N°193 mai 2016

GNU LINUX MAGAZINE / FRANCE

DÉVELOPPEMENT ET ADMINISTRATION SUR SYSTÈMES OPEN SOURCE

HUMEUR / DEV

Optimisation ou lisibilité... il faut choisir !

p.14

TTS / PYTHON

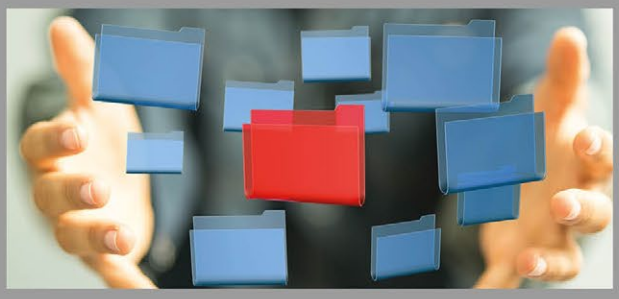
SYNTHÈSE VOCALE

FAITES PARLER VOTRE ORDINATEUR !

p.64

ETCKEEPER / GIT

Préservez vos configurations en toutes circonstances p.46



ACTUS / PERL

Perl 6 : le retour du dromadaire ? p.06

X11VNC / XVFB

Mettez en place un bureau virtuel sur votre serveur dédié p.32

C++ / LISP

Construisez un interpréteur et exécutez des calculs simples p.72

QR CODE / ALGO

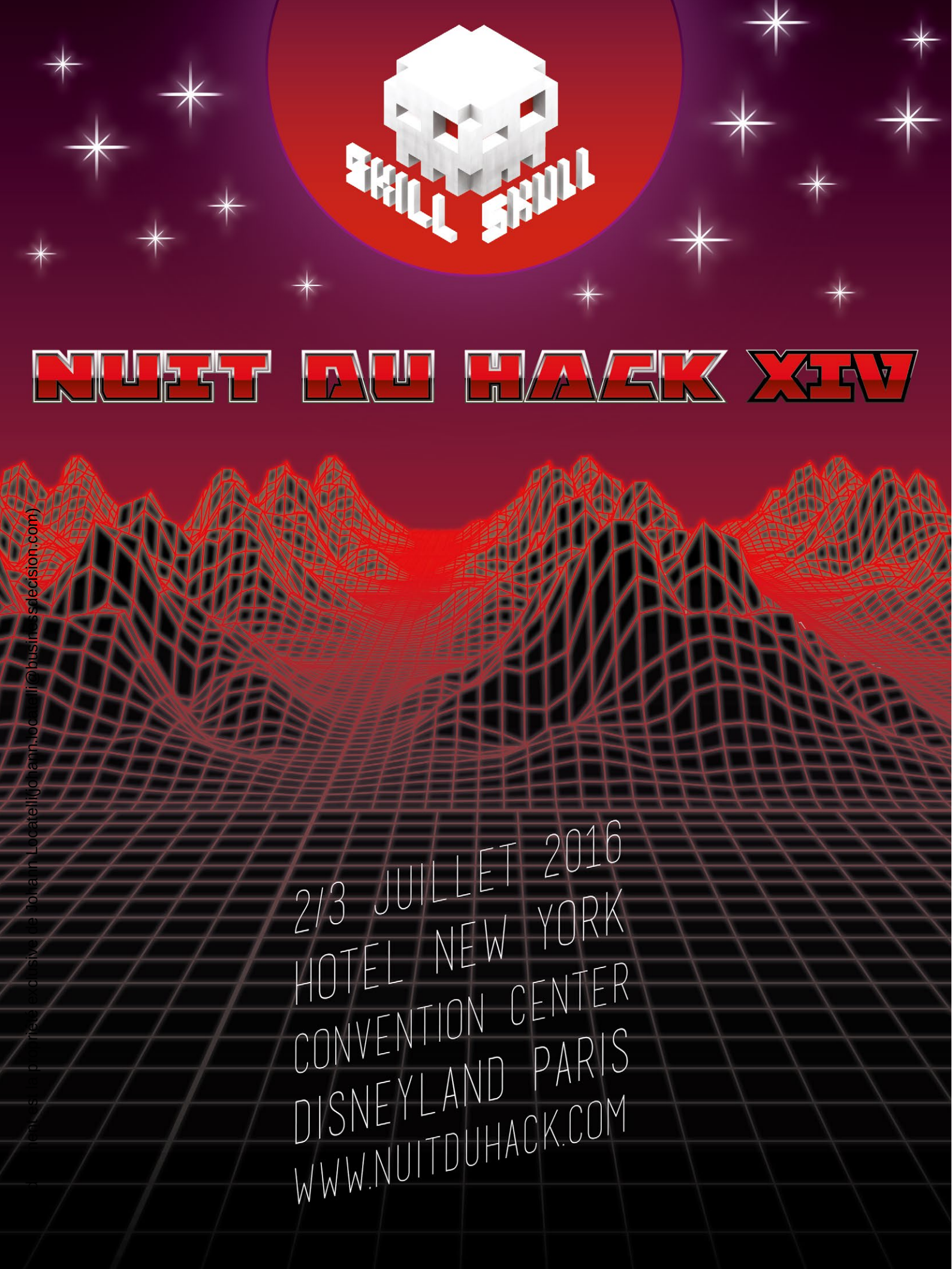
Étudiez le fonctionnement des QR codes

p.22

ET AUSSI : Tomcat en version « Stateless » - Rédiger des articles avec Asciidoc[tor] - Android Context

L 19275 - 193 - F: 7,90 € - RD

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



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 €

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

La complexité est un problème auquel tout développeur est confronté un jour ou l'autre et autant que ce soit le moins souvent possible ! La complexité ? Non, je ne parle pas de complexité algorithmique qui est nécessaire et permet d'optimiser le code. Il est question ici de la complexité induite par certains développeurs dans leurs programmes. N'avez-vous jamais rencontré un code censé réaliser une tâche extrêmement simple et clairement définie sans pour autant pouvoir le lire rapidement ? Un effet pernicieux de la fatigue ? Non, simplement un individu qui souhaite étaler son savoir, montrer qu'il « sait » de quoi il parle. Ainsi, pour répondre au problème « écris une fonction permettant d'additionner deux entiers », là où un développeur consciencieux se limiterait à l'écriture effective de ladite fonction, ces individus vont définir au minimum deux classes et une interface (et s'ils ont le moyen d'utiliser l'héritage, ça ne sera que mieux).

Quid de la philosophie UNIX : « ne faire qu'une seule chose, mais la faire bien » ? À quoi bon développer un système capable d'effectuer des opérations sur les nombres complexes quand on demande une simple addition d'entiers ? Cet exemple est bien entendu abrégé mais, aussi contradictoire que cela puisse paraître, la plus grande difficulté est de simplifier les choses. Pour pouvoir travailler en équipe et maintenir du code, il faut être capable d'écrire des instructions claires au sein d'une architecture qui aura peut-être demandé des heures de réflexions, mais paraîtra évidente. Le développeur qui « étale sa science » est un nuisible pour l'équipe, une source de perte de temps et de tensions (essayez de discuter avec quelqu'un qui a forcément raison).

