"
1.10.4
```

L'outil peut aussi nous mettre dans l'environnement de développement d'un logiciel spécifique. Par exemple, quelqu'un voulant bidouiller GIMP peut récupérer le source puis se mettre dans un environnement d'où on pourra recompiler la bête :

```
$ tar xf `guix build --source gimp`
$ cd gimp-2.8.14
$ guix environment gimp --container
[env]# echo $PATH
/gnu/store/...-profile/bin:/gnu/store/...-profile/sbin
[env]# echo $C_INCLUDE_PATH
/gnu/store/...-profile/include
[env]# echo $PKG_CONFIG_PATH
/gnu/store/...-profile/lib/pkgconfig
[env]# ./configure && make
```

Ce document est la propriété exclusive de Johann Locatelli(johann.locatelli@businessdecision.com)

Ici **guix environment** a démarré un Bash avec l'invite **[env]** dans lequel tous les paquets nécessaires pour compiler GIMP sont disponibles, et où toutes les variables requises sont définies. Plus simple que de le faire à la main, et sans interférence sur le reste du système !

L'option **--container** est facultative ; elle permet de créer l'environnement dans un conteneur isolé du reste du système, où seuls une partie de **/gnu/store** et le répertoire courant sont visibles, grâce à l'utilisation des *user namespaces* du noyau. Cela garantit d'avoir un environnement « propre » et isolé.

Un prochain article explorera l'utilisation de **guix environment** pour du développement Python.

7 Bidouiller la distrib'

Guix est conçu pour faciliter la bidouille. En ligne de commandes, on peut déjà construire ou installer un paquet en précisant un code source différent et/ou des dépendances différentes :

```
# Compile une "release candidate" de Emacs.
$ guix build emacs --with-source=./emacs-25.1rc2.tar.gz

# Recompile Git en remplaçant OpenSSL par LibreSSL dans tout
# son arbre de dépendances.
$ guix build git --with-input=openssl=libressl
```

Puisque toutes les structures de données et interfaces de programmation de Guix sont exposées, on peut aussi définir des variantes de paquets existants. Par exemple, pour créer une variante de Emacs qui ne dépende pas de **D-Bus**, on peut définir une variable **emacs-sans-dbus** dont la valeur est un paquet qui hérite du paquet **emacs**, mais retire la dépendance sur D-Bus, puis ajouter le fichier à **GUIX_PACKAGE_PATH** :

```
$ cat > /tmp/my-emacs.scm <<EOF
(define-module (my-emacs)
  #:use-module (guix packages)
  #:use-module (gnu packages emacs)
  #:use-module (srfi srfi-1))  ;manipulation de listes

(define-public emacs-sans-dbus
  (package (inherit emacs)
    (name "emacs-sans-dbus")
    (inputs (alist-delete "dbus" (package-inputs emacs)))))
EOF

$ export GUIX_PACKAGE_PATH=/tmp
$ guix package --list-available=emacs
emacs   24.5    out     gnu/packages/emacs.scm:69:2
[...]
emacs-sans-dbus 24.5    out     my-emacs.scm:7:2
$ guix build emacs-sans-dbus --dry-run
La dérivation suivante serait compilée:
   /gnu/store/...-emacs-sans-dbus-24.5.drv
```

Les commandes ont automatiquement pris en compte notre Emacs personnalisé, et celui-ci va magiquement suivre les changements et mises à jour faites au paquet **emacs** de la distribution, avec juste notre modification.

Les usagers peuvent donc facilement maintenir un dépôt de paquets privé avec leurs personnalisations, à la *personal package archive* (PPA), ou encore publier leurs propres définitions de paquets. Un aspect que nous n'avons qu'effleuré ci-dessus est la « programmabilité » : on peut par exemple écrire des fonctions qui renvoient des paquets en fonction des paramètres, ou encore réécrire le graphe de dépendance d'un paquet donné, comme le fait l'option **--with-input** ci-dessus. La frontière entre utilisation et développement de la distribution est floue !

Conclusion

GNU Guix permet à des utilisateurs non privilégiés d'installer des paquets de façon transactionnelle, de créer des environnements logiciels contrôlés, et de personnaliser la distribution. Guix fournit actuellement plus de 3 200 logiciels libres. Guix et **GuixSD** sont encore considérés en version *beta*, mais s'approchent dangereusement de la 1.0. La dernière version est sortie fin mars 2016, fruit du travail d'une cinquantaine de personnes - rejoignez-nous ! Dans un prochain article, nous verrons comment cela se généralise à une distribution tout entière avec GuixSD. ■

Références

[1] « *The LWN.net Linux Distribution List* », https://lwn.net/Distributions/

[2] Site officiel de *GNU Guix* : https://gnu.org/software/guix/

[3] Site officiel de *GNU Guile* : https://gnu.org/software/guile/

[4] Instructions d'installation de Guix : https://gnu.org/software/guix/download/

[5] Site officiel de *Nix* : https://nixos.org/nix/

[6] « Qu'est-ce que le logiciel libre ? », https://gnu.org/philosophy/free-sw.fr.html

Ce document est la propriété exclusive de Johann Locatelli(johann.locatelli@businessdecision.com)

Thème Sécurité RMLL 2016

Conférences et ateliers | Mozilla Paris | 4-6 Juillet

Venez assister gratuitement à 3 jours de conférences et ateliers et échanger avec des hackers, des chercheurs en Sécurité et des leaders de projets Libres :-)

« Les Licornes ne sont pas les seules à avoir droit à la Sécurité et aux Logiciels Libres »

Infos et réservation : https://sec2016.rmll.info/

Partenaires privilégiés :

Savoir-faire Linux · Quarkslab · mozilla · ABUL.ORG · AFUL · Linux Magazine France

Ce document est la propriété exclusive de Johann Locatelli(johann.locatelli@businessdecision.com)

DÉCONNEXION PDO : DU COMPTAGE DE RÉFÉRENCES EN PHP

Gabriel ZERBIB [Architecture, Cloud et gros volumes]

Vous utilisez pour votre projet les composants populaires et éprouvés de l'écosystème PHP et mettez en œuvre les pratiques recommandées dans les manuels. Pourtant votre application se heurte à des problèmes de conception : une piste se cache peut-être ici.

Mots-clés : PHP, Reference Counting, PDO, Closures

Résumé

PHP, comme la plupart des langages dynamiques, offre un mécanisme de collecte de résidus qui affranchit le développeur de se préoccuper de la gestion de la mémoire. Il est notre ami, mais il est préférable de savoir travailler avec lui pour éviter certains désagréments liés à la structure du langage.

Nous allons étudier dans ces lignes le comportement du comptage de références en PHP et l'incidence sur le cycle de vie d'un objet, à travers un exemple : la classe **PDO**.

1 L'objet PDO

On ne présente plus PDO (*PHP Data Objects*), la couche d'abstraction qu'offre PHP pour la manipulation de bases de données, en lieu et place des bonnes vieilles familles de fonctions **mysql_*** ou **pgsql_***, etc.

L'ancienne extension fournissait la fonction **mysql_connect** pour initier la connexion au serveur de bases de données. La classe PDO, elle, prend une approche différente en réalisant la connexion lors de la création d'une instance :

```
$pdo = new PDO($datasource, $username, $password);
```

En revanche, contrairement à l'approche procédurale, il n'y a plus l'équivalent de **mysql_close** pour terminer la connexion et libérer la ressource. Le manuel indique que c'est lors de la destruction de l'objet PDO que la déconnexion est réalisée.

Or, au grand regret de certains, il n'y a pas en PHP d'instruction *delete* comme on aurait eu envie de l'écrire en C++. La destruction d'un objet en PHP se produit automatiquement lorsque l'interpréteur estime qu'il n'est plus utilisé, c'est-à-dire lorsque le programme ne dispose plus d'aucun moyen d'y faire référence. Cela peut se produire, par exemple, si l'on indique explicitement au programme qu'il peut récupérer la variable et ses ressources :

```
$pdo = null;
```

C'est également le cas si la variable qui portait l'objet avait été créée sur la pile, dans une fonction :

Ce document est la propriété exclusive de Johann Locatelli(johann.locatelli@businessdecision.com)

```
function doSomehtingWithDb()
{
  $pdo = new PDO( ... );
  $pdo->query( ... );
}
```

Dans ce listing, la variable **$pdo** est locale à la fonction ; elle est automatiquement détruite à sa sortie, et dans le cas de la bibliothèque PDO cela aboutit à la déconnexion. Plus généralement, comme à l'habitude en PHP, toutes les variables et ressources sont libérées en fin de script, ce qui est aussi le cas des connexions PDO (sauf pour les connexions persistantes qui sortent du cadre de cet article).

2 Cycle de vie des programmes

Programmer en PHP donne parfois l'impression que le cycle de vie d'un script est nécessairement court, car calé sur le cycle d'une seule requête-réponse HTTP. Puisque tout le monde s'emploie à répondre au navigateur dans les meilleurs délais, il est généralement conçu que toute ressource allouée dans un script PHP (connexion à une base de données, ouverture d'un fichier, instances d'objets) sera rapidement libérée de façon automatique, même sans prendre aucune disposition de fermeture explicite. Des usages nouveaux ont pourtant émergé ces dernières années dans l'écosystème PHP, comme l'exécution de scripts longs en ligne de commandes ou de services d'arrière-plan de type *daemon* **[1]**, voire des conteurs applicatifs complets **[2]** en pur PHP. La particularité de ces paradigmes est que la terminaison organique du script peut bien ne survenir qu'au bout d'un temps très long, voire jamais. Dans ces cas, il devient donc nécessaire de libérer soigneusement tout ce dont on n'a plus besoin le plus tôt possible, pour ne pas aboutir à des fuites mémoires ou des blocages de ressources. Mais même dans des programmes simples, il peut s'avérer nécessaire parfois de contrôler explicitement l'instant de déconnexion à une base de données comme nous allons le voir.

3 Connexions simultanées

Soit un script qui lit des informations dans une table (un nom de compte Twitter, une adresse de vidéo YouTube, etc.) puis appelle des API distantes (extraire les mille derniers tweets de chaque abonné, trancher des images dans le clip, etc.). Supposons que la requête SQL initiale est rapide, et que l'exécution complète du script nécessite plusieurs dizaines de secondes, passées essentiellement dans la latence réseau et les API impliquées.

Note

Remarque d'architecture : une bonne solution à ce type de problème devrait faire intervenir plusieurs composants (*workers*) découplés les uns des autres, avec un système de file d'attente de messages, plutôt qu'un script monolithique qui traite toute la chaîne linéairement. C'est néanmoins ce deuxième choix qui nous permettra d'illustrer ici le problème soulevé.

Le programme commence son travail par l'accès à la base de données, puis poursuit sa longue tâche que l'on peut schématiquement représenter comme ceci (instructions simplifiées délibérément pour ne pas surcharger l'exemple) :

```
08: $pdo = new PDO(...);
09:
10: // requêter les infos du job à effectuer (rapide)
11: $data = $pdo->query('select ...')->fetch();
12:
13: // appel aux APIs qui prend un certain temps
14: executer_travail_long_avec_des_api($data);
15:
16: // Fin de script.
```

L'application est capable de gérer potentiellement des dizaines de traitements simultanés, car l'essentiel du travail est effectué à l'extérieur. Pourtant, si l'on ne prend pas soin de fermer la connexion **$pdo**, le traitement concurrent sera limité au nombre de connexions simultanées autorisées par la base de données pour le compte applicatif utilisé. Car une fois la requête initiale effectuée, la connexion demeure, bien qu'en sommeil :

```
mysql> show full processlist;
+-----+----------+-----------+-------+---------+------+----------+----------------------+
| Id  | User     | Host      | db    | Command | Time | State    | Info                 |
+-----+----------+-----------+-------+---------+------+----------+----------------------+
| 112 | app-user | localhost | appdb | Sleep   |   31 |          | NULL                 |
| 113 | app-user | localhost | appdb | Sleep   |   28 |          | NULL                 |
| 145 | app-user | localhost | appdb | Sleep   |   24 |          | NULL                 |
| 159 | app-user | localhost | appdb | Sleep   |   12 |          | NULL                 |
| 165 | app-user | localhost | appdb | Sleep   |    2 |          | NULL                 |
| 170 | app-user | localhost | appdb | Sleep   |    1 |          | NULL                 |
| 174 | root     | localhost | NULL  | Query   |    0 | starting | show full processlist |
+-----+----------+-----------+-------+---------+------+----------+----------------------+
```

Ce document est la propriété exclusive de Johann Locatelli(johann.locatelli@businessdecision.com)

Or il est généralement conseillé de maintenir ce nombre relativement bas. La nécessité d'augmenter le nombre de connexions simultanées autorisées à une base de données, trahit généralement un défaut de conception applicative, plutôt qu'un réel dépassement des capacités de passage à l'échelle.

```
CREATE USER 'app-user'@'localhost' IDENTIFIED BY 'app-password' WITH MAX_USER_CONNECTIONS 10;
Il faut donc modifier le script pour libérer PDO au plus tôt :
11: $data = $pdo->query('select ...')->fetch();
12: $pdo = null;  // <-- Terminaison explicite de la connexion
13: // appel aux APIs qui prend un certain temps
```

Car une fois les données lues depuis la base de données, le script n'a plus besoin d'y retourner pendant toute la durée du traitement des API externes. L'annulation de l'instance **`$pdo`** indique au programme qu'il peut se déconnecter de la base de données, libérant ainsi la place pour d'autres exécutions concurrentes du script.

```
mysql> show full processlist;
+-----+----------+-----------+-------+---------+------+----------+----------------------+
| Id  | User     | Host      | db    | Command | Time | State    | Info                 |
+-----+----------+-----------+-------+---------+------+----------+----------------------+
| 244 | root     | localhost | NULL  | Query   |  0 | starting | show full processlist |
+-----+----------+-----------+-------+---------+------+----------+----------------------+
```

Les accès à la base, pour des requêtes légères, sont alors si furtifs qu'ils n'apparaissent généralement plus dans la *process list*.

4 Les références en PHP

Le langage PHP s'appuie classiquement sur un compteur de références pour pister le nombre de « copies » d'une variable en mémoire. Les affectations suivantes disent au moteur que l'instance effective de l'objet de classe PDO est référencée deux fois :

```
$pdo1 = new PDO(...);
$pdo2 = $pdo1;
```

Par suite, la mise à **`null`** d'une des variables n'a aucune incidence sur l'instance (il existe deux poignées sur la valise ; en supprimer une ne compromet pas la manipulation du bagage).

```
$pdo1 = null;
faire_quelque_chose_avec_la_db($pdo2); // ok
C'est seulement lorsque la dernière référence d'un objet est perdue, que le moteur PHP
détruit l'objet sous-jacent.
$pdo2 = null; // conduit à la destruction de l'instance PDO
```

La vigilance s'impose donc, quant à l'utilisation de la classe PDO dans des scripts à exécution longue. L'augmentation par inadvertance du compteur de références pour notre objet **`$pdo`** conduirait au maintien de la connexion jusqu'à la fin du script. Or, il est parfois complexe de bien comprendre où se cachent les références dans les langages dynamiques.

5 Fermetures et fermeture

Considérons maintenant la situation suivante, où il est fait usage d'un *framework* de routage tel que **Slim [3]** à base de fonctions anonymes :

```
10: // Préparation des dépendances
11: $pdo = new PDO(...);
12:
13: // Déclaration des routes
14: $app->get('/exemple-precedent', function
(Request $req, Response $res) use ($pdo) {
15:   $data = $pdo->query(...)->fetch();
16:   $pdo = null;    //  <- tentative de
déconnexion
17:   // ... traitement long ici
18: });

// D'autres routes ici...

50: // Déclenchement de l'analyse de l'URI
et invocation des handlers de routes par Slim
51: $app->run();
52: // Fin du programme.
```

Routage

Le routage dans le cadre d'une application web est l'association d'un morceau de code à une famille d'URIs donnée. Il est souvent réalisé par la mise en œuvre de *closures*, ou fermetures. Une fermeture est un bloc de code, le plus souvent obtenu par la déclaration d'une fonction anonyme, ayant la particularité d'enfermer l'ensemble des variables disponibles dans son *scope* de déclaration, afin de les rendre disponibles pour son exécution même si elle est invoquée depuis un *scope* différent.

La clause **`use ($pdo)`** en ligne 14, qui permet d'indiquer à la fonction anonyme qu'elle doit enfermer ladite variable de façon à la rendre utilisable à l'intérieur, constitue une incrémentation de son compteur de référence. Aussi en ligne 16, la tentative **`$pdo = null`**

Ce document est la propriété exclusive de Johann Locatelli(johann.locatelli@businessdecision.com)

dans le corps du *handler* de l'URI /exemple-precedent est de bonne volonté, mais reste un échec : la copie locale de la référence **$pdo** est bien annulée, mais pas la référence enfermée par la *closure*, et l'objet sous-jacent demeure intact. La fonction anonyme poursuit son existence jusqu'à la fin de l'exécution de la méthode **$app->run()**, et ce n'est donc qu'à la fin du programme que l'instance PDO sera libérée et la connexion fermée.

6 Le destructeur

Bien qu'il ne soit pas possible de forcer la destruction d'un objet tant que son compteur de références n'est pas nul, il est toutefois possible d'écrire du code que le moteur promet d'exécuter automatiquement lors de la libération de l'instance. Le gestionnaire à implanter est la *magic method* peu connue **__destruct()**. Il est recommandé de lire attentivement dans le manuel **[4]** les particularités de son fonctionnement.

Nous allons l'utiliser pour mettre en pratique sur une classe simple, les observations étudiées plus haut.

```
class K
{
  private $name;

  public static function timer() { return time() - $_SERVER['REQUEST_
TIME']; }

  public function __construct($name) {
    echo self::timer() . ' : ' . __FUNCTION__ . ' ' . ($this->name =
$name) .  PHP_EOL;
  }
  public function __destruct() {
    echo self::timer() . ' : ' . __FUNCTION__ . ' ' . $this->name .
PHP_EOL;
  }
}
```

Cette classe annonce sa construction et sa destruction, en indiquant le temps écoulé depuis le lancement du script (grâce à la valeur super-globale **$_SERVER['REQUEST_TIME']**).

Utilisons-la dans trois situations différentes : destruction explicite simple, capture d'instance dans une fermeture, et destruction automatique d'instance par la terminaison du programme.

```
10: $q = new K('q');
11: $k = new K('k');
12: $x = new K('x');
13:
14: $c = function () use ($k) {
15:   $k = null;
16: };
17:
18: $c();      // sans effet
19: $k = null; // sans effet
```


Ce document est la propriété exclusive de Johann Locatelli(johann.locatelli@businessdecision.com)

DBM TECHNOLOGIES

L'apprentissage à l'ère du numérique

*Des **solutions e-learning** adaptées à votre entreprise*

L'avantage du e-learning

- Des cours accessibles quand vous le souhaitez pour vos équipes ou vos clients.
- Des plate-formes Open Source compatibles avec la norme SCORM.
- Des fonctions complémentaires (forum, documents partagés, agenda...).

Pour plus d'informations,
Contactez-nous

www.dbmtechnologies.com
contact@dbmtechnologies.com
+33 (0)4 72 83 52 28

```
20: $q = null; // destruction de q car non capturé
21:
22: sleep(6);
23: $c = null; // Libération de la closure, et donc de k
24:
25: echo K::timer() . ' : FIN.' . PHP_EOL;
L'exécution de ce script affiche :
0 : __construct q
0 : __construct k
0 : __construct x
0 : __destruct q
6 : __destruct k
6 : FIN
6 : __destruct x
```

Trois instances sont construites dès le début du script (lignes 10 à 12).

La fermeture **$c** en ligne 14 (en réalité, la clause **use**) emprisonne une référence de **k**, que l'on libère automatiquement en restituant la référence sur la fonction anonyme en ligne 23 après le **sleep**.

L'instance **q** n'est impliquée dans aucune clause **use**, et elle est donc bien libérée dès que la variable **$q** est annulée en ligne 20. Enfin, l'instance **x** n'est jamais explicitement restituée : elle est donc détruite automatiquement lorsque le script termine son exécution (après la fin de l'instruction de dernière ligne 25).

7 Encapsulation

La solution, dans notre programme de routage de la section 5, est d'encapsuler l'objet PDO dans un *pattern* de composition :

```
class PDOWrapper
{
  /** @var PDO */
  private $pdo;

  public function __construct( ... ) { $this->pdo = new PDO( ... ); }
  /** @return PDO */ public function getPDO() { return $this->pdo; }
  public function terminate() { $this->pdo = null; }
}
Alors, le programme précédent devient :
10: // Préparation des dépendances
11: $pdoWrapper = new PDPWrapper(...);
12:
13: // Routes
14: $app->get(..., function (...) use ($pdoWrapper) {
15:   $pdoWrapper->getPDO()->query( ... );
16:   $pdoWrapper->terminate();
17: });
```

Cette fois l'instance PDO est bien détruite lors du **$pdoWrapper->terminate()**, car la *closure* a enfermé une référence vers **$pdoWrapper** sans pour autant augmenter le compteur de références de l'objet PDO qui la compose.

Mais attention : pour qu'une telle solution donne satisfaction, il faut se garder soigneusement d'attraper une poignée explicite sur l'objet PDO encapsulé. En d'autres termes, ne jamais stocker :

```
$pdo = $pdoWrapper->getPDO();
```

En effet, tout le bénéfice de l'encapsulation disparaîtrait à moins d'aller plus loin en employant le *pattern* du Proxy, supprimant la méthode **getPDO()** et répliquant toutes les méthodes de la classe PDO. Malheureusement, l'implémentation d'un tel *wrapper* n'est pas toujours envisageable (notamment lorsqu'on utilise un ORM du marché du type **Doctrine [5]**). Et lorsque l'application est éclatée en de nombreux fichiers ou fait appel au *pattern* d'injection de dépendances, on n'a pas toujours une maîtrise suffisante des différentes ressources impliquées.

Conclusion

À titre personnel, je trouve dommage que la classe PDO n'expose pas de méthode **close** comme on pourrait la trouver dans d'autres modules tels que **Memcache** ou **ZipArchive** par exemple. Laisser reposer le cycle de vie des ressources sur celui du *garbage collector* ne me semble pas être un choix judicieux de la part des auteurs du langage. J'ai proposé d'écrire une RFC pour cette classe, mais elle n'a pas suscité l'intérêt des principaux mainteneurs qui estiment **[6]** que le problème trouve sa solution dans le « userspace » comme évoqué plus haut.

Chaque point de vue se défend, mais il faut reconnaître que le leur (mise à part l'incohérence de philosophie avec d'autres extensions PHP) a le mérite de forcer le développeur à se poser les bonnes questions sur son utilisation des ressources (bref, faire du C). ■

Références

[1] Daemonizable commands for Symfony : https://github.com/mac-cain13/daemonizable-command

[2] appserver.io : http://appserver.io/

[3] Slim : http://slimframework.com

[4] Constructeurs et destructeurs PHP : http://php.net/manual/en/language.oop5.decon.php

[5] Doctrine : http://www.doctrine-project.org/

[6] Discussion « PDO Close Connection » : http://news.php.net/php.internals/90840

Ce document est la propriété exclusive de Johann Locatelli(johann.locatelli@businessdecision.com)

ACTUELLEMENT DISPONIBLE

LINUX PRATIQUE N°95 !

DIFFUSEZ LA TNT HD DANS TOUTE LA MAISON !

NE LE MANQUEZ PAS

CHEZ VOTRE MARCHAND DE JOURNAUX ET SUR :

www.ed-diamond.com

Ce document est la propriété exclusive de Johann Locatelli(johann.locatelli@businessdecision.com)

PROGRAMMER AVEC GTK+

Christophe BORELLY [Professeur de l'ENSAM – IUT de Béziers]

Cet article va vous présenter les rudiments de la programmation d'applications graphiques en C à l'aide de la boîte à outils GTK+ (GIMP Toolkit).

Mots-clés : GTK+, Langage C, Applications graphiques, Événements, Styles, Internationalisation

Résumé

Dans cet article, vous apprendrez à développer une petite application en C permettant de simuler un système d'authentification utilisateur. Vous découvrirez comment organiser les éléments graphiques dans la fenêtre, comment gérer les événements utilisateur (comme le clic sur un bouton), comment paramétrer le style des éléments avec du CSS (*Cascading Style Sheet*) comme en HTML, et enfin comment internationaliser l'application (c'est-à-dire afficher les textes en fonction de la langue utilisée).

L'histoire de GTK+ **[1]** a commencé vers 1996 quand Peter MATTIS a voulu remplacer la boîte à outils **Motif** pour le développement de **GIMP** (*GNU Image Manipulation Program*). Cette librairie s'appelait initialement **GTK** (*GIMP ToolKit*) et a été renommée **GTK+** lors de sa réécriture en programmation orientée objet. Elle est écrite en C/C++, mais un très grand nombre d'autres langages sont supportés comme Python, Perl, PHP, etc.

GTK+ fait partie intégrante du projet **GNU** et a été également adopté par un grand nombre d'environnements de bureau (**GNOME**, **XFCE**, **Unity**, **Cinnamon**, etc.). La documentation en ligne se trouve d'ailleurs sur le site des développeurs GNOME **[2]**.

La dernière version stable de GTK+ (au moment de l'écriture de cet article) est la 3.18.7. Et même si on trouve en général deux versions sur un système Linux (GTK+2 et GTK+3), je ne présenterai ici que des exemples GTK+3 avec le code le plus récent possible (sans éléments dépréciés).

Il faut savoir qu'il y a quand même pas mal de différences de programmation avec GTK+2, notamment pour la gestion des styles et le positionnement des éléments dans une fenêtre.

Comme fil conducteur de cet article, je vous propose de créer une petite application permettant à un utilisateur de fournir des identifiants (nom d'utilisateur et mot de passe).

1 Programme de base pour GTK+

La programmation en GTK+ utilise des éléments graphiques appelés **widgets** (de la contraction de *window* (fenêtre) et *gadget*) et elle est basée sur les actions (ou événements) réalisées par les utilisateurs. On va donc retrouver un programme principal très petit (fonction **main**) et une ou plusieurs fonctions associées aux événements que l'on désire gérer dans l'application.

Voici donc un premier exemple simplissime qui crée juste une application composée d'une fenêtre vide centrée sur l'écran :

```
01: #include <gtk/gtk.h>
...
06: static void startApplication(GtkApplication *app,gpointer data) {
07:   GtkWidget *window=gtk_application_window_new(app);
08:   gtk_window_set_title(GTK_WINDOW(window),"Application GTK+3 v1");
```

EN COUVERTURE
Ce document est la propriété exclusive de Johann Locatelli(johann.locatelli@businessdecision.com)

```
09:   gtk_window_set_position(GTK_WINDOW(window),GTK_WIN_POS_CENTER);
10:   gtk_window_set_default_size(GTK_WINDOW(window),400,100);
11:   gtk_widget_show_all(window);
12: }
...
16: int main(int argc,char *argv[]) {
17:   GtkApplication *app=gtk_application_new("fr.iutbeziers.gtk3-01",
18:                                           G_APPLICATION_FLAGS_NONE);
19:   g_signal_connect(app,"activate",G_CALLBACK(startApplication),NULL);
20:   int status=g_application_run(G_APPLICATION(app),argc,argv);
21:   g_object_unref(app);
22:   return status;
23: }
```

Le programme principal (fonction **main**, lignes 16 à 23) contient à la ligne 17 la création de l'objet **GtkApplication** (il n'existe pas en GTK+2 d'ailleurs) permettant d'identifier l'exécutable de façon unique dans le système en l'associant au nom **fr.iutbeziers.gtk3-01**. On utilise un principe de nommage basé sur les noms fictifs de domaines inversés comme en programmation Android (pour ceux qui ont déjà joué avec le petit robot vert). Il suffit ici que le nom soit unique.

On indique ensuite ligne 19, que lorsque le signal **activate** est reçu, le programme doit démarrer la fonction **startApplication**. On ne lance réellement l'application qu'à la ligne 20 avec les arguments du programme principal (**argc** et **argv**). En fait, c'est cette fonction qui génère le signal **activate** et qui démarre une boucle d'attente des événements utilisateur.

Dans la fonction **startApplication** (lignes 6 à 12), on crée l'objet **window** représentant la fenêtre de l'application, puis on fixe le titre (ligne 8), on centre la fenêtre à la ligne 5 et on spécifie la taille par défaut en pixel de la fenêtre (ligne 10). La dernière instruction (ligne 11) affiche la fenêtre et ses composants (s'il y en a).

Pour compiler ce premier exemple, il faut bien sûr avoir installé le paquet de développement pour GTK+3 (contenant les fichiers d'en-tête en général). Sur les distributions à base de Debian (dont Raspbian), il suffit d'ajouter le paquet **libgtk-3-dev** :

```
$ sudo dpkg -l | grep libgtk-3
ii  libgtk-3-0:amd64                        3.10.8~8+qiana
 amd64        GTK+ graphical user interface library
ii  libgtk-3-bin                            3.10.8~8+qiana
 amd64        programs for the GTK+ graphical user interface library
ii  libgtk-3-common                         3.10.8~8+qiana
 all          common files for the GTK+ graphical user interface library
$ sudo apt-cache search libgtk-3
libgtk-3-0 - GTK+ graphical user interface library
libgtk-3-0-dbg - GTK+ libraries and debugging symbols
libgtk-3-bin - programs for the GTK+ graphical user interface library
libgtk-3-common - common files for the GTK+ graphical user interface library
libgtk-3-dev - development files for the GTK+ library
libgtk-3-doc - documentation for the GTK+ graphical user interface library
$ sudo apt-get install libgtk-3-dev
```

Ensuite, si la distribution intègre l'outil **pkg-config**, on peut obtenir les drapeaux de compilation avec les commandes suivantes :

```
$ pkg-config --cflags gtk+-3.0
-pthread -I/usr/include/gtk-3.0 -I/usr/include/atk-1.0 -I/usr/include/
at-spi2-atk/2.0 -I/usr/include/pango-1.0 -I/usr/include/gio-unix-2.0/
-I/usr/include/cairo -I/usr/include/gdk-pixbuf-2.0 -I/usr/include/
glib-2.0 -I/usr/lib/x86_64-linux-gnu/glib-2.0/include -I/usr/include/
harfbuzz -I/usr/include/freetype2 -I/usr/include/pixman-1 -I/usr/
include/libpng12
$ pkg-config --libs gtk+-3.0
-lgtk-3 -lgdk-3 -latk-1.0 -lgio-2.0 -lpangocairo-1.0 -lgdk_pixbuf-2.0
-lcairo-gobject -lpango-1.0 -lcairo -lgobject-2.0 -lglib-2.0
```

Ce qui peut donc donner au final la commande de compilation :