Si nous changeons de domaine, prenons le dessin, n'importe quel étudiant des beaux arts sera capable de dessiner un « bête » taureau. En retrouver les lignes essentielles sera un travail autrement plus complexe et la vue du résultat final passera pour une évidence. On pourra entendre « un enfant de maternelle peut le faire » ou « moi en cinq minutes je fais la même chose »... Certes, une fois que le résultat est là, que tout le travail de simplification a été effectué, la reproduction sera plus aisée. Si vous cherchiez une idée de destination pour vos prochaines vacances, vous pourrez théoriquement voir les onze états successifs de la lithographie *« Le Taureau »* de Pablo Picasso au MoMA (*Metropolitan Museum of Art*). Mais comme les œuvres voyagent, il n'y sera peut-être pas... avec un peu de chance, vous pourrez la voir en France (elle était à Beaubourg il y a quelques années). Sinon on peut aussi en trouver des photos sur internet : http://www.moma.org/collection/works/62951 pour l'état III ou encore http://www.moma.org/collection/works/63062 pour l'état XI. Vous comprendrez de quoi je veux parler.

L'idée est la même lorsque l'on effectue une présentation devant un auditoire : créer des *slides* bourrés de texte, de code et d'équations est à la portée du premier venu. Simplifier le propos, alléger les supports visuels tout en expliquant des choses complexes, là est la réelle difficulté et la réelle preuve de maîtrise. Pour y parvenir, il n'y a qu'une solution : rester humble et continuer à apprendre, par exemple en lisant *GNU/Linux Magazine*... Bonne lecture !

Tristan Colombo

DISPONIBLE DÈS LE 06 MAI

GNU/LINUX MAGAZINE HORS-SÉRIE N°84 !

LE GUIDE POUR
TESTER LES TECHNIQUES POUR MIEUX SE DÉFENDRE !

DISPONIBLE DÈS LE 06 MAI

CHEZ VOTRE MARCHAND DE JOURNAUX ET SUR :

www.ed-diamond.com

SOMMAIRE

GNU/LINUX MAGAZINE FRANCE N°193

NOUVEAU LANGAGE PERL 6 : UNE EXPRESSIVITÉ SANS PRÉCÉDENT

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

La première version officielle de production du langage de programmation Perl 6 est sortie fin décembre 2015. Perl 6 reste dans l'esprit de Perl, et sa syntaxe présente de fortes ressemblances avec les versions antérieures de Perl, mais c'est véritablement un nouveau langage, résolument moderne et d'une puissance expressive sans précédent.

Mots-clés : Perl 6, Rakudo-Perl 6, Langage multiparadigme, Typage graduel, Opérateurs de listes

Résumé

Perl 6 est un nouveau langage de programmation propre, moderne et multiparadigme : il offre des possibilités de programmation procédurale, orientée objet et fonctionnelle ; il ne vous impose aucun de ces modèles de programmation, c'est vous qui choisissez selon vos besoins spécifiques et vos préférences personnelles. Il propose aussi au choix un typage statique fort ou un typage dynamique.

Après une longue période de gestation, la première version stable « de production » du nouveau langage Perl 6 a été officiellement annoncée en décembre 2015 et est disponible au téléchargement. La distribution stable complète s'appelle **Rakudo Star** et contient Rakudo Perl 6, le compilateur Perl 6, la machine virtuelle **MoarVM**, une suite de modules additionnels et une abondante documentation. Cette distribution est disponible pour les environnements Linux, Mac OS X et Windows (voir encadré).

Perl 6 est une nouvelle mouture du langage Perl, qui reste dans l'esprit des versions antérieures de Perl (« fais ce que je veux », « il y a plus d'une façon de le faire », etc.), mais les concepteurs de Perl 6 ont décidé de rompre avec la tradition de compatibilité ascendante de la syntaxe remontant à plus de 25 ans, ce qui a permis un toilettage assez important de la syntaxe et ouvert des possibilités réellement nouvelles et radicalement modernes.

1 Une brève introduction à Perl 6

Prenez un langage de programmation généraliste quelconque et affichez le résultat de la simple opération arithmétique suivante : `0.3 - 0.2 - 0.1`. Il y a toutes les chances que vous obteniez un résultat tel que `-2.77555756156289e-17` (en **Perl 5**), `-2.775558e-17` (en **C** sous `gcc`) ou `-2.7755575615628914e-17`

(en **Java**, **Python 3**, **Ruby** ou **TCL**). Ce dernier résultat paraît peut-être plus précis (plus de décimales), mais il se trouve en fait être légèrement plus loin de la vérité, puisque ce total devrait normalement être égal à **0**. Ces résultats erronés ne sont pas dûs à une faiblesse des langages en question, mais au fait que nos ordinateurs ne sont pas très bons en arithmétique avec des nombres fractionnaires.

Essayons avec la boucle REPL de Perl 6 :

```
> my $l'addition-devrait-être-zéro = .3 - .2 - .1;
0
> say sprintf "%.50f", $l'addition-devrait-être-zéro;
0.00000000000000000000000000000000000000000000000000
```

On voit que Perl 6 affiche la valeur correcte (**0**), même quand on demande d'afficher les 50 premières décimales. Ce résultat est correct parce qu'en interne, Perl 6 stocke des nombres décimaux dans un type *Rat* (rationnel), sous la forme d'un couple numérateur / dénominateur, et peut donc obtenir une précision arbitraire... et surtout un résultat correct. On peut même comparer le résultat à **0** :

```
> say "Vrai" if $l'addition-devrait-être-zéro == 0;
Vrai
```

Tout développeur un tant soit peu expérimenté sait qu'il ne faut surtout pas faire ce genre de comparaison dans la plupart des langages de programmation. En Perl 6, ça marche parfaitement. Cet exemple montre la mise en pratique du principe « *Do What I Mean* » (DWIM) cher aux concepteurs du langage Perl 6 : on s'attend normalement à ce que cette opération soit égale à **0**, mais on constate que ce n'est pas le cas dans la plupart des langages de programmation. Perl 6 « fait exactement ce que nous voulons dire ». Ce simple problème d'addition est un premier exemple de ce qu'un *langage radicalement moderne* devrait faire : sauf peut-être si nous développons des microcodes pour une puce embarquée, nous ne devrions plus, en 2016, avoir à nous demander si notre ordinateur sait compter correctement, ce devrait être acquis depuis longtemps. Avec Perl 6, votre ordinateur sait enfin le faire.

Installer et utiliser Rakudo / Perl 6

À l'heure où nous écrivons, les paquets proposés par les grandes distributions Linux proposent des versions trop anciennes. Vérifiez pour votre propre distribution, mais, par exemple, Debian propose un paquet « stable » datant de 2014 et un paquet de test de novembre 2015. Idéalement, il faudrait au moins la version 2016.01 de Rakudo-Star pour avoir la version stable de production de Perl 6. On peut imaginer que la situation va évoluer et que des paquets de la version de production seront disponibles dans les semaines ou les mois qui viennent pour diverses distributions de Linux, mais il est aujourd'hui préférable de télécharger le fichier **tar** à l'adresse http://rakudo.org/downloads/star/ et de l'installer soi-même. Par exemple :

```
$ wget http://rakudo.org/downloads/star/rakudo-star-2016.01.tar.gz
$ tar xzf rakudo-star-2016.01.tar.gz
$ cd rakudo-star-2016.01
$ perl Configure.pl --backend=moar --gen-moar
$ make
$ make rakudo-test
$ make rakudo-spectest
$ make install
```

Des informations complémentaires peuvent être trouvées sur le site de Rakudo : http://rakudo.org/how-to-get-rakudo/.