```
$ CFLAGS=$(pkg-config --cflags gtk+-3.0)
$ LIBS=$(pkg-config --libs gtk+-3.0)
$ gcc -Wall $CFLAGS gtk3-01-application.c -o gtk3-01-application $LIBS
```

Le résultat obtenu avec cette application est visible en figure 1.

Fig. 1 : Fenêtre générée par notre première application.

Note

Vous pouvez récupérer les sources de cet article sur le GitHub de GLMF. J'ai créé un fichier **Makefile** qui permet d'automatiser la compilation, il suffit de taper **make** dans le répertoire des sources.

Si vous voulez internationaliser l'application (voir section 6), il vous faudra taper les commandes suivantes :

```
$ make po
$ sudo make -C po install
```

EN COUVERTURE

Ce document est la propriété exclusive de Johann Locatelli(johann.locatelli@businessdecision.com)

2 Ajouts de widgets dans la fenêtre

Il existe évidemment un grand nombre de widgets **[2]** dont le nom est en général assez explicatif comme **GtkLabel**, **GtkButton**, **GtkEntry**, **GtkImage**, etc. Mais il faut cependant faire un peu attention aux éléments dépréciés si on veut écrire un code propre (la documentation est là pour ça).

Afin d'organiser logiquement ces éléments, on se sert de ce que l'on appelle des conteneurs de présentation (*layout containers*). On dispose par exemple de **GtkBox** (boîtes horizontales ou verticales), de **GtkGrid** (remplacement de **GtkTable** en GTK+3), de **GtkFlowBox** (depuis la version 3.12), etc.

Dans ce deuxième exemple, nous allons ajouter deux labels et deux zones de texte ainsi qu'un bouton de validation. Il y a plusieurs façons de réaliser cela suivant le ou les conteneurs que l'on veut utiliser. Comme en général, les systèmes d'authentification sont disposés sous forme de tableau à 3 lignes et 2 colonnes, le plus judicieux est de se servir d'un élément **GtkGrid** (nouveau d'ailleurs en GTK+3).

On crée donc simplement une grille avec la fonction de la ligne 14 du code suivant, puis on l'ajoute à la fenêtre (ligne 15). Il faut d'ailleurs savoir qu'une fenêtre ne peut contenir qu'un seul élément et celui-ci est en général un conteneur. En GTK+3, il y a 20 conteneurs possibles **[3]**. On peut également noter dans cet exemple, que l'on a fixé une marge de **10** pixels tout autour du contenu de la fenêtre avec la fonction utilisée ligne 12.

Après cela, on utilise plusieurs fonctions (lignes 16 à 19) afin de fixer l'espacement entre les lignes et les colonnes de la grille. Puis on centre la grille en vertical et en horizontal dans la fenêtre. Il y a 3 autres valeurs possibles pour l'alignement : **GTK_ALIGN_FILL**, **GTK_ALIGN_START** et **GTK_ALIGN_END**.

```
...
06: static void startApplication(GtkApplication *app,gpointer
data) {
...
12:   gtk_container_set_border_width(GTK_CONTAINER(window),10);
13:
14:   GtkWidget *grid=gtk_grid_new();
15:   gtk_container_add(GTK_CONTAINER(window),grid);
16:   gtk_grid_set_row_spacing(GTK_GRID(grid),2);
17:   gtk_grid_set_column_spacing(GTK_GRID(grid),5);
18:   gtk_widget_set_valign(grid,GTK_ALIGN_CENTER);
19:   gtk_widget_set_halign(grid,GTK_ALIGN_CENTER);
...
```

Il reste maintenant à ajouter les éléments dans la grille en indiquant les numéros de colonnes et de lignes (en partant de **0**) puis en précisant le nombre de cases à utiliser en horizontal et en vertical. Si on veut ajouter un *widget* dans la première case, on indiquera **0,0** (pour la ligne **0** et la colonne **0**) et **1,1** (pour indiquer **1** case en largeur et **1** en hauteur). Les lignes du code ci-dessous ajoutent sur la première ligne (**row=0**), le label et la zone de texte pour le nom d'utilisateur avec la fonction **gtk_grid_attach()**. Vous avez également à la ligne 26, un exemple de fonction permettant de fixer, comme on peut le voir sur la figure 2, le texte de suggestion en grisé de la zone. On utilise en général le mot anglais *hint* en programmation. À la ligne suivante, en commentaire, je vous donne la fonction qui permet de fixer la taille en caractères de l'élément (par défaut, c'est 20 caractères).

```
...
21:   int col=0,row=0;
22:   GtkWidget *label_user=gtk_label_new("UserName");
23:   gtk_grid_attach(GTK_GRID(grid),label_user,col,row,1,1);
24:   col++;
25:   GtkWidget *entry_user=gtk_entry_new();
26:   gtk_entry_set_placeholder_text(GTK_ENTRY(entry_user),"UserName");
27:   //gtk_entry_set_width_chars(GTK_ENTRY(entry_user),25);
28:   gtk_grid_attach(GTK_GRID(grid),entry_user,col,row,1,1);
...
```

On ajoute ensuite la ligne contenant le mot de passe au conteneur. J'ai également laissé plusieurs commentaires dans cette partie du code. À la ligne 35, la fonction permet de fixer la taille maximum à **8** caractères pour le mot de passe. Ensuite, la ligne 37 peut être utilisée pour changer le caractère affiché à chaque lettre tapée. Par défaut, GTK+3 utilise un gros point noir de code **9679** (voir figure 2). Enfin, aux lignes 38 et 39, on peut notifier à l'application que cet élément va contenir un mot de passe. La seule fonction vraiment utile pour le mot de passe est celle de la ligne 36 qui indique qu'il faut cacher les caractères.

```
...
29:   col=0;row++;
30:   GtkWidget *label_pass=gtk_label_new("Password");
31:   gtk_grid_attach(GTK_GRID(grid),label_pass,col,row,1,1);
32:   col++;
33:   GtkWidget *entry_pass=gtk_entry_new();
34:   gtk_entry_set_placeholder_text(GTK_ENTRY(entry_
pass),"Password");
35:   //gtk_entry_set_max_length(GTK_ENTRY(entry_pass),8);
36:   gtk_entry_set_visibility(GTK_ENTRY(entry_pass),FALSE);
37:   //gtk_entry_set_invisible_char(GTK_ENTRY(entry_pass),42);
38:   //gtk_entry_set_input_purpose(GTK_ENTRY(entry_pass),
39:                                 //GTK_INPUT_PURPOSE_PASSWORD);
40:   gtk_grid_attach(GTK_GRID(grid),entry_pass,col,row,1,1);
...
```

EN COUVERTURE
Ce document est la propriété exclusive de Johann Locatelli(johann.locatelli@businessdecision.com)

Pour la dernière ligne de la grille, contenant seulement un bouton, il faut noter que celui-ci est réparti sur 2 colonnes en largeur et 1 en hauteur (voir ligne 48). J'ai laissé aux lignes 43 à 47, un exemple de paramétrage permettant au bouton de ne pas remplir la totalité de la place qui lui est allouée, de le centrer en vertical et en horizontal et de fixer sa taille préférée.

```
...
41:   col=0;row++;
42:   GtkWidget *btn=gtk_button_new_with_label("Authentication");
43:   //gtk_widget_set_hexpand(btn,FALSE);
44:   //gtk_widget_set_vexpand(btn,FALSE);
45:   //gtk_widget_set_halign(btn,GTK_ALIGN_CENTER);
46:   //gtk_widget_set_valign(btn,GTK_ALIGN_CENTER);
47:   //gtk_widget_set_size_request(btn,220,50);
48   gtk_grid_attach(GTK_GRID(grid),btn,col,row,2,1);
...
```

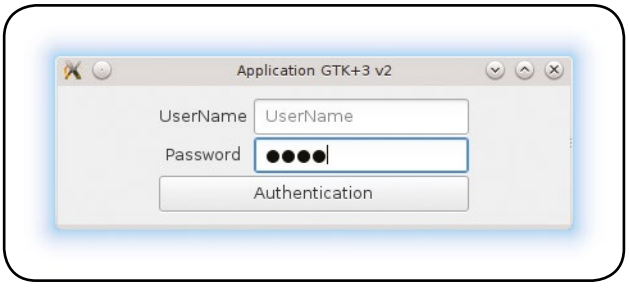


Fig. 2 : Deuxième application avec 2 labels, 2 zones de texte et un bouton.

Note

Il faut savoir que si on indique la valeur **-1** pour la largeur ou la hauteur, dans la fonction **gtk_widget_set_size_request()**, la dimension en question n'est pas prise en compte. Donc si on veut fixer seulement la largeur du *widget* **w1** à **50** pixels et régler uniquement une hauteur de **30** pixels pour le *widget* **w2**, on peut écrire :

```
gtk_widget_set_size_request(w1,50,-1);
gtk_widget_set_size_request(w2,-1,30);
```

3 Gestion des actions utilisateurs

Une fois les éléments graphiques ajoutés, on peut exécuter une fonction dès qu'un événement particulier se produit sur un *widget* donné. On appelle cela un signal, mais suivant l'élément en question, il peut y avoir un ou plusieurs signaux possibles. Par exemple pour un **GtkButton**, il y a 6 signaux **[4]** (**activate**, **clicked**, **enter**, **leave**, **pressed** et **release**). Chacun ayant une signification particulière qui est détaillée dans la documentation.

Pour notre application, on veut valider les identifiants fournis quand l'utilisateur clique sur le bouton. Il suffit donc d'associer le signal **clicked** du bouton à une fonction spécifique. La fonction doit cependant suivre un prototype particulier (comme on l'a vu en section 1 avec la fonction **startApplication()**). Deux paramètres sont nécessaires : le premier est un pointeur sur le *widget* qui a généré le signal et le second est un pointeur générique (**gpointer**) vers un paramètre utilisateur.

Dans un premier temps, pour se familiariser avec les signaux, dans le code suivant on va simplement modifier le texte du bouton à chaque clic. On crée donc une variable globale ligne 3 qui servira de compteur puis on génère un message texte intégrant cette valeur à la ligne 6 et enfin on modifie le label du bouton qui a généré le signal (le premier paramètre de la fonction).

L'association de la fonction **click()** avec le signal **clicked** du bouton est réalisée ligne 54. Il faut donner dans l'ordre, le nom du *widget* qui va générer le signal, puis le nom du signal désiré, puis le nom de la fonction à exécuter et enfin le paramètre utilisateur que l'on a fixé à **NULL**, car on ne l'utilise pas pour le moment.

À la ligne 55, vous avez un exemple d'utilisation de ce paramètre utilisateur. Cela sert quand on veut modifier ou récupérer des données d'un ou plusieurs autres *widgets* que l'élément qui a généré le signal. On passe ici la variable **window** à la fonction **click()** et on voit l'utilisation de ce paramètre dans le code des lignes 9 à 12. On teste en premier la valeur du paramètre **data** et s'il est non **NULL**, on fait ensuite un *casting* du pointeur pour ne pas avoir de message d'erreur puis on change le titre de la fenêtre.

```
...
03: int nb=0; // Compteur du nombre de clics
...
08: static void click(GtkWidget *b,gpointer data) {
09:   char msg[10];
10:   snprintf(msg,sizeof(msg),"Click: %d",nb++);
11:   gtk_button_set_label(GTK_BUTTON(b),msg);
12:   // Utilise le parametre indique ligne 55
13:   if (data!=NULL) {
14:     GtkWindow *window=(GtkWindow *)data;
15:     gtk_window_set_title(window,msg);
16:   }
17: }
...
```

Ce document est la propriété exclusive de Johann Locatelli(johann.locatelli@businessdecision.com)

```
21: static void startApplication(GtkApplication *app,gpointer data) {
...
54:   g_signal_connect(btn,"clicked",G_CALLBACK(click),NULL);
55:   //g_signal_connect(btn,"clicked",G_CALLBACK(click),window);
...
```

4 Ajout de la fonction d'authentification

Pour notre application, il faut récupérer les valeurs texte des deux zones de texte et les comparer d'une façon ou d'une autre aux identifiants attendus : avec des valeurs fixées en dur dans le programme, en utilisant un fichier texte ou une base de données, en se servant d'un serveur, etc.

S'il faut récupérer différentes valeurs dans la fonction d'authentification, il y a plusieurs stratégies : soit on crée une structure spéciale contenant les pointeurs vers les deux zones de texte, soit on se sert des variables déjà existantes comme **app** ou **window**. Cette dernière solution est un petit peu plus longue à mettre en place, mais elle est plus évolutive, car on a potentiellement accès à tous les éléments de l'application contrairement au premier cas où il faut modifier manuellement la structure pour ajouter des références vers d'autres widgets.

Je vais tout de même vous présenter les deux méthodes, et vous pourrez ensuite faire votre choix.

4.1 Création d'une structure spéciale

Voici la structure que l'on peut créer pour accéder aux deux zones de texte :

```
typedef struct authEntries {
  GtkWidget *user;
  GtkWidget *pass;
} AUTH;
```

Ensuite, il y a encore deux manières différentes de s'en servir.

Il faut soit définir une variable globale (ou **static**), initialiser les pointeurs et associer le clic sur le bouton à la fonction d'authentification. Notez bien le **&** pour fournir l'adresse de la structure en quatrième paramètre.

```
static AUTH authEnt;
authEnt.user=entry_user;
authEnt.pass=entry_pass;
g_signal_connect(btn,"clicked",G_CALLBACK(auth),&authEnt);
```

La seconde façon consiste à allouer dynamiquement la zone mémoire pour cette structure. Mais il faudra libérer cette mémoire avec **g_free()** dès que l'on n'aura plus besoin de la structure.

```
AUTH *pAuthEnt=g_malloc(sizeof(AUTH));
pAuthEnt->user=entry_user;
pAuthEnt->pass=entry_pass;
g_signal_connect(btn,"clicked",G_CALLBACK(auth),pAuthEnt);
```

Enfin, dans la fonction d'authentification, on récupère la structure et on peut ensuite accéder aux zones de texte.

```
if (data!=NULL) {
  AUTH *authEnt=(AUTH *)data;
  if (authEnt->user!=NULL && authEnt->pass!=NULL) {
    ...
  }
}
```

4.2 Utilisation de la fenêtre comme paramètre utilisateur

Dans le cas de l'utilisation de la fenêtre comme paramètre utilisateur, il faut trouver un moyen de récupérer les éléments contenus dans la fenêtre. On peut le faire grâce à la fonction **gtk_container_get_children()** qui renvoie la liste des *widgets* enfants d'un conteneur. Ensuite, pour retrouver un *widget* particulier, il suffit de lui avoir donné un nom unique avec la fonction **gtk_widget_set_name()** et en parcourant la liste, on cherche le nom qui nous intéresse. Une difficulté supplémentaire réside dans le fait que parmi les éléments de la liste, il peut y avoir à nouveau des conteneurs. La solution la plus simple à ce problème est de créer une fonction récursive. J'ai amélioré un code trouvé sur **stackoverflow** dans l'exemple ci-dessous pour la fonction **findChildByName()**.

```
88: GtkWidget* findChildByName(GtkWidget *container,const gchar *name) {
89:   if (GTK_IS_WIDGET(container)) {
90:     const gchar *cName=gtk_widget_get_name(container);
91:     if (g_strcmp0(cName,name)==0) { // Le nom du conteneur
correspond
92:       return container;
93:     }
94:     if (GTK_IS_BIN(container)) { // Conteneur avec un seul enfant
95:       GtkWidget *child=gtk_bin_get_child(GTK_BIN(container));
96:       return findChildByName(child,name);
97:     }
98:     if (GTK_IS_CONTAINER(container)) { // Recherche dans la liste
des enfants
99:       GList *childs=gtk_container_get_children(GTK_
CONTAINER(container));
100:       GList *item;
```

EN COUVERTURE
Ce document est la propriété exclusive de Johann Locatelli(johann.locatelli@businessdecision.com)

```
101:      for(item=childs;item!=NULL;item=g_list_next(item)) {
102:        GtkWidget *widget=findChildByName(item->data,name);
103:        if (widget!=NULL) {
104:          g_list_free(childs);
105:          return widget;
106:        }
107:      }
108:      if (childs!=NULL) g_list_free(childs);
109:    }
110:  }
111:  return NULL;
112: }
```

On a bien sûr donné un nom à chaque zone de texte pour que la recherche par nom soit possible (voir lignes 55 et 63 suivantes). Comme nous le verrons en section suivante, le nom des éléments peut également être utilisé pour appliquer un style particulier.

À la ligne 70, on passe la fenêtre en paramètre de la fonction appelée lors du clic sur le bouton.

```
...
54:   GtkWidget *entry_user=gtk_entry_new();
55:   gtk_widget_set_name(entry_user,"user");
...
62:   GtkWidget *entry_pass=gtk_entry_new();
63:   gtk_widget_set_name(entry_pass,"pass");
...
70:   g_signal_connect(btn,"clicked",G_CALLBACK(auth),window);
...
```

Je vous laisse adapter le code suivant après la ligne 22 pour réaliser l'authentification que vous préférez. Dans une application réelle, si le mot de passe est correct, il faudrait effacer le contenu de la fenêtre (voir **gtk_container_get_children()** et **gtk_widget_destroy()**) et ajouter de nouveaux éléments à la fenêtre. C'est un bon exercice, à vous de jouer.

```
10: static void auth(GtkWidget *b,gpointer data) {
11:   GtkWidget *user=NULL;
12:   GtkWidget *pass=NULL;
13:   if (data!=NULL) {
14:     GtkWidget *window=(GtkWidget *)data;
15:     user=findChildByName(window,"user");
16:     pass=findChildByName(window,"pass");
17:   }
18:   if (user!=NULL&&pass!=NULL) {
19:     const gchar *u=gtk_entry_get_text(GTK_ENTRY(user));
20:     const gchar *p=gtk_entry_get_text(GTK_ENTRY(pass));
21:     //printf("User: %s\n",u);
22:     //printf("Pass: %s\n",p);
...
29:     gtk_entry_set_text(GTK_ENTRY(pass),"");
30:   }
31: }
```

5 Mise en forme et styles

La version 3 de GTK+ propose un système de style très proche du CSS utilisé en HTML. C'est là d'ailleurs une des grandes différences avec GTK+ version 2, dans lequel on se servait plutôt de fichiers de ressources.

Le format et la liste des propriétés CSS supportées sont disponibles dans la documentation de **GtkCssProvider** **[5]**. Comme en CSS, on définit les styles par rapport à différents critères. Ce peut être par exemple en fonction de leur classe (ligne 1), ou du type d'objet comme **GtkWindow**, **GtkGrid**, **GtkLabel** (lignes 2, 3, 4 et 5 suivantes), ou bien par rapport à leur état (lignes 12 et 17) ou enfin en utilisant leurs identifiants (ligne 21). Notez ici que les boutons créés intègrent un **GtkLabel** (**gtk_button_new_with_label()**), on indique donc ligne 5 que le style s'applique à un **GtkButton** ou bien un **GtkButton** suivi d'un **GtkLabel**.

Voici le fichier de style **style.css** utilisé en tant qu'exemple sur notre application :

```
01 .button {margin:0;padding:5px;}
02 GtkWindow {background-color:#F9E589;}/* orange */
03 GtkGrid {background-color:#CCDBFF;}/* bleu */
04 GtkLabel {padding:2px;font:Sans 12;font-weight:bold;}
05 GtkButton, GtkButton GtkLabel {
06   border-radius:5px;
07   border-width:2px;
08   font:Sans 12;
09   background-color:#F98F89;/* rouge */
10   border-color:#CE3551;
11 }
12 GtkButton:hover, GtkButton:hover GtkLabel {
13   background-color:#89C1F9;/* bleu */
14   border-color:#3571AD;
15   transition:500ms ease-in-out;
16 }
17 GtkButton:hover:active, GtkButton:hover:active GtkLabel {
18   background-color:#82ED90;/* vert */
19   border-color:#2C934D;
20 }
21 GtkEntry#user {color:#E81BF7;}/* rose */
```

Il reste à ajouter au code, dans la fonction **startApplication()**, les quelques lignes suivantes pour appliquer le style aux éléments de l'application :

```
GtkCssProvider *provider=gtk_css_provider_new();
GdkDisplay *display=gdk_display_get_default();
GdkScreen *screen=gdk_display_get_default_screen(display);
gtk_style_context_add_provider_for_screen(screen,
                                          GTK_STYLE_
PROVIDER(provider),
                                           GTK_STYLE_PROVIDER_
PRIORITY_USER);
```

Ce document est la propriété exclusive de Johann Locatelli(johann.locatelli@businessdecision.com)

```
GError *error=NULL;
gtk_css_provider_load_from_path(provider,"style.css",&error);
if (error!=NULL) {
  fprintf(stderr,"Unable to load CSS file: %s !\n",error->message);
  g_error_free(error);
}
g_object_unref(provider);
```

Je vous laisse retrouver, avec les commentaires dans le fichier CSS et les figures 3 et 4, les couleurs qui seront appliquées aux différents *widgets*.

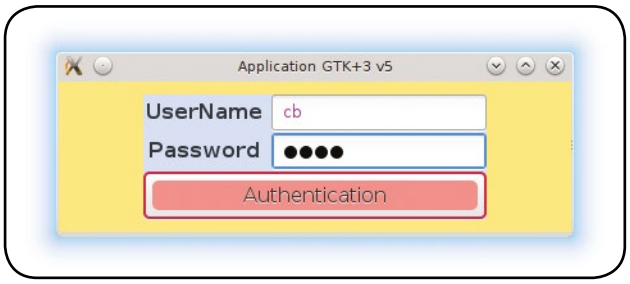


Fig. 3 : Application utilisant le fichier de style CSS.

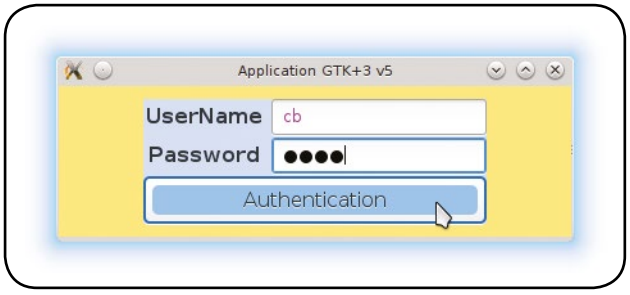


Fig. 4 : Quand la souris passe sur le bouton, la couleur de fond change.

6 Internationalisation

La dernière chose que nous verrons dans cet article concerne l'internationalisation de l'application. Cela signifie que les textes affichés dans la fenêtre vont s'adapter en fonction du paramétrage linguistique du système. Sur un système **fr_FR**, on aura les textes en français et pour un système **en_US**, ce sera en anglais.

On va utiliser pour cela, la librairie **gi18n** de la **GLib [6]** et les outils **xgettext** et **msgfmt**. Le principe consiste à marquer dans le code les chaînes de caractères à transformer en fonction de la langue, puis de fabriquer un modèle de fichier de traduction (**.pot** - *Portable Object Template*) et décliner ensuite ce fichier dans les langues désirées (**.po** - *Portable Object*). Il reste après à compiler les fichiers PO pour en obtenir la version binaire (**.mo** - *Machine Object*) et à les placer dans l'arborescence des fichiers de localisation du système.

Dans le code suivant, on ajoute en début de fichier, la librairie gi18n et on définit le nom du projet (ligne 8) et le répertoire contenant les locales à la ligne 9 (cela peut être automatisé avec **autoconf**/**automake** par exemple). Puis on utilise la macro **_()** pour marquer les chaînes que l'on veut traduire (voir par exemple la ligne 57).

J'ai créé une fonction (voir lignes 14 à 18) qui indique au programme d'aller chercher automatiquement les traductions dans le répertoire des locales. On appelle celle-ci au début du programme principal.

```
01: #include <gtk/gtk.h>
02: #include <glib/gi18n.h>
...
08: #define PACKAGE "cb-login"
09: #define LOCALE_DIR "/usr/share/locale"
...
14: void loadInternationalisation(char *package,char *locale) {
15:   bindtextdomain(package,locale);
16:   bind_textdomain_codeset(package,"UTF-8");
17:   textdomain(package);
18: }
...
57: GtkWidget *label_user=gtk_label_new(_("UserName"));
...
83: int main(int argc,char *argv[]) {
84:   loadInternationalisation(PACKAGE,LOCALE_DIR);
...
```

Une fois le code source adapté, on peut créer un fichier de modèle de traduction avec **xgettext**. Noter l'option **keyword** et le nom du fichier POT créé (**cb-login.pot**) :

```
$ xgettext --sort-output --no-wrap --escape --keyword=_ \
    --copyright-holder="IUT de Beziers" \
    --msgid-bugs-address="cb@iutbeziers.fr" \
    --package-name="cb-login" \
    --package-version="1.0" \
    -o cb-login.pot gtk3-06-internationalisation.c
```

C'est à partir de ce fichier POT généré par l'outil que l'on va décliner les différents fichiers de traduction. Mais avant tout, on peut compléter les valeurs créées en majuscules dans l'entête du fichier POT pour décrire plus précisément le projet.

Ensuite, pour le français par exemple, il suffit de compléter les valeurs **msgstr** (lignes 22, 26 et 30) des mots à traduire et d'enregistrer cela dans le fichier **fr.po**. On répète l'opération pour chaque langage : **en.po**, **de.po**, **es.po**, etc.

EN COUVERTURE
Ce document est la propriété exclusive de Johann Locatelli(johann.locatelli@businessdecision.com)

```
...
07: msgid ""
08: msgstr ""
09: "Project-Id-Version: cb-login 1.0\n"
10: "Report-Msgid-Bugs-To: cb@iutbeziers.fr\n"
...
15: "Language: fr_FR\n"
16: "MIME-Version: 1.0\n"
17: "Content-Type: text/plain; charset=UTF-8\n"
18: "Content-Transfer-Encoding: 8bit\n"
19:
20: #: gtk3-06-internationalisation.c:74
21: msgid "Authentication"
22: msgstr "Authentification"
23:
...
28: #: gtk3-06-internationalisation.c:57 gtk3-06-
internationalisation.c:62
29: msgid "UserName"
30: msgstr "Nom d'utilisateur"
```

Enfin, on fabrique la version binaire (**fr.mo**) avec **msgfmt** et on la copie dans le répertoire des locales du système, en utilisant le nom du programme défini à la ligne 4 du code source.

```
$ msgfmt fr.po -o fr.mo
$ sudo cp fr.mo /usr/share/locale/fr/LC_MESSAGES/cb-login.mo
```

Il ne reste qu'a tester cela en lançant le programme soit directement (avec la langue par défaut du système) soit en précisant celle-ci dans la variable système **LC_ALL** (voir figure 5).

```
LC_ALL=nl_NL ./gtk3-06-internationalisation
```

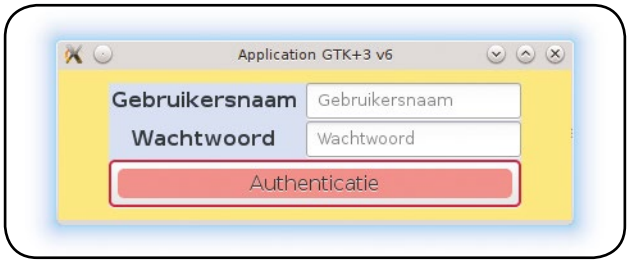


Fig. 5 : Version néerlandaise de l'application.

Conclusion

Voila, vous devez pouvoir maintenant commencer à réaliser vous-même de petits programmes graphiques interactifs. Mais, si vous patientez un peu, je vous expliquerai comment améliorer vos applications en ajoutant des menus ou comment accélérer le développement avec **GtkBuilder** et l'outil **Glade**. ■

Références

[1] Site officiel de GTK+ : http://www.gtk.org/

[2] Site de documentation en ligne de GTK+ : https://developer.gnome.org/gtk3/stable/

[3] Liste des différents conteneurs en GTK+3 : https://developer.gnome.org/gtk3/stable/GtkContainer.html#GtkContainer.object-hierarchy

[4] Signaux disponibles pour un GtkButton : https://developer.gnome.org/gtk3/stable/GtkButton.html#GtkButton.signals

[5] Sélecteurs de styles CSS en GTK+3 : https://developer.gnome.org/gtk3/stable/GtkCssProvider.html#gtkcssprovider-selectors

[6] Librairie d'internationalisation de GLib : https://developer.gnome.org/glib/stable/glib-I18N.html

Ce document est la propriété exclusive de Johann Locatelli(johann.locatelli@businessdecision.com)

UN VRAI LANGAGE

Guillaume SAUPIN [Responsable R&D QosEnergy, Data Scientist]

Les deux précédents articles de cette série nous ont permis de mettre au point les fonctionnalités de base de notre langage de programmation. Nous sommes maintenant outillés pour pouvoir parser et évaluer des expressions d'un langage de type Lisp. De plus, nous disposons d'un mécanisme de plugins permettant d'étendre facilement notre langage. Nous allons maintenant passer aux choses sérieuses et permettre à notre langage de s'étendre en déclarant des fonctions, et surtout en supportant les macros.

Mots-clés : C++, Lisp, LLVM, Compilation, Parseur, Langage

Résumé

Dans ce troisième article, nous allons étendre très largement les possibilités de notre langage, en ajoutant non seulement les fonctions, mais aussi les macros. Pour cela, nous commencerons par ajouter quelques structures pour le contrôle de flux, comme l'indispensable `if`. Nous détaillerons ensuite la forme spéciale `let`, qui permet d'ajouter des variables. Il apparaîtra ensuite que l'ajout de fonctions est finalement proche de cette forme. Enfin, nous irons au-delà de ce que sont capables de faire la plupart des langages en dotant notre langage de macros.

Nous avons conclu notre précédent article en dotant notre langage de programmation d'un ensemble de fonctionnalités, nommé *core*, et permettant entre autres de supporter les opérations arithmétiques de base.

Nous allons à présent ajouter un *plugin* pour définir de nouvelles fonctions.

Mais avant ça, nous allons étendre encore plus notre langage, notamment avec des structures impératives telles que le `if`.

1 L'essentiel

Il nous manque au moins deux fonctionnalités indispensables à tout langage de programmation. La première est le contrôle de flux, avec des expressions de type `if`, `then`, etc.

La seconde, c'est la possibilité d'ajouter des variables. En effet, nous avons décrit un objet `Environment` qui assure cette fonctionnalité, mais nous n'avons rien ajouté dans le code pour en bénéficier.

1.1 Contrôle de flux

En **Lisp**, il existe plusieurs formes spéciales permettant de contrôler le flux d'exécution d'un programme. Nous allons nous concentrer sur les plus utiles.

1.1.1 Le standard : if

Commençons par le `if`, dont voici un exemple :