Une fois l'installation effectuée, vous voudrez sans doute mettre à jour votre **PATH** dans le fichier **~/.bashrc** puis sourcer ce fichier. Ensuite, en tapant **perl6 -v** à la ligne de commande, vous devriez avoir quelque chose du genre :

```
$ perl6 -v
This is Rakudo version 2016.01.1 built on MoarVM version 2016.01
implementing Perl 6.c.
```

En tapant **perl6** à l'invite du shell, on entre dans la boucle REPL (*read, evaluate, print loop*) qui affiche une invite de ligne de commande permettant de tester des instructions ou expressions Perl 6 simples.

```
> say "Hello World!";
Hello World!
```

Pour exécuter un script, tapez simplement **perl6** suivi du nom du programme :

```
perl6 script.pl6
```

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.

L'exemple ci-dessus illustre au passage quelques autres caractéristiques de Perl 6. Comme en Perl 5 (et en PHP), les noms de variables commencent par un caractère spécial, nommé *sigil*, ici le signe **$** pour une variable scalaire, suivi de l'identifiant proprement dit, qui doit commencer par un caractère alphabétique ou le caractère de soulignement **_**. Elles doivent être déclarées avant d'être utilisées, ici avec le mot-clef **my**. Perl 6 supporte pleinement l'Unicode, et les identifiants de variables ou de fonctions peuvent contenir des lettres françaises accentuées (ci-dessus, le « ê » et le « é »), mais on aurait aussi bien pu utiliser des lettres grecques :

```
> my $Ð = (5**.5 + 1)/2; # nombre d'or
1.618033988749895
> say "Le nombre d'or est égal à $Ð.";
Le nombre d'or est égal à 1.618033988749895.
> my $π = 4 * atan 1;
3.14159265358979;
```

On aurait également pu utiliser les lettres d'un quelconque autre alphabet. Là encore, la modernité... De plus, comme on le voit dans le premier exemple de code, ces identifiants peuvent également contenir des tirets (« - ») ou des apostrophes (« ' »), à condition qu'ils soient situés entre deux lettres.

On constatera également dans les exemples ci-dessus que Perl 6 n'a généralement pas besoin de beaucoup de parenthèses, sauf quand c'est nécessaire pour des raisons de précédence des opérateurs (comme dans la formule du nombre d'or). Par exemple, les arguments d'une fonction interne peuvent généralement être placés sans parenthèses après le nom de la fonction appelée (et séparés par des virgules s'il y en a plusieurs). De même, on peut souvent enchaîner plusieurs fonctions comme dans l'exemple **say sprintf ...** ci-dessus, Perl 6 sait très bien prendre la ou les valeurs de retour d'une fonction et la ou les passer en argument à la fonction placée à sa gauche. Cela rend la syntaxe souvent plus fluide et plus limpide.

Le caractère **#** introduit un commentaire s'étendant jusqu'à la fin de la ligne. Il existe d'autres formes de commentaires pouvant être placés au milieu d'une ligne de code ou pouvant couvrir plusieurs lignes, par exemple :

```
say #`(Commentaire au milieu d'une ligne de code) "Hello World
!";  # imprime Hello World !
say #`[Ceci est
un commentaire
multiligne] "Hello World !";
# idem
```

Mais nous n'entrerons pas plus dans les détails ici.

2 Les variables

2.1 Les diverses sortes de variables

Il existe trois principales sortes de variables, qui se distinguent par le *sigil* précédant l'identifiant proprement dit :

- le *sigil* **$** indique une variable scalaire, c'est-à-dire une variable ne contenant qu'une seule valeur (par exemple une valeur numérique, une chaîne de caractères, un objet ou une référence vers une autre entité) ;
- le *sigil* **@** désigne une variable de tableau, c'est-à-dire une liste de valeurs indexées par des entiers ;
- le *sigil* **%** dénote une table de hachage, qui est définie comme un ensemble de paires clef-valeur.

On accède aux éléments d'un tableau avec l'opérateur **[...]** et à ceux d'un hachage avec **{...}**. Contrairement à Perl 5, une variable de tableau ou de hachage conserve son *sigil* d'origine quand on accède à ses éléments individuels :

```
> my @tableau = 1, 2, 4, 6;
[1 2 4 6]
> say @tableau[2];
4
> my %hachage = jan => 1, fév => 2, mar => 3;
fév => 2, jan => 1, mar => 3
> say %hachage{"fév"};
2
```

2.2 Portée des variables

Nous avons vu que le mot-clef **my** sert à déclarer une variable. Plus précisément, il introduit une variable *lexicale*, c'est-à-dire une variable dont la portée est limitée au bloc de code dans lequel elle se trouve. Un bloc de code est, en simplifiant un peu, un fragment de code délimité par des accolades ouvrantes et fermantes servant à structurer le code. Une variable lexicale est visible et utilisable entre l'endroit où elle est déclarée avec le **my** et l'accolade fermant le bloc où se trouve la déclaration. Si la déclaration a lieu à l'extérieur de tout bloc, alors la variable est globale au script (ce qui n'est généralement pas conseillé).

```
{
    my Str $var = "texte";
    say $var;   # affiche "texte"
}
say $var; # ERREUR: $var n'est plus accessible
```

On peut aussi avoir deux variables de même nom ayant des valeurs différentes selon la portée dans laquelle elles se trouvent :

```
my Int var = 42;
{
    my Str $var = "texte";
    say $var;   # affiche "texte"
}
say $var;       # -> 42
```

Ces exemples sont un peu artificiels à ce stade, mais cette possibilité s'avérera très utile quand nous utiliserons des blocs décrivant des boucles ou le corps de fonctions.

2.3 Les types de données

Une variable peut être déclarée avec un type imposant des contraintes sur ses valeurs possibles :

```
my Int $c;
$c = 4;        # tout va bien, $c vaut 4
say $c.WHAT    # (Int)
$c = 4.2;      # ERREUR : Type check failed in assignment to $c...
```

Mais la déclaration n'est pas nécessaire et Perl 6 détermine lui-même le type de la donnée dans la mesure du possible :

```
my $d = 4.2;                    # Pourrait aussi s'écrire : my
$d = Rat.new(42, 10);
say $d.WHAT;                    # (Rat)
say '$d = ', $d.numerator, " / ",
    $d.denominator;             # affiche : $d = 21 / 5
```

On dit que Perl 6 est typé de façon « graduelle » : il autorise aussi bien un typage statique (comme C ou Java) qu'un typage dynamique (comme Perl 5, Ruby ou Python). Le meilleur de deux mondes, en quelque sorte. En fait, les variables **$c** et **$d** ci-dessus sont assimilables à des objets, respectivement de type **Int** et **Rat**, ce qui explique la possibilité d'invoquer sur eux les méthodes **.WHAT** ou **.numerator**. Pour les types les plus courants, il n'est souvent pas nécessaire d'invoquer le constructeur **.new** pour les obtenir. Ainsi, il existe plusieurs façons de créer des objets de type complexe :

```
my $z1 = 5 + 3i;
say $z1.WHAT;                   # (Complex)
my $z2 = Complex.new(4, 9);     # 4+9i
my Complex $z3 = 2 + 5i;
say $z1 + $z2 + $z3;            # affiche : 11+17i
```

L'utilisation d'une simple affectation est suffisante pour créer un objet du type voulu, mais les deux syntaxes utilisant explicitement le type **Complex** sont un peu plus sûres, car elles vérifient le type du littéral affecté à la variable. De même, considérons des objets de type **Date** :

```
my $d = Date.new(2015, 12, 24); # Veille de Noël : 2015-12-24
say $d.year;                    # 2015
say $d.month;                   # 12
say $d.day;                     # 24
say $d.day-of-week;             # 4  (donc, jeudi)
my $n = Date.new('2015-12-31'); # Saint-Sylvestre
say $n > $d;                    # False
say $n - $d;                    # 7 (delta 7 jours)
say $n + 1;                     # 2016-01-01
say 1+$n.later(:2months);       # 2016-03-01
```

Remarquez au passage combien la manipulation des dates devient facile et intuitive. C'est cela aussi la modernité de Perl 6. Ici, la syntaxe utilisant le constructeur **new** était nécessaire parce que sans elle, le programme ne pourrait reconnaître une date et effectuerait simplement une double soustraction entre entiers :

```
> my $d = 2015-12-25;  # Non, ce n'est pas Noël
1978
```

2.4 Les variables scalaires

Les variables scalaires, introduites par le *sigil* **$**, ne contiennent qu'un seul élément (mais cet élément peut lui-même être composite, par exemple si c'est un objet comme nous venons de le voir dans les exemples ci-dessus). Les variables scalaires contiennent souvent des chaînes de caractères ou des nombres.