```
(if (plusp a)
  a
  -a)
```

Ce document est la propriété exclusive de Johann Locatelli(johann.locatelli@businessdecision.com)

La forme spéciale **if** prend trois arguments. Le premier est le test : si le résultat de ce test est positif, alors, le second argument est exécuté. S'il est négatif, c'est le troisième et dernier argument qui est exécuté. Cet exemple retourne donc la valeur absolue de **a**.

Cette forme fonctionne différemment des formes classiques où le premier terme est une fonction. En effet, dans ces cas-là, tous les arguments sont évalués avant que la fonction ne s'applique sur eux. C'est ce qui justifie que ce type de forme est nommée **spéciale**.

Le code qui permet ce fonctionnement se trouve dans le module spécial, que l'on charge avec un :

```
(load "./special.so" "registerSpecialHandlers")
```

Il est codé comme suit :

```
std::shared_ptr<Atom> iff = SymbolAtom::New(env, "if");
```

On crée le symbole **if**, et on lui associe une fonction :

```
iff->closure = [](Sexp* sexp, Cell::CellEnv& env) {
  std::shared_ptr<Cell> test = sexp->cells[1];
  std::shared_ptr<Cell> True = sexp->cells[2];
  std::shared_ptr<Cell> False = sexp->cells[3];
  if(test->eval(env)->real)
    return True->eval(env);
  else
    return False->eval(env);
};
```

Les trois arguments sont récupérés, puis suivant le résultat du test, le second ou le troisième terme sont évalués dans l'environnement. À noter que la valeur *faux*, est encodée par un réel nul, et la valeur *vrai* par tout autre réel non nul.

1.1.2 Plus lispien : le cond

Dans le même ordre d'idée, on trouve le **cond**, que je trouve particulièrement pratique, et dont voici un exemple :

```
(cond
  (nil
   (print "trop nil"))
  ((eq 12 13)
   (print "coucou"))
  ((eq 12 12)
   (progn
     (print "cond succeed"))))
```

Ce contrôleur de flux évalue successivement plusieurs tests, et dans le cas d'un résultat positif, exécute le code qui suit et retourne. Ici, ce fragment de code affiche en conséquence *« cond succeed »*.

Dans notre module spécial, cela donne :

```
  std::shared_ptr<Atom> cond = SymbolAtom::New(env, "cond");
  cond->closure = [](Sexp* sexp, Cell::CellEnv& env) {
    for(auto condIt = sexp->cells.begin() + 1 ; condIt != sexp-
>cells.end() ; condIt++)
      {
        std::shared_ptr<Sexp> cond = std::dynamic_pointer_
cast<Sexp>(*condIt);
        std::shared_ptr<Cell> test = cond->cells[0];
        if(test->eval(env)->real) {
          std::shared_ptr<Cell> res;
          for(auto codeIt = cond->cells.begin() + 1 ; codeIt !=
cond->cells.end() ; codeIt++)
            res = (*codeIt)->eval(env);
          return res;
        }
      }
    return Cell::nil;
  };
```

Avec la même facilité, on peut définir les autres **when**, **unless**, **switch** dont doit disposer tout langage.

1.2 Déclaration de variables

Autre point tout aussi indispensable à tout langage de programmation, la capacité à déclarer des variables est à peine plus compliquée à mettre en œuvre, maintenant que nous disposons d'un environnement.

1.2.1 Le let

En lisp, cette association entre un identifiant et une valeur se fait avec la structure suivante :

```
(let ((a 2)
      (b 3))
  (+ a b))
```

Ceci déclare deux variables, **a** et **b**, auxquelles on associe respectivement les valeurs **2** et **3**. Ces variables peuvent alors être réutilisées dans le corps du **let**, ici **(+ a b)** qui retourne **5**.

De manière générale, la syntaxe du **let** est :

```
(let declaration-list
  body)
```

Ce document est la propriété exclusive de Johann Locatelli(johann.locatelli@businessdecision.com)

La liste de déclaration contient des listes de paires (symbole/valeur), qui associe la valeur au symbole. *body* contient pour sa part le corps du code sous forme d'une ou plusieurs *Sexps*.

Là encore, il s'agit d'une forme spéciale qui ne répond pas à la sémantique générale (fonction puis liste d'arguments).

```
  std::shared_ptr<Atom> let = SymbolAtom::New(env, "let");
  let->closure = [](Sexp* sexp, Cell::CellEnv& env) {
    std::shared_ptr<Sexp> vars = std::dynamic_pointer_
cast<Sexp>(sexp->cells[1]);

    std::map<std::string, std::shared_ptr<Cell> > newEnv;
    for(int vId = 0 ; vId < vars->cells.size() ; vId++) {
      std::shared_ptr<Sexp> binding = std::dynamic_pointer_
cast<Sexp>(vars->cells[vId]);
      std::shared_ptr<Atom> label = std::dynamic_pointer_
cast<Atom>(binding->cells[0]);
      std::shared_ptr<Cell> value = std::dynamic_pointer_
cast<Cell>(binding->cells[1]);

      newEnv[label->val].reset();

      const std::shared_ptr<Cell> res = value->eval(env);
      newEnv[label->val] = res;
    }

    env.addEnvMap(&newEnv);
    std::shared_ptr<Cell> res;
    for(auto bodyIt = sexp->cells.begin() + 2 ; bodyIt !=
sexp->cells.end() ; bodyIt++) {
      res = (*bodyIt)->eval(env);
    }
    env.removeEnv();
    if (res)
      return res;
    else
      return Cell::nil;
  };
```

Comme dans les exemples précédents, nous passons par la déclaration d'un nouveau symbole **let**, auquel nous associons la *closure* qui sera appelée lors de l'évaluation.

Cette *closure* itère ensuite sur la liste de déclarations, évalue la partie valeur, et associe le symbole à cette valeur dans notre nouvelle carte. Cette nouvelle carte est ensuite ajoutée à l'environnement courant, puis les *Sexps* du *body* sont évaluées tour à tour. Comme le veut la convention en lisp, le résultat de la dernière expression évaluée est retourné.

Notez la présence du **newEnv[label->val].reset();** qui libère la **Cell** éventuellement déjà associée avec un symbole, de manière à éviter une fuite mémoire.

1.2.2 Le let*

Il existe une autre forme spéciale, utilisée assez fréquemment, et dont le fonctionnement est très proche du **let**. Il s'agit du **let***, qui se distingue du **let** par le fait qu'il est possible au sein de la déclaration des variables de réutiliser des variables définies préalablement dans la même liste. En prenant un exemple similaire au précédent, voici comment elle fonctionne :

```
(let* ((a 2)
       (b (/ (* a 3) 2)))
  (+ a b))
```

La variable **a** est réutilisée pour calculer la valeur de **b**.

Avec les outils dont nous disposons, nous pouvons aussi assurer cette fonctionnalité avec une poignée de lignes de code :

```
  std::shared_ptr<Atom> letStart = SymbolAtom::New(env, "let*");
  letStart->closure = [](Sexp* sexp, Cell::CellEnv& env) {
    std::shared_ptr<Sexp> vars = std::dynamic_pointer_
cast<Sexp>(sexp->cells[1]);

    std::map<std::string, std::shared_ptr<Cell> > newEnv;
    env.addEnvMap(&newEnv);
    for(int vId = 0 ; vId < vars->cells.size() ; vId++) {
      std::shared_ptr<Sexp> binding = std::dynamic_pointer_
cast<Sexp>(vars->cells[vId]);
      std::shared_ptr<Atom> label = std::dynamic_pointer_
cast<Atom>(binding->cells[0]);
      std::shared_ptr<Cell> value = std::dynamic_pointer_
cast<Cell>(binding->cells[1]);

      newEnv[label->val].reset();

      const std::shared_ptr<Cell> res = value->eval(env);
      newEnv[label->val] = res;
    }

    std::shared_ptr<Cell> res;
    for(auto bodyIt = sexp->cells.begin() + 2 ; bodyIt !=
sexp->cells.end() ; bodyIt++) {
      res = (*bodyIt)->eval(env);
    }
    env.removeEnv();
    if (res)
      return res;
    else
      return Cell::nil;
  };
```

Il s'agit quasi du même code que celui du **let**. Seul l'ajout de la nouvelle carte à l'environnement a été fait un peu plus tôt dans le code, avant le traitement des déclarations.

Ce document est la propriété exclusive de Johann Locatelli(johann.locatelli@businessdecision.com)

2 Plus de fonctionnel

Notre langage supporte désormais la forme spéciale **let**. Cette forme, qui associe des variables à leurs valeurs et qui exécute ensuite un ensemble d'expressions est en fait très proche de l'évaluation d'une fonction.

2.1 Ajouter une fonction

Lors de l'appel d'une fonction, on associe des valeurs à des variables, et on exécute un ensemble d'expressions : le corps de la fonction. Seul diffère donc le moment où valeurs et variables sont associées.

Rappelons qu'en lisp, l'ajout de fonctions se fait généralement comme suit :

```
(defun my-add (a b)
  (+ a b))
(my-add 5 (+ 4 5));;appel de la fonction my-add
```

En regardant de plus près la déclaration de fonction ci-dessus, on voit qu'elle suit la forme générique suivante :

```
(defun name args-list
  body)
```

Où *name* est le nom de la fonction, *args-list* est la liste de ses paramètres, et *body* est un ensemble d'expressions formant son corps.

Dans le cas précédent, le nom était **my-add**, la liste des arguments **(a b)**, et le corps de la fonction **(+ a b)**.

L'exécution de cette fonction revient alors simplement à évaluer les arguments **a** et **b**, et à utiliser ces valeurs en lieu et place de **a** et **b** dans le corps de la fonction. Soit en lisp, **(my-add 5 (+ 4 5))** est l'équivalent de :

```
(let ((a 5)
      (b (+ 4 5)))
 (+ a b))
```

Le parallèle entre l'appel d'une fonction et la forme spéciale **let** apparaît très clairement.

Toujours dans notre même esprit de modularité, l'ensemble des fonctionnalités propres aux fonctions va être ajouté dans un module, que nous chargerons comme suit :

```
(load "./functional.so" "registerFunctionalHandlers")
```

Le code permettant l'ajout d'une fonction est très proche dans l'esprit de celui du **let** :

```
  std::shared_ptr<Atom> defun = SymbolAtom::New(env, "defun");
//handle recursion
  defun->closure = [](Sexp* sexp, Cell::CellEnv& env) {
    std::shared_ptr<Atom> fname = SymbolAtom::New(env,
sexp->cells[1]->val);
    std::shared_ptr<Sexp> args = std::dynamic_pointer_
cast<Sexp>(sexp->cells[2]);
    std::shared_ptr<Sexp> body = std::dynamic_pointer_
cast<Sexp>(sexp->cells[3]);
    std::dynamic_pointer_cast<SymbolAtom>(fname)->code = body;
    std::dynamic_pointer_cast<SymbolAtom>(fname)->args = args;
    if(args && body)
      {
  fname->closure = [env, args, body](Sexp* self, Cell::CellEnv&
callingEnv) mutable {
    if(&env != &callingEnv)
      for(auto eIt = callingEnv.envs.begin() ; eIt !=
callingEnv.envs.end() ; ++eIt)
        {
    env.addEnvMap(*eIt);
        }

    std::map<std::string, std::shared_ptr<Cell> > newEnv;
    for(int c = 0 ; c < args->cells.size() ; c++)
      {
        std::shared_ptr<Cell> val = self->cells[c+1]->eval(env);
        std::shared_ptr<SymbolAtom> symb = std::dynamic_pointer_
cast<SymbolAtom>(val);
        if(symb && env.find(symb->val) != env.end())
    newEnv[args->cells[c]->val] = env[symb->val];
        else
    newEnv[args->cells[c]->val] = val; // Eval args before
adding them to env (avoid infinite loop when defining recursive
function)
      }
          env.addEnvMap(&newEnv);
    std::shared_ptr<Cell> res = body->eval(env);
    //The following lines are useless :
    env.removeEnv();
    if(&env != &callingEnv)
      for(auto eIt = callingEnv.envs.begin() ; eIt !=
callingEnv.envs.end() ; eIt++)
        env.removeEnv();
    return res;
  };

      }

    return fname;
  };
```

Ce document est la propriété exclusive de Johann Locatelli(johann.locatelli@businessdecision.com)

Nous récupérons dans un premier temps les trois informations présidant à la déclaration d'une fonction : son nom, ses arguments, puis son corps. Le nom de la fonction est utilisé pour créer un symbole qui sera accessible dans l'environnement et qui pointera vers la fonction.

Ensuite, nous utilisons une fonctionnalité très intéressante du **C++11**, qui nous permet de supporter les fermetures dans nos fonctions. C'est-à-dire que nous allons capturer les variables de l'environnement existant lors de la création de la fonction.

Pour ce faire, lorsque nous créons la *closure* **`fname->closure`**, vous avez remarqué que nous passons **`[env, ...]`**. Cela signifie que ces variables sont capturées (par copie), et quelles seront du coup toujours existantes et utilisables au sein de la *closure*, même si nous sommes à ce moment sortis de la portée dans laquelle elles existent normalement.

Ainsi, nous pourrons réutiliser lors de l'appel de la fonction l'environnement **`env`** qui existait lors de sa création. Toutes les variables présentes dans cet environnement peuvent donc être réutilisées. C'est ce que nous faisons, en réutilisant cet environnement et en lui ajoutant les cartes de l'environnement existant lors de l'appel de la fonction : **`callingEnv`**.

Le reste du code est similaire à celui du **`let`**. Il se contente d'évaluer les arguments et de les associer à leurs variables, puis d'évaluer le corps de la fonction.

Une fois le corps évalué, l'environnement de l'appel est retiré de l'environnement de la création.

2.2 LOL

En une quarantaine de lignes de code, nous avons doté notre langage du support des *Let Over Lambda*, qui sont si pratiques. Nous sommes désormais en mesure d'exécuter correctement le petit exemple de code que je vous avais présenté dans le premier article. Il montre succinctement comment on peut se doter d'un mécanisme proche de l'orienté objet :

```
(let ((counter 0))
  (defun inc-counter ()
    (incf counter))
  (defun get-counter ()
     counter)
  (defunc dec-counter ()
    (decf counter))
```

Ici, la fermeture porte sur la variable **`counter`**, qui est capturée par les fonctions **`inc-counter`**, **`dec-counter`** et **`get-counter`**.

2.3 Fonctions anonymes

Avec la même simplicité, il est possible de définir des fonctions anonymes. Elles se déclarent avec le mot clef **`lambda`**, et s'utilisent comme suit :

```
(funcall (lambda (a b) (+ a b)) 2 3)
```

Je ne rappelle pas leur code C++ ici, car il est quasi exactement le même que celui pour les fonctions classiques, hormis le fait qu'elles ne sont pas associées à un symbole portant leur nom, puisqu'elles sont anonymes. **`funcall`**, pour sa part, est codé ainsi :

```
  std::shared_ptr<Atom> funcall = SymbolAtom::New(env, "funcall");
  funcall->closure = [](Sexp* sexp, Cell::CellEnv& env)  {
    std::shared_ptr<Cell> lambda = std::dynamic_pointer_cast<Cell>(sexp->cells[1])-
>eval(env);
    std::shared_ptr<Sexp> args = Sexp::New();
    for(auto aIt = sexp->cells.begin()+1 ; aIt != sexp->cells.end() ; aIt++)
      args->cells.push_back(*aIt);

    return lambda->closure(args.get(), env);
  };
```

Il se contente de récupérer la fonction anonyme, de regrouper les arguments dans une même *Sexp*, et d'invoquer la fonction.

2.4 Macro

Voici maintenant venu le temps d'attaquer la substantifique moelle du Lisp : les macros. Pour rappel, il s'agit ni plus ni moins que de pouvoir générer du Lisp depuis du Lisp en utilisant toutes les fonctionnalités du Lisp !

Après avoir implémenté cette partie, nous serons devenus des programmeurs qui programment des programmes qui écrivent des programmes ;)

Ce document est la propriété exclusive de Johann Locatelli(johann.locatelli@businessdecision.com)

2.4.1 Manipuler du code en Lisp

De quelle façon peut-on manipuler du code en Lisp ? Dans la plupart des langages, il n'existe généralement aucun moyen de générer ou modifier du code de manière simple, et en bénéficiant de toutes les possibilités du langage.

Au mieux, on dispose parfois d'une extension du langage, comme les *templates* en C++. Ou alors, on dispose d'une fonction **eval**, comme en **Ruby**, et il faut alors se résigner à utiliser la panoplie des outils de manipulation des chaînes de caractères pour arriver à nos fins.

En Lisp, nous l'avons vu, le code est représenté à l'aide de listes, or le Lisp est justement un langage qui excelle dans la manipulation des listes. Donc pour pouvoir manipuler du code, il faut juste se débrouiller pour qu'il ne soit pas directement évalué, comme c'est le cas lors de l'invocation d'une fonction.

Il existe justement une forme spéciale, **quote**, qui permet de retourner un bout de code tel quel, sans l'évaluer :

```
> (quote (+ (* 3 4) 12))
  (+ (* 3 4) 12) ;;; on retourne le paramètre de quote intact
> '(+ (* 3 4) 12)
  (+ (* 3 4) 12) ;;; idem en utilisant le simple quote ' comme
raccourci syntaxique
```

Cette fonctionnalité peut sembler un peu inutile à première vue, mais si l'on creuse un peu, on voit qu'il est possible de modifier un bout de code grâce à ce simple mécanisme :

```
> (mapcar (lambda (x)
     (cond ((eq x '+) '-)
           (t x)))
  '(+ (* 3 4) 12)))
(- (* 3 4) 12) ;;; le plus est remplacé par un moins
```

La fonction **mapcar** itère sur tous les éléments de la liste qui lui est passée, en l'occurrence notre code, et remplace les **+** par des **-**.

Voilà déjà un premier pas vers la modification de code. Ce qui serait intéressant de pouvoir faire, c'est de pouvoir réutiliser simplement des variables déjà existantes dans du code.

Par exemple, dans une optique de débogage, il peut être intéressant de générer à partir de code existant un bout de code affichant le code et le résultat de son évaluation. Il va donc falloir injecter ce code dans un autre bout de code. C'est ce que permet le *backquote* ou **`**. Cette forme spéciale fonctionne de la manière suivante : elle retourne tel quel le code quelle précède, à l'exception des expressions précédées par une virgule, qui sont évaluées et puis insérées.

```
> (let ((mon-code '(+ 2 3)))
       `(format t "~a~a~a" (quote ,mon-code) " evaluates to "
,mon-code))
(format t "~a~a~a" '(+ 2 3) " evaluates to " (+ 2 3)) ;;; mon
code a été injecté
```

Ainsi, dans le bout de code ci-dessus, **mon-code** est remplacé par sa valeur **(+ 2 3)**.

La fonction **format** affiche sur la sortie standard les paramètres qui lui sont passés. Dans notre cas, si nous exécutions le code que nous venons de générer, nous obtiendrions :

```
> (format t "~a~a~a" '(+ 2 3) " evaluates to " (+ 2 3))
(+ 2 3) evaluates to 5
```

Nous venons de découvrir les outils de base pour manipuler le code. Nous allons maintenant voir comment leur utilisation est facilitée par les macros et, surtout, nous allons voir avec quelle facilité nous pouvons étendre notre langage avec cette fonctionnalité si puissante.

2.4.2 Qu'est-ce qu'une macro ?

Une macro est une forme spéciale dont les arguments ne sont pas évalués. Dit autrement, et au regard de ce que nous venons de voir, c'est comme si les arguments passés à une macro étaient automatiquement quotés.

Ensuite, ces arguments peuvent être réutilisés dans le corps de la macro, généralement au sein d'une *backquote*, afin de générer du code. En généralisant notre exemple d'aide au débogage, on obtient :

```
> (defmacro with-debug (code)
  (let ((res code))
    `(format t "~a evaluates to ~a" (quote ,code) ,res) ))
```

Cette macro s'utilise alors comme suit :

```
> (with-debug (* 2 5))
(* 2 5) evaluates to 10
```

Ainsi, lors de l'invocation de cette macro, le code suivant a été généré puis exécuté :

```
(format t "~a evaluates to ~a" '(* 2 5) (* 2 5))
```

Ce document est la propriété exclusive de Johann Locatelli(johann.locatelli@businessdecision.com)

On peut s'en convaincre en invoquant **macroexpand**, qui génère le code de la macro sans l'exécuter :

```
> (macroexpand '(with-debug (* 2 5)))
(format t "~a evaluates to ~a" '(* 2 5) (* 2 5))
```

2.4.3 Implémentation

Comme pour les autres fonctionnalités de notre langage, l'implémentation des macros se fait en moins de cinquante lignes :

```
  std::shared_ptr<Atom> defmacro = SymbolAtom::New(env,
"defmacro");
  defmacro->closure = [](Sexp* sexp, Cell::CellEnv& env) {
    std::shared_ptr<Atom> fname = SymbolAtom::New(env,
sexp->cells[1]->val);
    std::shared_ptr<Sexp> args = std::dynamic_pointer_
cast<Sexp>(sexp->cells[2]);
    std::shared_ptr<Cell> body = sexp->cells[3];

    if(args && body)
      {
  fname->closure = [env, args, body, fname](Sexp* self,
Cell::CellEnv& callingEnv) mutable {
          if(&env != &callingEnv)
     for(auto eIt = callingEnv.envs.begin() ; eIt != callingEnv.
envs.end() ; ++eIt)
       {
   env.addEnvMap(*eIt);
       }
    std::vector<std::shared_ptr<Cell> > cls(self->cells.
begin()+1, self->cells.end());
    std::stringstream ss;

    std::map<std::string, std::shared_ptr<Cell> > newEnv;
    for(int c = 0 ; c < args->cells.size() ; c++)
      {
        newEnv[args->cells[c]->val] = self->cells[c+1];
//don't eval as we are in a macro !
      }
    env.addEnvMap(&newEnv);
          std::shared_ptr<Cell> copy = body->duplicate();
          std::shared_ptr<Cell> expansion = copy->eval(env);
//perform macro expansion
    //replace macro code by expansion
    *self = *dynamic_cast<Sexp*>(expansion.get());
    std::shared_ptr<Cell> res = expansion->duplicate()-
>eval(env); //eval code
    if(&env != &callingEnv)
      for(auto eIt = callingEnv.envs.begin() ; eIt != callingEnv.
envs.end() ; eIt++)
             env.removeEnv();

    return res;
  };
      }
    env.func[fname->val] = fname;
    return fname;
  };
```

On retrouve les mêmes principes que pour l'ajout de fonction, à savoir la fusion de l'environnement de déclaration et celui d'appel, ainsi que l'enregistrement des arguments dans ce nouvel environnement. Par contre, les arguments ne sont pas évalués, afin de manipuler du code et non le résultat de son évaluation.

Le corps de la macro est ensuite dupliqué, et évalué, ce qui génère le nouveau code. C'est la phase d'expansion de la macro. On remplace ensuite l'ancien code, à savoir l'appel à la macro par son expansion. Et on finit par nettoyer l'environnement.

C'est aussi simple que ça !

Conclusion

Avec cet article, et quelques centaines de lignes de code, nous avons pu doter notre langage du support pour le contrôle de flux, la déclaration de variables, les fonctions, et surtout les macros.

Ce qui j'espère vous a frappé dans cette présentation, c'est la simplicité avec laquelle sont introduites des fonctionnalités aussi puissantes que les macros. Cette simplicité découle de la façon dont est représenté en interne le code lisp : sous forme de liste. C'est la fameuse homoiconicité du Lisp !

Nous avons désormais un langage plutôt complet, et facilement évolutif. Reste à s'attaquer à ses performances. C'est ce que nous verrons dans le prochain et dernier article, qui ajoutera à notre langage la compilation. ■

Note

Le code de cet article a été développé sous Ubuntu, et est disponible sur GitHub : https://github.com/kayhman/llisp.

Pour en savoir plus...

Je recommande ardemment la lecture du livre « *Let Over Lambda* » de Doug Hoyte, dont on peut trouver les premiers chapitres en ligne : http://letoverlambda.com/.

Une approche moins brutale peut être trouvée dans le « *Practical common Lisp* » de Peter Sebeil, ou dans le « *On Lisp* » de Paul Graham, dont les versions électroniques sont librement disponibles en ligne.

Ce document est la propriété exclusive de Johann Locatelli(johann.locatelli@businessdecision.com)

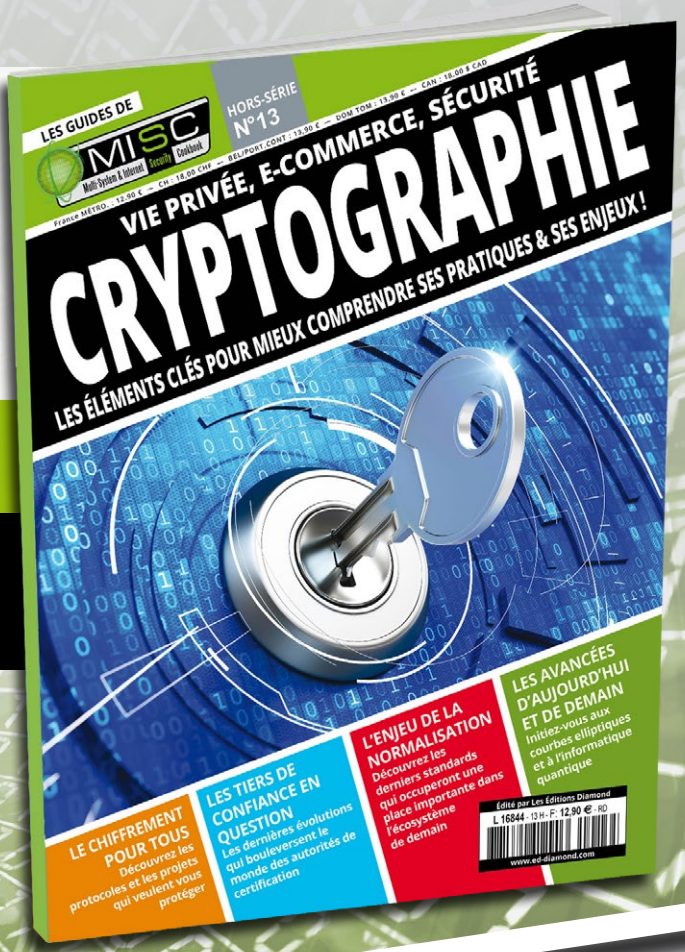
Ce document est exclusive de Johann Locatelli(johann.locatelli@businessdecision.com)

COMPRENEZ LES PRATIQUES ET LES ENJEUX ACTUELS DE LA CRYPTOGRAPHIE !

MISC HORS-SÉRIE N° 13

Toujours disponible sur :

www.ed-diamond.com



OPTIMISEZ LA PROTECTION DE VOTRE SYSTÈME LINUX ET DE VOS APPLICATIONS !

GNU/LINUX MAGAZINE HORS-SÉRIE N° 76

Toujours disponible sur :

www.ed-diamond.com

(RE)DÉCOUVRONS JQUERY

Laurent NAVARRO [Développeur @Toulouse]

jQuery est une bibliothèque connue de tous ceux qui développent sur le Web, parfois juste de nom, parfois juste parce qu'ils ont copié/collé quelques lignes magiques, je vous invite aujourd'hui à l'étudier d'un peu plus près.

Mots-clés : jQuery, HTML, CSS, JavaScript, Web, Programmation

Résumé

L'objectif de cet article est de reprendre les bases de jQuery afin de formaliser un peu son utilisation et d'essayer d'aller un peu plus loin que les utilisations ultra-basiques.

JQuery est une bibliothèque JavaScript de manipulation du DOM (*Document Object Model*) open source supportée par tous les navigateurs et qui vous aidera à faire du code JavaScript relativement puissant sans avoir à trop vous soucier des problèmes de compatibilité entre les navigateurs. Cette bibliothèque est très répandue puisque présente sur environ 70 % des sites web, c'est quasiment un standard de fait.

La devise de jQuery est *« Write less do more »*, et il faut bien reconnaître que dans la pratique, dès qu'on a un peu appris à s'en servir, c'est assez vrai.

jQuery est assez léger (84 Kio en version *minimifiée*), assez performant et la documentation est abondante **[1]**. Vous aurez le choix entre la V1 et la V2 qui sont fonctionnellement assez équivalentes ; la principale différence est que la V1 supporte IE 6, 7 et 8 et que la V2 est un peu plus performante (car moins rétrocompatible). Vous pouvez utiliser des serveurs CDN si votre site est sur Internet.

1 Les sélecteurs basiques

Le principe global de jQuery est de sélectionner des éléments (à l'aide de sélecteurs) et d'y appliquer des opérations dessus.

Les sélecteurs permettent de dire sur quels éléments de la page HTML nous allons travailler. L'invocation de jQuery avec un sélecteur nous retournera les éléments (`0`, `1` ou `n`) qui correspondent à notre sélecteur. Si aucun élément n'est sélectionné, cela ne provoquera pas les tristement célèbres erreurs JavaScript liées à un *id* absent.

Vos sélecteurs (ainsi que toutes les opérations avec jQuery d'ailleurs) peuvent être testés avec la console de développement ; ceci vous permettra de voir facilement le résultat de vos expressions.

Commençons par quelque chose de simple : nous allons colorier le texte « Hello World » en rouge. Voici la page html :

```
<!doctype html>
<html>
  <body>
    <script src="jquery-2.2.0.js"></script>
    <p id="demo1">Hello World</p>
    <script>
    $('#demo1').css('color', 'red');
    </script>
  </body>
</html>
```

Nous commençons par inclure la bibliothèque jQuery par :

```
<script src="jquery-2.2.0.js"></script>
```

Ce document est la propriété exclusive de Johann Locatelli(johann.locatelli@businessdecision.com)

Ensuite nous sélectionnons notre paragraphe, sur lequel nous avons mis l'identifiant **demo1**, avec le sélecteur **#demo1** et fixons son style **color** en rouge :

```
$('#demo1').css('color', 'red');
```

L'invocation de jQuery s'effectue à travers la fonction **$** qui prend comme paramètre un sélecteur et nous retourne un objet jQuery. Cet objet résultat, contient une référence vers tous les éléments sélectionnés et nous permet d'y appliquer des opérations dessus. Dans le cas présent, nous fixons le style **color** à rouge à l'aide de la méthode **css()**.

1.1 Syntaxe des sélecteurs

Les sélecteurs jQuery sont issus de la syntaxe des sélecteurs CSS et sont exprimés dans des chaînes de caractères passées à la fonction **$**.

Le préfixe **#** est utilisé pour sélectionner un élément par son *id* (exemple **#MonID**). Notez que les *id* sont sensibles à la casse.

Le préfixe « point » (**.**) permet de sélectionner des éléments par leur classe (attribut **class** des objets). Il est à noter que cette classe n'a pas forcement à être définie dans une css, ainsi il est fréquent d'ajouter une classe à des objets dans le seul but de pouvoir les récupérer plus tard dans jQuery. L'exemple ci-dessous va mettre en bleu les 2 paragraphes et mettre en gras le premier :

```
<p class="hello autreclasse">Hello World</p>
<p class="hello">Hello You</p>
<script>
    $('.hello').css('color', 'blue');
    $('.autreclasse').css('font-weight', 'bold');
</script>
```

Un sélecteur peut aussi spécifier un nom d'élément HTML **p**, **div**, **h1**, **input**, **body**, etc. (non sensible à la casse). Ceci permettra de sélectionner tous les éléments HTML correspondants. **body** est souvent utilisé pour ajouter du code à la fin du document.