2.4.1 Chaînes de caractères

Les chaînes de caractères sont des séquences immuables de caractères quelconques. Elles sont généralement encadrées par des guillemets ou des apostrophes. La différence est que les variables (et les séquences d'échappement) sont interpolées dans une chaîne de caractères entre guillemets, mais pas dans celles entre apostrophes.

```
my $user = 'Laurent';
say "Bonjour, $user !";    # Bonjour Laurent !
say 'Bonjour, $user !';    # Bonjour $user !
```

Le tilde « ~ » est l'opérateur de concaténation de chaînes :

```
say "Hello" ~ "World !";   # Hello World !
```

Il existe de nombreuses fonctions ou méthodes travaillant sur des chaînes :

```
my $nom = "Charlie";
say flip $nom;  # eilrahC (syntaxe fonctionnelle)
say $nom.flip;  # eilrahC (syntaxe de méthode)
say uc $nom;    # CHARLIE
```

```
say $nom.uc;    # idem
say $nom.chars; # 7 (nombre de caractères)
say substr $nom, 2, 3; # arl (sous-chaîne)
say "Je suis " ~ $nom; # Je suis Charlie (concaténation)
```

On remarque que les sous-routines internes de Perl 6 admettent pour la plupart une syntaxe de fonction et une syntaxe de méthode. Il est possible de les combiner à volonté, notamment pour clarifier l'intention ou la précédence :

```
> say flip "Charlie".substr(2, 3).uc
LRA
```

2.4.2 Données numériques

Nous avons déjà vu des exemples de données numériques de types **Int** (entier) ou **Rat** (rationnel). La racine carrée de **2**, le logarithme de **5** ou **1017** sont de type **Num** :

```
say 2.sqrt.WHAT; # (Num)
say 5.log.WHAT;  # (Num)
say 1e17.WHAT;   # (Num)
```

De nombreuses fonctions ou méthodes permettent de travailler sur les données numériques ou seulement sur certains types numériques :

```
say 19.is-prime;  # True (19 est premier)
say 4.7.nude;     # (47 10), c-à-d 47/10
```

Cela n'est pas spécifique aux variables numériques, mais notons au passage qu'il est possible de définir dynamiquement des « sous-types » ou sous-ensembles de types existants. On peut par exemple créer un type nombre impair :

```
subset Impair of Int where { $_ % 2 } # non divisibles par 2
# Impair est maintenant un sous-type
my Impair $x = 3; # OK
my Impair $y = 2; # erreur de type
```

2.5 Les tableaux

Les tableaux sont des listes contenant des valeurs multiples. Les valeurs n'ont pas besoin d'être du même type (on peut par exemple mélanger des chaînes et des nombres), mais c'est souvent une bonne idée qu'elles le soient dans la mesure où les éléments d'un tableau doivent en principe avoir une certaine cohérence sémantique. On accède aux valeurs individuelles d'un tableau à l'aide d'indices qui sont des nombres entiers, le premier élément d'un tableau portant l'indice **0**.

```
my @nombres_shadoks = ['GA', 'BU', 'ZO', 'MEU'];
say @nombres_shadoks[1];     # imprime: BU
```

Si les éléments d'un tableau sont des chaînes de caractères sans espace, on peut utiliser un opérateur de citation de liste **<...>** pour écrire plus simplement, sans guillemets, apostrophes, ni virgules :

```
> my @nombres_shadoks = <GA BU ZO MEU>;
[GA BU ZO MEU]
> say @nombres_shadoks.elems; # nombre d'éléments
4
> my $dernier = pop @nombres_shadoks;
MEU
> say @nombres_shadoks;
[GA BU ZO]
> say elems @nombres_shadoks; # nombre d'éléments
3
> say "Les shadoks ont perdu leur dernier chiffre: $dernier";
Les shadoks ont perdu leur dernier chiffre: MEU
> push @nombres_shadoks, $dernier;
[GA BU ZO MEU]
```

La fonction **pop** retire le dernier élément du tableau et le renvoie ; la fonction **push** ajoute un (ou plusieurs) élément(s) à la fin d'un tableau. On aurait aussi pu utiliser une syntaxe de méthode :

```
> my $dernier = @nombres_shadoks.pop
MEU
> say @nombres_shadoks
[GA BU ZO]
> @nombres_shadoks.push($dernier);
[GA BU ZO MEU]
```

La fonction **splice(a, n)** retire **n** élément(s) d'un tableau à partir de la position **a** et les renvoie:

```
> my @nombres_shadoks = <GA BU ZO MEU>;
[GA BU ZO MEU]
> my @elem_1_2 = splice @nombres_shadoks, 1, 2;
[BU ZO]
```

Mais si l'on désire récupérer collectivement plusieurs éléments d'un tableau *sans* modifier le tableau, on peut utiliser une syntaxe de *tranches de tableaux* :

```
my @nombres_shadoks = <GA BU ZO MEU>;
say @nombres_shadoks[0..2];  # (GA BU ZO)
```

Il existe de nombreuses autres fonctions ou méthodes utilisables, fournies notamment par les classes internes **Array** et **List** de Perl.

Les tableaux peuvent être multidimensionnels. On peut accéder aux éléments d'un tableau multidimensionnel en séparant les indices par un « **;** » :

```
my $prem-rang_deux-col = @tableau[0;1];
```

2.6 Les hachages

Un hachage est un ensemble de paires clef-valeur. L'idée générale est la même que celle des dictionnaires ou *maps* dans d'autres langages.

```
> my %capitales = ("Italie", "Rome", "Allemagne", "Berlin",
"Espagne", "Madrid");
Allemagne => Berlin, Espagne => Madrid, Italie => Rome
```

On peut rendre le code plus concis avec une syntaxe de liste :

```
> my %capitales = <Italie Rome Allemagne Berlin Espagne Madrid>;
Allemagne => Berlin, Espagne => Madrid, Italie => Rome
```

Et on peut le rendre plus clair en utilisant la « virgule grasse » **=>** dès l'initialisation :

```
my %capitales = (Italie => "Rome", Allemagne => "Berlin", Espagne
=> "Madrid");
```

La « virgule grasse » **=>** a essentiellement le même rôle qu'une virgule, mais elle rend l'intention plus claire et dispense de mettre la clef entre guillemets.