```
$("body").append("Ajout à la fin");
```

L'utilisation des éléments HTML sera surtout intéressante en se limitant à une partie de l'arbre des objets. Cela pourra se faire en utilisant plusieurs sélecteurs avec une notion de hiérarchie ; par exemple pour prendre tous les **p** de la div **Div1**, on écrira dans ce cas tout simplement les deux sélecteurs séparés par un espace dans l'ordre de la hiérarchie **"#Div1 p"**. On applique donc le sélecteur **p** uniquement dans les hiérarchies résultantes du sélecteur précédent (**#Div1** dans notre exemple).

Une autre pratique fréquente est de faire un OU entre les sélecteurs, cela se fait tout simplement avec le séparateur **,** entre les sélecteurs. Exemple : **'.classe1,.classe2'** qui va sélectionner tous les objets ayant la classe **classe1** ET les objets ayant la classe **classe2**.

2 Opérations basiques

De nombreuses opérations sont disponibles sur les résultats des sélections ; nous allons regarder les plus fréquentes.

La méthode **html(contenu)** permet de fixer le contenu HTML de l'intérieur des éléments sélectionnés (la balise de l'élément sélectionné n'est pas modifiée, seul ce qui est entre les balises l'est). La méthode **text** est similaire, mais manipule du texte pur :

```
Mon contenu HTML : <span id="s1"></span> <br>
Mon contenu Texte : <span id="s2"></span> <br>
<script>
    $('#s1').html("<B>Du <i>HTML</i></B>");
    $('#s2').text("<B>Du <i>HTML</i></B>");
</script>
```

Cet exemple va donner visuellement le texte de la figure 1.

Mon contenu HTML : **Du *HTML***
Mon contenu Texte : <B>Du HTML</B>

Fig. 1 : Différence de rendu entre html() et text().

Vous noterez que l'utilisation de la méthode **text()** n'a pas interprété le code HTML, mais l'a inséré tel quel.

Ces méthodes existent aussi dans une version sans paramètre et, dans ce cas, elles retournent le contenu du premier élément retourné par le sélecteur pour **html** alors que **text**, elle, va agréger tous les textes (sans les balises HTML).

La méthode **val()** à un fonctionnement similaire (lecture/écriture), mais est utilisable pour manipuler les valeurs des champs d'un formulaire. En lecture, elle ne retourne que la valeur du premier champ alors qu'en écriture elle écrira tous les champs.

Ce document est la propriété exclusive de Johann Locatelli(johann.locatelli@businessdecision.com)

```
Mon input : <input type="text" size=10 id="i1" value="world">
<br>
Mon textarea : <textarea cols="20" rows="2" id="i2"></textarea>
<script>
    $('#i2').val("Hello "+$('#i1').val());
</script>
```

Ce code fixe la valeur de la textarea (**#i2**) en lisant celle du champ texte (**#i1**). Dans notre exemple, le résultat sera « Hello world ».

Les méthodes **attr()** et **prop()** permettent de manipuler les attributs et les propriétés suivant la même logique.

3 Les événements

Un autre aspect intéressant de jQuery est la facilité qu'il procure pour manipuler les événements. Il faut bien reconnaître que ce dernier point est parfois utilisé de façon quelque peu abusive (certains pensent que c'est le seul moyen de traiter un événement) :

```
<button id="bt1">Dis Bonjour</button>
<input type="text" id="i3">
<script>
$(document).ready(function() {
   $('#bt1').click(function(){
       $('#i3').val("Hello");
   });
});
</script>
```

Fig. 2 : Exemple de manipulation d'un champ.

Dans cet exemple (voir figure 2), nous utilisons deux événements. Le premier est l'événement **ready** du document qui est déclenché quand le DOM est prêt, cela n'inclut pas le fait que toutes les images soient chargées, c'est généralement un bon moment pour exécuter du code au chargement de la page.

La méthode **ready()** prend en paramètre la fonction à appeler lorsque l'événement aura lieu (on parle de *handler* ou de *callback*). Il peut y avoir autant de *handler* que vous voulez sur chacun des événements. Ici, nous avons utilisé une expression lambda, c'est-à-dire une fonction anonyme (le corps du *handler* est entre les parenthèses de l'appel à la méthode **ready()**). Nous aurions pu écrire ceci à la place :

```
function CallbackReady(){
    $('#bt1').click(function(){
        $('#i3').val("Hello");
    });
}
$(document).ready(CallbackReady);
```

L'utilisation d'expressions lambda est très répandue dans la communauté des utilisateurs jQuery, parfois de façon excessive là aussi. On voit parfois tout le code d'une application JavaScript de plusieurs dizaines de lignes entièrement écrit dans une expression lambda associée à l'événement **ready**.

Nous aurions aussi pu écrire **$(CallbackReady);**, car le passage d'un callback à la fonction **$** est un synonyme de la méthode **ready**.

Dans le *handler* de **ready**, nous associons dynamiquement à **bt1** un *handler* sur l'événement **click**.

Il faut noter qu'un cas aussi simple peut parfaitement être traité par un *callback* JavaScript classique :

```
<button id="bt3" onclick=" $('#i5').val('Hello');">Dis Bonjour
</button>
<input type="text" id="i5" size=10>
```

La méthode **click()** peut aussi être utilisée pour simuler l'événement clic sur un élément, dans ce cas il n'y a pas de paramètres :

```
$('#bt3').click();
```

Il est possible de capturer la plupart des événements communs depuis jQuery via des méthodes dédiées. Cependant, toutes ces méthodes ne sont que des spécialisations de la méthode **bind()** qui permet de s'associer à n'importe quel événement Javascript, nous aurions pu écrire le code suivant qui est équivalent :

```
  $('#bt1').bind('click', function(){
      $('#i3').val("Hello");
  });
```

Il y a des méthodes dédiées pour tous les événements souris, **click** et **dblclick** bien sûr, mais aussi tout ce qui est relatif au déplacement de la souris, à l'entrée de la souris dans une zone ou à la montée et descente des boutons

Ce document est la propriété exclusive de Johann Locatelli(johann.locatelli@businessdecision.com)

de la souris. Par défaut, il n'y a rien pour la molette, mais il y existe cependant de nombreuses solutions avec jQuery (bind, plugins, etc.).

Autour des formulaires et des champs qui les composent, on trouvera les événements **change** lorsqu'une valeur a été changée, les prises et perte de focus d'un champ et **submit** :

```
<input type="text" size=5 id="src1"> x 10
= <input type="text" size=5 id="dst1">
<script>
    $("#src1").change(function () {
        $('#dst1').
val(parseFloat($(this).val())*10);
    });
</script>
```

12 x 10 = 120

Fig. 3 : Exemple événement change.

Cet exemple (voir figure 3) va détecter le changement de valeur dans **src1** en attachant un *handler* via la méthode **change()**, récupérer la valeur, la convertir en flottant, la multiplier par **10** et la mettre dans **dst1**.

Vous remarquerez la technique utilisée pour récupérer la valeur des **src1** : j'utilise ici **$(this)** qui représente l'objet auquel est câblé l'événement (**src1** dans cet exemple) enveloppé dans un **$()** pour qu'il soit manipulable avec jQuery.

Une autre approche aurait été possible en utilisant l'objet de type **Event** qui est transmis en paramètre au *handler* :

```
<input type="text" size=5 id="src2"> x 10
= <input type="text" size=5 id="dst2">
<script>
$("#src2").change(function (evt) {
    $('#dst2').val(parseFloat($(evt.
target).val())*10);
});
</script>
```

Cet objet (nommé **evt** ici) contient plusieurs propriétés, dont la position de la souris, des informations sur le clavier et les objets impliqués. Ici nous utilisons **target** qui est l'objet cible de l'événement.

4 Promenade dans le voisinage

Maintenant que nous avons les notions de base, amusons-nous à faire des choses plus drôles. Je vous propose d'implémenter un mini panier qui ressemble à la figure 4.

Description	PU	Qté	Montant
Super ordinateur	643 €		
Arduino Uno	20 €		
		Total	

Fig. 4 : Mini panier.

Voilà le code, dans lequel j'ai volontairement mélangé les approches à des fins pédagogiques :

```
01 <!doctype html>
02 <html>
03 <head>
04   <meta charset="UTF-8">
05   <title>Mon super panier</title>
06   <style>
07     table.Grille {
08       border-collapse: collapse;
09       border: 1px solid black;    }
10     table.Grille td, table.Grille th {
11       border: 1px solid black;
12       padding: 2px 3px;    }
13   </style>
14 </head>
15 <body>
16   <script src="jquery-2.2.0.js"></script>
17   <table class="Grille" id="t1">
18     <tr>
19       <th>Description</th><th>PU</th>
20       <th>Qté</th>    <th>Montant</th>
21     </tr>  <tr class="orderline">
22       <td>Super ordinateur</td>    <td>643</td>
23       <td><input type="text" size=3></td>
24       <td class="totcol"></td>
25     </tr>  <tr class="orderline">
26       <td>Arduino Uno</td>    <td>20</td>
27       <td><input type="text" size=3></td>
28       <td class="totcol"></td>
29     </tr>  <tr>
30       <td colspan="3" style="text-align: right;font-weight: bold;">Total</td>
```

Ce document est la propriété exclusive de Johann Locatelli(johann.locatelli@businessdecision.com)

```
31      <td><input type="text" size=5 name="total" readonly></td>
32    </tr>
33  </table>
34  <script>
35  $(document).ready(function () {
36    $('#t1 .orderline input').change(function () {
37      var res = "";
38      var qte = parseFloat($(this).val());
39      if (!isNaN(qte))
40        res =qte*parseFloat($(this).closest('td').prev().text());
41      $(this).closest("tr").find("td:nth-child(4)").text(res);
42      // Calcul du total
43      res=0;
44      $('#t1 .totcol').each(function(){
45        var montant = parseFloat($(this).text());
46        if (!isNaN(montant))
47          res+=montant;
48      });
49      $("input[name=total]").val(res);
50    });
51  });
52  </script>
53 </body>
54 </html>
```

Vous l'aurez compris, le but est que l'utilisateur saisisse les quantités et que nous calculions les montants de chaque ligne et le montant total. L'idée est ici d'écrire du code qui puisse marcher pour un nombre de lignes variable.

Les lignes 1 à 16 sont le *header* HTML, l'intégration de jQuery et la définition de classes CSS pour avoir de belles bordures à mon tableau via la classe **`Grille`**.

De 17 à 33, vous trouverez le tableau avec le contenu, mais sans valeur dans les quantités.

Ligne 35, le *handler* sur **`ready`** pour mettre en place tout notre code.

Ligne 36, dans la table **`#t1`** pour toutes les lignes du corps de mon panier que j'ai marqué avec la classe **`.orderline`**, sur les **`input`**, je mets en place un *handler* sur l'événement **`change`**.

Dans l'esprit HTML5, j'aurais plutôt pu utiliser un sélecteur de toutes les lignes de la section **`tbody`** plutôt que marquer les lignes avec la classe **`orderline`**.

Les lignes 37 à 41 vont calculer la colonne *Montant* en multipliant le chiffre de la colonne *PU* par le champ *quantité*.

La variable **`qte`** est assez simplement récupérée en utilisant **`$(this).val()`** et en la convertissant en flottant.

Un petit **`isNan(qte)`** nous permet de ne faire notre calcul que si la **`qte`** est valide et de considérer une valeur vide sinon.

La ligne 40 mérite quelques explications. En partant de l'élément courant (le champ quantité) nous allons chercher son parent **`td`** le plus proche avec la méthode **`closest('td')`**. De là, nous allons récupérer l'élément précédent, qui est en l'occurrence la cellule **`td`** immédiatement à gauche (c'est-à-dire *PU*) avec la méthode **`prev()`** et nous allons récupérer son contenu avec la méthode **`text()`**. Une fois obtenu, nous le convertissons en flottant et le multiplions avec **`qte`**.

La ligne 41 utilise elle aussi quelques astuces. **`$(this).closest("tr")`** nous permet de remonter au niveau de la ligne courante, de là, nous allons chercher la quatrième cellule **`td`** (puisque le montant est dans la quatrième colonne) avec **`find("td:nth-child(4)")`** et y écrire la valeur avec **`text(res)`**.

Les lignes 42 à 49, permettent de calculer le total en bas à droite, c'est-à-dire la somme des montants de chaque ligne.

Pour identifier ma colonne montant, j'ai ici utilisé une technique plus conventionnelle et moins dépendante de la disposition graphique en marquant les cellules concernées avec la classe **`.totcol`** ; je me restreins cependant à celle de la grille **`#t1`**. Cette classe n'est pas déclarée dans les styles, mais j'aurais cependant pu en profiter pour lui mettre le style suivant :