On peut ajouter un nouvel élément au hachage par une affectation avec une nouvelle clef :

```
%capitales{"France"} = "Paris";
```

Ou :

```
%capitales<France> = "Paris";
```

La méthode **push** permet également d'ajouter une nouvelle paire au hachage :

```
push %capitales, (Danemark => "Copenhague");
%capitales.push: (USA => "Washington");
say %capitales.elems ;   # 6 (nombre de paires)
```

Les méthodes **kv**, **keys** et **values** renvoient respectivement des listes de paires, de clefs et de valeurs :

```
> say %capitales.kv
(France Paris Allemagne Berlin Italie Rome USA Washington
Danemark Copenhague Espagne Madrid)
> say %capitales.keys;
(France Allemagne Italie USA Danemark Espagne)
> say %capitales.values;
(Paris Berlin Rome Washington Copenhague Madrid)
```

Notons qu'un hachage stocke et restitue les paires dans un ordre apparemment aléatoire (indépendant de celui dans lequel elles ont été créées), mais les trois méthodes renvoient les éléments dans un ordre consistant.

3 Les opérateurs

Perl 6 possède la plupart des opérateurs communs aux langages de programmation usuels et de nombreux autres.

.method	Méthode post-fixée
++ --	Auto-incrémentation, auto-décrémentation (préfixées ou post-fixées)
******	Exponentielle
! + - ~ ? | ||	Symboles unaires : négation logique, plus, moins, concaténation, coercition booléenne, ou logique
*** / % %% div gcd lcm**	Multiplication, division, modulo, divisibilité, division entière, PGDC, PPMC
+ -	Addition, soustraction
x xx	Réplication de chaîne (renvoie une chaîne), réplication d'élément (renvoie une liste)
~	Concaténation de chaînes
~~ != == < <= > >= eq ne lt le gt ge	Opérateur de comparaison intelligente, opérateurs de comparaisons numériques, opérateurs de comparaisons de chaînes
&&	ET booléen (de haute précédence)
||	OU booléen (de haute précédence)
?? !!	Opérateur conditionnel ternaire
= => += -= **= xx= .=	Opérateurs d'affectation
so not	Booléens unaires de basse précédence : coercition booléenne et non logique
and andthen	ET booléen (de basse précédence)
or xor orelse	OU booléen (de basse précédence)

3.1 Principaux opérateurs

Le tableau page précédente résume les opérateurs les plus communs, par ordre de précédence descendante (de la priorité la plus forte à la plus faible).

Nous verrons plus loin qu'il est facile de construire ses propres opérateurs.

3.2 On en a rêvé, Perl 6 le fait

Les opérateurs de comparaison logique peuvent être chaînés comme en arithmétique, ce qui peut simplifier notablement l'écriture :

```
say "Valeurs dans l'ordre" if $u < $v < $x < $y < $z;
# équivaut à : ... if $u < $v and $v < $x and $x < $y ...
# ou à : ... if ($u < $v) and ($v < $x) and ...
```

De même, les jonctions permettent d'écrire des comparaisons concises de ce style :

```
my $x = 7;
say "Trouvé" if $x == 4|6|9|7|15; # -> Trouvé
# Peut aussi s'écrire :
say "Trouvé" if $x if 7 == any <4 6 9 7 15>;
# équivaut à : ... if ($x == 4) or ($x == 6) or ...
```

3.3 Les métaopérateurs et hyperopérateurs

Les opérateurs travaillent sur des données, les métaopérateurs travaillent sur des opérateurs et permettent en quelque sorte de créer de nouveaux opérateurs.

Le métaopérateur de réduction **[...]** permet de transformer un opérateur infixé associatif en un opérateur de liste renvoyant un scalaire :

```
> say [+] 1, 2, 3, 4;
10
```

Ici, tout se passe comme si on avait placé l'opérateur **+** entre chaque élément de la liste, comme si on avait écrit :

```
> say 1 + 2 + 3 + 4;
10
```

On peut de même calculer très facilement la factorielle de **10** en modifiant l'opérateur de multiplication :

```
my $fact10 = [*] 1..10;   # -> 3628800
```

Il existe d'autres métaopérateurs. Par exemple, le métaopérateur **X** renvoie un produit cartésien entre deux ou plusieurs listes :

```
> <a b c> X 1, 2;
((a 1) (a 2) (b 1) (b 2) (c 1) (c 2))
```

Un hyperopérateur applique une opération à chaque membre d'une liste ou de plusieurs listes et renvoie une nouvelle liste. Ici, chaque élément de la liste est multiplié par 5 :

```
> my @a = 6..10;
[6 7 8 9 10]
> say  5 "*" @a;
[30 35 40 45 50]
```

À noter que cet hyperopérateur utilise en principe les guillemets français **"..."**, mais vous pouvez utiliser les chevrons ASCII **<<...>>** si votre éditeur ne vous permet pas d'écrire facilement ces guillemets français.

Avec deux ou plusieurs listes, l'hyperopérateur permet d'effectuer des opérations membre à membre sur les listes. Voici par exemple une concaténation membre à membre entre trois listes :

```
> my @x = ('a'..'e') "~" (3..7)  "~" ('v'..'z');
[a3v b4w c5x d6y e7z]
```

Les métaopérateurs et hyperopérateurs créent de nouveaux opérateurs en modifiant la sémantique d'opérateurs existants. Nous verrons dans un prochain article qu'il est possible de créer facilement des opérateurs entièrement nouveaux. Tout cela tend à rendre le langage intrinsèquement malléable et extensible.

Conclusion

Ce bref tour d'horizon initial a permis de faire connaissance avec les principaux types de données de Perl 6 et d'apprendre quelques-unes des principales caractéristiques de sa syntaxe. Perl 6 est un langage propre, typé, concis, puissant et expressif. Il est également résolument moderne et offre des fonctionnalités souvent uniques et à la pointe du progrès dans le domaine des langages de programmation.

Le prochain article complétera ce tour d'horizon en passant notamment en revue les structures de contrôle, les fonctions et la création dynamique de nouveaux opérateurs. Là encore, le lecteur pourra constater que non seulement ce langage est innovant et généralement en avance sur ce qui existe ailleurs, mais qu'il est aussi extensible presque à volonté et donc éminemment susceptible de rester à l'avenir en avance sur son temps. ■

DISPONIBLE LE 27 MAI

LINUX PRATIQUE HORS-SÉRIE N°36 !

PERSONNALISEZ VOTRE SITE WORDPRESS

DISPONIBLE LE 27 MAI

CHEZ VOTRE MARCHAND DE JOURNAUX ET SUR :

www.ed-diamond.com

CODE ET MAINTENABILITÉ

The Cheshire Cat [Everyone here is mad]

Écrire du code, c'est ce que nous faisons. Mais pensons-nous au futur quand nous l'écrivons ? Ne devrions-nous pas ? Qu'est-ce qui fait que notre code sera ou ne sera pas maintenable et comment améliorer les choses ?

Mots-clés : Humeur, Développeur, Maintenabilité, Équipe, Code, Bonne pratique

Résumé

Le but du code, ce n'est pas d'être écrit, le code en lui-même n'est pas une finalité. Le code est écrit pour que le programme qui le constitue fonctionne et ce programme doit fonctionner « longtemps ». Qu'est-ce que cela sous-entend pour nous, développeurs ? C'est ce dont nous allons commencer à parler ce mois-ci.

Depuis combien de temps êtes-vous sur votre projet actuel ? Dans combien de temps allez-vous changer de projet ? De poste ? D'entreprise ? Nous sommes aujourd'hui dans une situation assez étrange qui fait que longtemps après votre départ, votre code lui est toujours là. C'est un peu votre héritage que vous léguez aux générations futures de l'équipe projet qui prendra votre relève. Votre code durera d'ailleurs peut-être bien plus que ce que vous ne le pensiez, bien plus que ce que vous ne le voudriez d'ailleurs. Personnellement, je sais que le code que j'ai écrit il y a dix ans, alors que mes griffes étaient souples et piquantes, tourne encore. Et que des équipes projets bossent chaque jour, pour faire que les choses continuent à fonctionner. Et pour le coup, je me dis que certains aimeraient bien me tirer les moustaches. Pour ce premier article concernant la maintenabilité du code, étudions deux travers que nous avons bien souvent et qui font que celles et ceux qui héritent de nos projets ont parfois envie de nous étriper.

1 Les développeur·euse·s sont des gens intelligents et veulent le prouver

Il y a quelques années, une grosse dizaine, je faisais donc du C++. J'aimais énormément ce langage, c'est d'ailleurs toujours le cas. Ce que j'aime avec le C++, c'est tout ce que l'on peut faire avec lui et le fait que la courbe d'apprentissage soit, à partir d'un moment, assez abrupte. La méta-programmation à base de *template* n'est en effet pas le plus simple des concepts. Faire du C++ et jouer avec ces concepts-là me donnait l'impression de me battre avec un casse-tête de haut niveau. Réussir à faire ce que je voulais d'une manière que je trouvais alors élégante était le plus important et me procurait une énorme satisfaction. La même que celle que l'on peut éprouver lorsqu'on arrive à démontrer d'une manière particulièrement élégante quelque chose en mathématiques. Je me souviens d'une conversation concernant ce projet. Au final, la conclusion de notre discussion était que ceux qui allaient reprendre les choses allaient vraiment devoir connaître le C++ en long, en large et en travers. Le fait que tout ce que j'avais pu mettre en place était totalement *overkill* et tout sauf limpide ne m'avait pas effleuré. Après tout, c'était du C++ valide et de haut niveau. Si c'était défini dans le langage, c'était pour être utilisé. Et si les dev C++ qui reprendraient le projet ne comprenaient pas ce que j'avais fait, ce serait de leur faute, ils n'avaient qu'à être meilleurs. Fichtre, je me tirerais vraiment les moustaches très fort pour avoir pensé cela.

Mais comme je le dis en titre, les dev sont des gens intelligents et veulent le prouver. Quitte à trouver des solutions tarabiscotées juste pour le plaisir de les trouver, de les écrire et de pouvoir ensuite les relire en s'auto-congratulant en mode : « Mais oui, c'est vraiment moi qui ai écrit ces quelques lignes vraiment super intelligentes ».

Un exemple : imaginons que vous vouliez multiplier un chiffre par quatre en étant sûr que le résultat soit positif. On peut tout simplement utiliser quelque chose comme cela :

```
return Math.Abs(i * 4);
```

Mais il y aura toujours quelqu'un pour se dire que ce n'est pas assez optimisé, pas assez intelligent. Il vous pondra quelque chose comme cela :

```
return i > 0 ? i << 2 : ~(i << 2) + 1;
```

Alors oui, je sais. Pas besoin de me menacer tout de suite avec votre grande baignoire d'eau froide. Oui, dans certains cas, le bon code à écrire c'est celui utilisant les décalages. Mais soyons honnêtes, dans combien de cas le projet impose-t-il d'écrire la version incompréhensible utilisant les décalages plutôt que celle moins optimisée mais beaucoup plus claire ?

Et puis rappelez-vous les préceptes de tonton Agile. On vous remerciera d'avoir écrit du code clair et simple à lire. On ne vous remerciera pas d'avoir fait le code le plus rapide possible, mais qui nécessite un commentaire (qu'en plus vous n'écrirez pas) pour être compris.

Et si vous voulez savoir s'il est nécessaire d'écrire un code alambiqué mais ultra performant plutôt qu'un code clair et juste performant, posez-vous une question simple : est-ce que les avantages de ce code superbement intelligent ne vont pas être totalement annulés, dans le futur, par le temps qui va être passé à comprendre à quoi il sert et à le maintenir ? Si la réponse est oui, vous savez quoi faire ...

2 Les développeur·euse·s aiment découvrir des choses

Nous aimons, tous et toutes, tels les membres du jury d'une émission de télé-crochet quelconque, découvrir de nouvelles pépites. Nous n'appuyons pas sur un gros bouton rouge pour faire retourner l'énorme fauteuil dans lequel nous sommes assis, non. À la place nous utilisons le *framework* qui a moins de cinquante *commits* d'âge sur GitHub et qui est maintenu par trois personnes dans le monde dans un projet en production qui a vocation à durer plusieurs années.

Alors bien entendu qu'il ne faut pas se transformer en vieux chat ronflant et ronchon. Bien entendu qu'il ne faut pas se scléroser sur des technos anciennes juste parce qu'on les maîtrise bien et qu'il faut être capable d'évoluer, de passer au bon moment sur les technos qui sont prometteuses et qui apportent un vrai plus. Par exemple, il faut savoir ne pas choisir Flex pour un projet Web en 2010, même si cela fait plusieurs années qu'on en fait.

Mais à mon sens chaque choix technologique devrait se faire en ayant deux questions à l'esprit. Lesquelles sont :

- Est-ce que je fais le bon choix pour le projet ou est-ce que j'ai simplement envie de me faire plaisir en testant un truc, et puis je m'en fous, j'ai fini ma mission dans quatre ou cinq mois, je n'aurai pas à gérer la crise si mon choix est mauvais ?
- Est-ce que si c'était quelqu'un d'autre qui avait fait le choix que je m'apprête à faire et que j'étais celui ou celle qui reprenait les choses dans quelques mois, j'aurais envie de hurler ?

Et puis une fois que l'on a fait un choix, il faut savoir s'y tenir ou si l'on s'est trompé et que l'on doit finalement changer un élément de notre *stack*, il faut avoir une bonne raison. Je ne suis pas sûr, par exemple, que si vous avez passé six ou sept mois à peaufiner une belle app Web en **React**, la bonne chose à faire, lorsque vous découvrez **Preact** soit de décider de tout refaire avec celui-ci (même si oui, je sais, Preact est largement compatible avec React, la migration se ferra sans douleur, etc., etc.).

Conclusion

Un grand nombre de développeur·euse·s que j'ai rencontré, surtout dans le dev Web d'ailleurs, pensaient tous que leur code n'allait pas être très longtemps en production, que donc ce n'était pas vraiment grave s'ils faisaient les mauvais choix techniques ou s'ils ou elles se faisaient plaisir à faire de l'optimisation de bout de ficelle ou à étaler leur connaissance des possibilités de la techno utilisée. Et la plupart du temps, ils avaient tort. Et lorsqu'ils ou elles en reparlaient, quelques mois ou années après, c'était avec un rougissement de honte. Alors si à l'avenir, vous voulez vous éviter de tel rougissement de honte, rappelez-vous que vous n'avez pas à prouver votre intelligence en écrivant le code le plus compliqué possible. Et que vous n'avez pas besoin d'être le plus *early adopter* des dev pour être un bon dev. ■

PEUT-ON VRAIMENT CALCULER AVEC UN ORDINATEUR ?

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

Alors que le recours à l'ordinateur pour calculer toujours plus vite, modéliser toujours plus finement est inscrit dans une logique de progrès, il paraît indispensable de comprendre comment sont effectués ces calculs pour garder un regard critique sur la qualité des résultats qui ne sont qu'une approximation (plus ou moins bonne) des vraies valeurs. Comment sont stockés les réels, comment sont effectués les calculs, d'où viennent les erreurs de précision, peut-on les contenir ? Vous l'apprendrez en lisant cet article et sa suite.

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

Résumé

Dans cette première partie, nous commencerons par donner quelques exemples afin de montrer que les erreurs de calcul arrivent très vite et peuvent, dans certains cas, donner des résultats totalement faux. Nous expliquerons ensuite comment sont stockés les réels dans les ordinateurs et montrerons ainsi que ces nombres réels (les flottants) ne représentent qu'un sous-ensemble des vrais réels.