```
.totcol{
  font-family:monospace;
  text-align:right;
}
.totcol:not(:empty)::after {content: ' Ð';}
```

Ceci afin de l'aligner à droite avec une fonte non proportionnelle et y ajouter un petit symbole euro à droite de tous les chiffres, mais pas des cellules vides. À noter que ce symbole n'est pas récupéré par la méthode **`text()`**, il est juste utilisé pour l'affichage.

Revenons sur les lignes 44 à 48, sur toutes les cellules **`.totcol`** nous itérons grâce à la méthode **`each()`** qui va appeler un *handler* (troisième niveau de profondeur ici) pour chacune des

Ce document est la propriété exclusive de Johann Locatelli(johann.locatelli@businessdecision.com)

cellules. Ce *handler* va récupérer le texte et le convertir en numérique (ligne 45). Si c'est une valeur valide (ligne 46), il va l'ajouter à la variable **res** (ligne 47) préalablement initialisée à **0**.

Une fois le total calculé, nous allons l'affecter à l'input ayant pour nom **total**, nous utilisons ici un sélecteur avec un filtrage par valeur d'attribut **$("input[name=total]")**. Attention à ne pas mettre d'espace avant le **[** sinon vous introduisez une relation de descendance qui ne va pas donner le résultat escompté !

5 Manipulation dynamique du DOM

Je vous propose maintenant d'ajouter des articles dans notre panier à l'aide de la liste de valeurs de la figure 5.

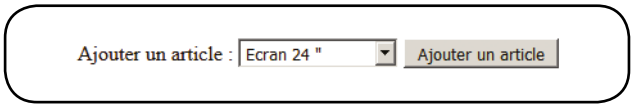


Fig. 5 : Liste de valeurs pour notre mini panier.

Le code est le suivant :

```
01: Ajouter un article : <select id="listearticle">
02:   <option></option>
03:   <option value="Art1" data-pu="643">Super ordinateur
</option>
04:   <option value="Art2" data-pu="20">Arduino Uno</option>
05:   <option value="Art3" data-pu="145">Ecran 24 "</option>
06: </select> <input type="button" onclick="AjoutArticle();"
value="Ajouter un article">
07: <script>
08: function AjoutArticle(){
09:   var SelOption=$('#listearticle option:selected');
10:   var newline=$("<tr class='orderline'><td></td><td></
td><td><input type='text' size=3></td><td class='totcol'></td>
</tr>");
11:   newline.find("td:first").text(SelOption.text()).next().
text(SelOption.data('pu'));
12:   $('.orderline:last').after(newline);
13: }
14: </script>
```

Les lignes 1 à 6 contiennent la liste des articles et le bouton sur lequel nous avons mis un appel à la fonction **AjoutArticle()** sur l'événement **onclick** (aucun intérêt de le faire avec jQuery ici).

La ligne 9 récupère l'objet HTML option qui est sélectionné. À noter que si nous avions juste voulu la valeur de l'option sélectionnée (**Art1** par exemple) nous aurions pu simplement écrire **$('#listearticle').val()**.

La ligne 10 fabrique un nouvel élément HTML correspondant à une ligne vide de notre tableau. En effet, la fonction **$** peut aussi prendre une chaîne HTML et dans ce cas fabrique un élément HTML qui n'est pas affiché.

La ligne 11 manipule cette ligne de tableau, afin d'y écrire le texte associé à l'option sélectionnée de la *listbox* **.text(SelOption.text())** dans la première colonne **newline.find("td:first")**. **find()** se comporte un peu comme **$** mais en partant d'un objet jQuery.

Les opérations qui ne retournent rien de particulier en jQuery, retournent l'objet qui les a invoquées, ainsi il est possible d'enchaîner les opérations. Ceci évite de réécrire et surtout de réexécuter un sélecteur, mais n'est pas toujours très simple à lire. Ainsi notre expression **.next().text(SelOption.data('pu'))**, va écrire le PU dans la seconde colonne.

Regardons un peu d'où sort le Prix Unitaire. Ce *PU* a été mis dans les options avec l'attribut **data-pu="20"**. Les valeurs des attributs de la forme **data-XXX** sont accessibles en lecture à travers la méthode **data('pu')**. La méthode **data(clé, valeur)** (avec 2 paramètres) permet de stocker dans l'objet n'importe quelle valeur JavaScript de façon typée.

Pour finir, la ligne 12 ajoute notre nouvelle ligne après la dernière ligne ayant la classe **orderline**.

Reste un dernier point : en l'état la mise à jour des montants et totaux vue au paragraphe précédent ne marchera pas sur les nouvelles lignes, car l'événement **onchange** n'est pas défini sur les nouveaux **input** de quantité. Nous pourrions l'ajouter lors de l'insertion, mais je vous propose une solution plus élégante en remplaçant la ligne 36 qui était :

```
$('#t1 .orderline input').change(function () {
```

Par une version plus dynamique (et performante) :

```
$('#t1').on('change','.orderline input',function () {
```

Cet appel à la méthode **on()** permet de dire que, sur ma table **#t1**, je capture tous les événements **change** et que si la cible de cet événement répond au filtre **'.orderline input'**, alors j'exécute le *callback*. Cette solution permet de fonctionner y compris si un élément est ajouté dans **#t1** après cette instruction, contrairement à **bind()** et ses variantes (**change**, **click**) qui s'appuient

Ce document est la propriété exclusive de Johann Locatelli(johann.locatelli@businessdecision.com)

sur les attributs **onchange**, **onclick**, etc. des éléments qui existent au moment où la méthode **bind()** est exécutée. La méthode **on()** permet aussi d'être plus performant, s'il y a de nombreux éléments, car ça évite de modifier le DOM pour chacun d'eux : avec **on()**, il y a un seul *handler* au niveau de l'élément sélectionné (**#t1** dans notre exemple).

Note

À noter, qu'il est possible d'ajouter du contenu avant ou après un élément, à l'intérieur ou à l'extérieur de l'élément.

Voici une petite illustration :

```
<div id="d1" style="border: 2px solid red; width: 300px; ">
    Mon contenu
</div>
<script>
    $('#d1').append("APPEND ICI").prepend("PREPEND ICI").
before("Before ICI").after("After ICI");
</script>
```

Qui nous donne le résultat de la figure 6.

Before ICI
PREPEND ICI Mon contenu APPEND ICI
After ICI

Fig. 6 : Différents types d'insertion.

6 Quelques opérations de plus

Étudions à présent quelques autres opérations assez communes.

Il est possible d'ajouter/supprimer des classes à des objets (ce qui va pouvoir impacter leur présentation ou pas) à l'aide des méthodes **addClass()**, **removeClass()**, **toggleClass()**, pour tester si une classe est présente, on utilisera **hasClass()**.

La méthode **css()** permet de lire/écrire les attributs de style.

Les méthodes **hide()**, **show()**, et **toggle()** permettent de masquer/rendre visible des éléments.

Les méthodes **toggle()** et **toggleClass()** permettent de basculer, rendre visible/invisible ou ajouter/supprimer une classe en fonction de l'état courant.

Je vous propose d'illustrer ces méthodes avec un petit exemple de bloc de textes expansibles.

▾ Mon super produit
Ici le détails de mon produit
sur plusieurs lignes.
▸ Mon autre produit

Fig. 7 : Exemple de textes escamotables.

Il s'agit d'avoir un texte précédé d'un triangle vers la droite (*caret right*) et quand on clique dessus, un texte associé apparaît et le triangle passe vers le bas (voir figure 7). Si on re-clique dessus ça masque le texte et remet le triangle vers la droite.

```
01: <link rel="stylesheet" href="https://maxcdn.bootstrapcdn.
com/font-awesome/4.5.0/css/font-awesome.min.css">
02: <div class="expandable">
03:     <i class="fa fa-fw fa-caret-right"></i> Mon super produit
04:     <div>Ici le détails de mon produit<br>
05:         sur plusieurs lignes. </div>
06: </div>
07: <div class="expandable">
08:     <i class="fa fa-fw fa-caret-right"></i> Mon autre produit
09:     <div>Ici le détails de mon autre produit</div>
10: </div>
11: <script>
12:     $('.expandable div').hide().css('margin-left','20px');
13:     $('.expandable .fa-caret-right').click(function(){
14:         $(this).next('div').toggle();
15:         $(this).toggleClass("fa-caret-right fa-caret-down")
16:     }).css('cursor','pointer');
17: </script>
```

La première ligne intègre, via un CDN ici, la très sympathique bibliothèque d'icône open source **Font Awesome [2]** qui est une bibliothèque d'icônes sous la forme d'une police vectorielle qui va nous être utile pour dessiner les triangles.

Ligne 2 à 10 se trouve le contenu : chaque bloc est dans un **div** marqué de la classe **expandable** et la partie masquable est dans la sous-div (j'aurais pu faire d'une autre façon, ce n'est qu'un exemple). Les tags **i** sont utilisés pour intégrer les icônes et la classe **fa-fw** permet que la largeur du triangle soit fixe (normalement le triangle horizontal est plus large que le vertical).

La ligne 12 masque toutes les sous-div avec **hide** et en profite pour leur mettre une petite marge à gauche à l'aide de la méthode **css()**. Nous aurions pu masquer ce contenu dans le code html avec ** **.

Je suis personnellement plus partisan de l'approche consistant à mettre en statique l'état initial du contenu, plutôt que

Ce document est la propriété exclusive de Johann Locatelli(johann.locatelli@businessdecision.com)

SYMPOSIUM
SUR LA SÉCURITÉ
DES TECHNOLOGIES
DE L'INFORMATION
ET DES COMMUNICATIONS

1 – 3
juin
2016
RENNES

SSTIC SYMPOSIUM SUR LA SÉCURITÉ DES TECHNOLOGIES DE L'INFORMATION ET DE LA COMMUNICATION
UNIVERSITÉ DE RENNES 1
CentraleSupélec
MINISTÈRE DE LA DÉFENSE
DGA
cea
AIRBUS GROUP
MISC Multi-System & Internet Security Cookbook

de faire l'ajustement des styles en Javascript au chargement quand ce n'est pas nécessaire, mais là c'était pour l'exemple.

La ligne 13 pose un *handler* sur le clic des *carets*.

Ligne 14, lors du clic, on rend visible ou invisible la **div** qui suit le *caret*.

Ligne 15, on *toggle* les classe **fa-caret-right** et **fa-caret-down**, ce qui a pour effet de passer du triangle droit au triangle bas, car quand j'enlève l'un, je mets l'autre (au début c'est un **fa-caret-right**).

Ligne 16, je profite d'avoir sélectionné le *caret* pour modifier son style et lui dire que le curseur doit montrer que cette zone est cliquable, bien sûr ceci aurait aussi pu être fait via une classe. Bien que ça soit cascadé à la méthode **click()**, ce code n'est exécuté qu'une seule fois.

6.1 Cases à cocher et boutons radio

Les cases à cocher et les boutons radio ont une petite spécificité, liée au fait qu'ils ont toujours une valeur et que ce qui va nous intéresser est de savoir s'ils sont cochés ou pas.

```
<input type="checkbox" value="Oui" name="MyCB1"> Oui <br>
<input type="radio" value="Oui" name="MyRadio1"> Oui <br>
<input type="radio" value="Non" name="MyRadio1"> Non <br>
```

Dans cet exemple, si vous exécutez **$('input[name=MyCB1]').val()** vous aurez toujours la valeur **Oui**, car en effet c'est bien la valeur de son attribut **value**. Le truc, c'est que dans un post classique, ce champ n'est posté que s'il est coché, c'est pour ça que ça marche bien d'habitude.

Pour savoir si c'est coché, il suffit de faire **$('input[name=MyCB1]').prop('checked')** qui retourne un booléen. Cette solution est relativement pratique pour les cases à cocher, mais n'est pas très adaptée pour les boutons radio. Pour ces derniers, il faudrait interroger chacun des boutons séparément.

Attention, en les adressant par **$('input[name=MyRadio1]').prop('checked')** on récupère l'état du premier bouton radio, ce qui n'est probablement pas ce qu'on souhaite faire.

La solution la plus pratique pour les boutons radio (qui marche aussi pour les cases à cocher) est :

```
$('input[name=MyRadio1]:checked').val()
```

Ici, grâce au sélecteur **:checked** on récupère la valeur associée au bouton radio qui est coché (Oui ou Non) ou **undefined** si aucun n'est coché. Si avoir **undefined** vous embête, vous pouvez utiliser le OU logique **||** pour forcer une chaîne vide ou une valeur par défaut :

```
$('input[name=MyRadio1]:checked').val()||""
```

6.2 Raccourcis clavier

Je ne résiste pas à vous montrer à quel point il est facile de capturer un raccourci clavier avec jQuery. Dans notre exemple précédent, si on souhaite que **<Ctrl>** + **<E>** fasse un *expand* de toutes les zones, il suffit d'écrire ceci :

```
$(document).bind('keydown', function(e) {
    if((e.keyCode === 69)&& (e.ctrlKey)) { // Ctrl +E
      $(".fa-caret-right").click();
      e.preventDefault();
      }
  });
```

On capture toutes les pressions de touches, et à chaque fois on vérifie si la touche **<Ctrl>** est enfoncée et si c'est la touche **<E>** (code ASCII **69**) qui est enfoncée. Si c'est le cas, on sélectionne tous les triangles non ouverts et on simule un clic dessus avec **$(".fa-caret-right").click();**, puis on bloque la propagation de la pression de la touche.

Conclusion

J'aurais encore pu écrire de nombreuses choses sur jQuery, des livres entiers (dont certains sont très bien) y sont consacrés. J'espère vous avoir montré qu'avec peu de code, on peut facilement faire des choses assez sympathiques sur une page web de façon dynamique.

Je vous encourage à jouer un peu avec cette bibliothèque très agréable, afin d'apprendre à la dompter, et de l'utiliser facilement lorsque vous en aurez besoin.

Dans un prochain article, j'aborderai la dimension AJAX que je n'ai pas du tout abordé dans cet article et qui est un autre pan important de jQuery. ■

Références

[1] Documentation de jQuery : http://api.jquery.com/

[2] Librairie Font-Awesome : https://fortawesome.github.io/Font-Awesome/

Ce document est la propriété exclusive de Johann Locatelli(johann.locatelli@businessdecision.com)

Ce document est la propriété exclusive de Johann Locatelli@businessdecision.com)

LE CLOUD GAULOIS, UNE RÉALITÉ ! VENEZ TESTER SA PUISSANCE

EXPRESS HOSTING

Cloud Public
Serveur Virtuel
Serveur Dédié
Nom de domaine
Hébergement Web

sales@ikoula.com
01 84 01 02 66
express.ikoula.com

ENTERPRISE SERVICES

Cloud Privé
Infogérance
PRA/PCA
Haute disponibilité
Datacenter

sales-ies@ikoula.com
01 78 76 35 58
ies.ikoula.com

EX10

Cloud Hybride
Exchange
Lync
Sharepoint
Plateforme Collaborative

sales@ex10.biz
01 84 01 02 53
www.ex10.biz

Je recherche

- [x] Paris
- [ ] Marseille
- [ ] Toulouse
- [ ] Lyon

24 offres trouvées

- [x] CDI

- [x] Consultant Technologique IAM
- [ ] Développeur web Full Stack (AngularJS / Node.JS)
- [ ] Lead développeur Java
- [ ] Open Source Gourou
- [ ] Ingénieur support produit
- [ ] Développeur Zend Framework 2
- [ ] Admin Devops
- [ ] Architecte système & réseaux
- [ ] Consultant technologique messagerie Open Source
- [ ] Administrateur système et réseaux Open Source
- [ ] Développeur Javascript Front-End
- [ ] Développeur Openstack

Voir toutes les offres

Je rejoins Linagora

JOBS. LINAGORA .COM

Ce document est la propriété exclusive de Johann Locatelli(johann.locatelli@businessdecision.com)

Ce document est la propriété exclusive de Johann Locatelli(johann.locatelli@businessdecision.com)