Nous sommes tous (plus ou moins) conscients que les calculs réalisés sur ordinateur conduisent à des pertes de précision. Mais en connaissons-nous les raisons ? Avons-nous les moyens de les contenir ?

Avant de répondre à ces questions, je donnerai quelques exemples concrets pour illustrer le fait que nos ordinateurs commettent des erreurs, souvent minimes (mais avec des conséquences potentiellement graves), parfois très importantes. Puis, j'expliquerai comment sont stockés les réels sur ordinateur afin de détailler, dans un second article, les mécanismes mis en œuvre pour calculer et comprendre d'où viennent ces imprécisions.

1 Des erreurs dans les calculs

1.1 Somme de réels

Nous allons tout d'abord effectuer la somme de **n** fois la valeur (réelle) **x** sur un ordinateur selon la formule

$$\sum_{i=1}^{n} x$$

Cet exemple est purement didactique car personne n'aurait l'idée de faire ce calcul ainsi alors que le résultat s'obtient directement (en une opération) avec $n.x$

On trouve cependant de tels codes (ou très proches) ici ou là...

Pour cela, nous allons utiliser le programme **somme.cpp** écrit en C++ :

```
#include <iomanip>
#include <iostream>

int main(){
  float x;
  std::cout<<"Entrez la valeur de x"<<std::endl;
  std::cin>>x;
  float res = 0.0;
    for (int i=0 ; i<1000 ; i++){
    res += x;
  }
```

```
  std::cout<<"Somme de 1000 fois "<<x<<" =
"<<std::setprecision(10)<<res<<std::endl;
  return 0;
}
```

Note

On utilise ici **std::setprecision(10)** pour afficher **10** chiffres (significatifs).

Lorsque l'on exécute ce programme, on obtient les résultats suivants :

Valeur de x	Résultat obtenu
0,5	500
0,25	250
0,1	99,99904633
0,7	700,006958

On observe donc que pour certaines valeurs (**0,5** ou **0,25**), le résultat obtenu est exact alors que pour d'autres (**0,1** ou **0,7**), il est proche de la vraie valeur, parfois par excès, parfois par défaut.

La précision peut évidemment être augmentée en utilisant des réels « double précision ». Ainsi, en remplaçant les **float** par des **double** (voir **somme_double.cpp** ci-dessous), on obtient les résultats reportés dans le tableau suivant (avec 20 chiffres significatifs) :

```
#include <iomanip>
#include <iostream>

int main(){
  double x;
  std::cout<<"Entrez la valeur de x"<<std::endl;
  std::cin>>x;
  double res = 0.0;
    for (int i=0;i<1000;i++){
    res += x;
  }
  std::cout<<"Somme de 1000 fois "<<x<<" =
"<<std::setprecision(20)<<res<<std::endl;
  return 0;
}
```

Valeur de x	Résultat obtenu
0,5	500
0,25	250
0,1	99,999999999998593125
0,7	700,00000000000636646

Les résultats sont effectivement plus proches des vraies valeurs mais le constat est le même, on a simplement repoussé le problème : les vraies valeurs ne sont pas atteintes pour **0,1** ou **0,7**.

Sommes-nous alors dans l'obligation de n'utiliser que certaines valeurs dans nos programmes pour que les résultats soient justes ?

1.2 Petites erreurs, grandes conséquences

On pourrait penser que ces petites imprécisions sont sans conséquence. C'est bien souvent le cas, fort heureusement ! Mais, ce type d'erreurs peut conduire à des drames, comme lors de la guerre du Golfe en 1991 en Irak. La coalition américaine disposait d'antimissiles Patriot capables d'intercepter en vol des missiles ennemis. Ce dispositif a parfaitement fonctionné jusqu'au jour où il a manqué sa cible causant la mort de 28 personnes. L'horloge interne de l'antimissile incrémentait le temps par intervalle de **0,1** seconde (avec une erreur de **0,0000000953** en étant codé sur 24 bits). Au bout de 100 heures, l'erreur cumulée était de 0,34s, suffisamment élevée pour que l'antimissile manque sa cible (le missile parcourt environ 500m en 0,34s).

Ce drame aurait donc pu être facilement évité en choisissant une autre valeur d'incrémentation ; ainsi avec un intervalle de temps de **0,25** par exemple, le calcul aurait été parfaitement juste, l'antimissile n'aurait pas manqué sa cible et la vie de 28 personnes aurait été épargnée.

1.3 Des erreurs parfois beaucoup plus importantes

1.3.1 La suite de Muller

La suite de Muller est définie par :

$$\begin{cases} u_0 = 2 \\ u_1 = -4 \\ u_{n+1} = 111 - \frac{1130}{u_n} + \frac{3000}{u_n . u_{n-1}} \end{cases}$$

Le programme **muller.cpp** (ci-dessous) permet de calculer (en double précision) et d'afficher les 30 premières valeurs de cette suite :

```
#include <iomanip>
#include <iostream>

int main(){
  double u[30];
  u[0] = 2;
  u[1] = -4;

  for(int i=2 ; i<30 ; i++){
    u[i] = 111. - 1130./u[i-1] +
3000./(u[i-1]*u[i-2]);
    std::cout<<"u"<<i<<" =
"<<u[i]<<std::endl;
  }
  return 0;
}
```

L'exécution de ce programme donne les résultats suivants :

```
$./muller
u2 = 18.5
u3 = 9.37838
u4 = 7.80115
u5 = 7.15441
u6 = 6.80678
u7 = 6.59263
u8 = 6.44947
u9 = 6.34845
u10 = 6.27444
u11 = 6.2187
u12 = 6.17585
u13 = 6.14263
u14 = 6.12025
u15 = 6.16609
u16 = 7.23502
u17 = 22.0621
u18 = 78.5756
u19 = 98.3495
u20 = 99.8986
u21 = 99.9939
u22 = 99.9996
u23 = 100
u24 = 100
u25 = 100
u26 = 100
u27 = 100
u28 = 100
u29 = 100
```

On s'aperçoit que cette suite semble converger vers la valeur **100** (comme sur tout système en précision finie) alors qu'en réalité elle converge (mathématiquement) vers la valeur **6**.

Comment une telle erreur est-elle possible ?

1.3.2 La fonction de Rump

La fonction de Rump est définie par :

$$f(a,b) = (333 + \tfrac{3}{4})b^6 + a^2(11a^2b^2 - b^6 - 121b^4 - 2) + \tfrac{11}{2}b^8 + \tfrac{a}{2b}$$

En prenant pour valeur **a = 77617** et **b = 33096**, on obtient **≈-2,34.1029** (en simple précision) et **≈-1,1.1021** (en double précision). En codant cette fonction sous une autre forme (mathématiquement équivalente), on peut aussi obtenir les valeurs **≈+7,09.1029** ou encore **≈ +1, 17**.

Mais quelle est la vraie valeur ? La réponse est consternante...voire inquiétante : aucune des valeurs précédentes. La vraie valeur est : **−0,82739605994 6821368141165095479816 29**.

Ainsi, non seulement aucun système ne permet d'obtenir la vraie valeur mais en plus, les résultats sont en totale contradiction que ce soit sur l'ordre de grandeur ou le signe.

2 Représentation des réels en machine

Nous allons maintenant expliquer comment sont stockées les valeurs réelles dans les ordinateurs afin de commencer à comprendre d'où viennent ces erreurs de calcul.

2.1 Rappel sur les bases

Nous sommes évidemment familiers avec la base **10** mais ce que nous faisons quotidiennement avec cette base peut s'étendre à n'importe quelle autre. Ainsi, en base **b** tout nombre décimal **d** peut s'écrire avec **m+n+1** chiffres (digits) de la façon suivante : $d_m d_{m-1} \ldots d_1 d_0, d_{-1} \ldots d_{-n}$

Les chiffres d'indice positif (ou nul) représentent les chiffres à gauche de la virgule et ceux d'indice négatif les chiffres à droite de la virgule. La valeur de **d** se calcule alors selon la formule :

$$d = \sum_{i=-n}^{m} d_i.b^i$$

Note

Par convention, et quand cela est nécessaire, nous noterons la base en indice des nombres ainsi représentés. Par exemple **52,34**$_{10}$ est une représentation en base **10** alors que **101,11**$_2$ utilise la base **2**.

Si l'on revient en base **10**, le nombre **52,34**$_{10}$ est bien égal à

$$\sum_{i=-2}^{1} d_i.10^i = 5.10^1 + 2.10^0 + 3.10^{-1} + 4.10^{-2}$$

soit : $(5.10^3 + 2.10^2 + 3.10^1 + 4).10^{-2}$

Il peut aussi se représenter sous une forme fractionnaire : $\frac{5234}{100}$

Nous allons nous intéresser plus particulièrement à la base **2**, utilisée par les ordinateurs.

De la même façon, la valeur (en base **10**) du nombre dont l'écriture en base **2** est **101,11**$_2$ se calcule ainsi :

$$\sum_{i=-2}^{2} d_i.2^i = 1.2^2 + 0.2^1 + 1.2^0 + 1.2^{-1} + 1.2^{-2}$$

soit

$$(1.2^4 + 0.2^3 + 1.2^2 + 1.2^1 + 1.2^0).2^{-2}$$

Sous sa forme fractionnaire il s'écrit :

$$\frac{23}{4}$$

Conséquences

Ayant formalisé les écritures des nombres en base **b**, on en déduit qu'on ne peut représenter exactement que les nombres fractionnaires de la forme :

$$\frac{X}{b^k}$$

où **X** et **k** sont des entiers .

Ainsi, en base **10**, on ne peut représenter que les nombres de la forme

$$\frac{X}{10^k}$$

et en base 2 que les nombres de la forme

$$\frac{X}{2^k}$$

Certains nombres peuvent être représentés dans une base mais pas dans une autre.

La représentation de

$$\frac{1}{3}$$

en base **10** n'est pas exacte, elle s'écrit : **0,[3]**$_{10}$. De même pour celle de

$$\frac{1}{10}$$

qui, en base **2**, s'écrit **0,000[1100]**$_2$.

Note

Quel est le sens de l'écriture avec des [ ] ? La suite de chiffres entre crochets est répétée indéfiniment. Ainsi $0,000[1100]_2$ désigne la suite (infinie) : 0,000110011001100110011001100...

On commence à comprendre pourquoi la somme de 1000 fois 0,1 ne donne pas exactement 100 sur ordinateur. Mais ce n'est pas la seule raison ! Nous l'expliquerons plus en détail dans les prochaines pages.

2.2 Représentation des réels en virgule flottante

La représentation d'un réel en virgule flottante signifie que (contrairement à une représentation en virgule fixe), on peut placer la virgule où l'on souhaite, à un facteur bk près. Ainsi, $37,5_{10}$ peut s'écrire : 37500.10^{-3} ; $0,0375.10^{3}$; $0,375.10^{2}$; ...

Par convention, la représentation normalisée (unique) en virgule flottante d'un réel est définie par les caractéristiques suivantes :

- la partie entière (à gauche de la virgule) est systématiquement nulle ;
- tous les chiffres significatifs sont à droite de la virgule ;
- l'exposant est un entier (positif ou négatif).

Un nombre réel, en représentation normalisée, peut alors être stocké uniquement par sa mantisse (inférieure à 1) et son exposant.

$37,5_{10}$, sous sa forme normalisée ($0,375.10^{2}$), peut être stocké par sa mantisse (0,375) et son exposant (2).

Cette définition de la représentation normalisée est indépendante de la base. Ainsi, de la même manière, la représentation normalisée de 11011 (en base 2) est $(0,11011).2^{101}$. Explication : on a décalé la virgule de cinq chiffres vers la gauche et on a donc besoin de multiplier par 2^5 qui s'écrit bien évidemment 2^{101} en base 2 !

Il est intéressant de noter (et on y reviendra dans quelques instants) que tous les chiffres significatifs étant à droite de la virgule, le premier chiffre après la virgule ne peut donc pas être un 0. En base 2, ce sera donc forcément un 1.

2.3 Représentation des réels sur ordinateur

Un nombre réel est représenté sur ordinateur par un triplet (voir figure 1) :

- le signe (stocké sur 1 bit) : par convention, il sera négatif si ce bit vaut 1, positif sinon ;
- l'exposant (stocké sur p bits) . Pour des raisons pratiques, on ne stocke pas la vraie valeur de l'exposant mais l'exposant décalé c'est-à-dire un nombre entier compris entre 0 et 2^{e-1} plutôt qu'un nombre compris entre $-2^{e-1}+1$ et 2^{e-1}. Le décalage (ou biais) est égal à $2^{e-1}-1$.Pour une valeur de e=8, l'exposant décalé sera donc compris entre 0 et 255 plutôt qu'entre -127 et 128 et le décalage sera de 127.
- la mantisse (stockée sur m bits) représente un nombre réel compris entre 0 et 1. Comme nous l'avons vu plus haut, le premier bit de la mantisse étant toujours 1 (ceci est valable uniquement pour la base 2), nous pouvons omettre de le stocker pour gagner un bit supplémentaire de précision ; il sera appelé bit implicite. La conséquence est que la mantisse réelle sera égale à 1 + mantisse stockée. La valeur de l'exposant sera également adaptée (on retranche 1) pour représenter le même nombre.

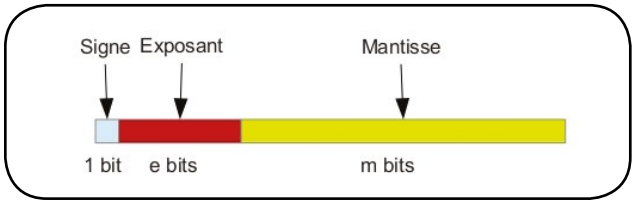


Fig. 1 : Représentation d'un nombre réel sous sa forme signe – exposant - mantisse.

La norme IEEE-754 (sur laquelle nous reviendrons plus en détail) fixe la valeur de e et m pour les réels simple et double précision selon le tableau ci-dessous :

Type	Signe (nombre de bits)	Exposant (nombre de bits)	Mantisse (nombre de bits)
Simple précision	1	8	23
Double précision	1	11	52

De manière générale, la valeur d'un entier stocké sur 1+e+m bits en base 2 se calcule selon la formule

$$s.2^{e^*}.m^*$$

avec :

- s=-1 ou 1 ;
- e^* : l'exposant réel (exposant décalé – décalage) ;

- m^* : la mantisse réelle (1 + mantisse stockée).

La valeur de ce réel s'obtient donc (avec les valeurs stockées) selon la formule

$$s.2^{e-decalage}.(1+m)$$

Note

La formule permettant de calculer la valeur d'un nombre à partir de sa représentation n'est pas valable pour tous les nombres réels. Il existe des exceptions pour représenter des réels de très petites valeurs absolues (y compris le 0), de $-\infty$, de $+\infty$ et des NaN (*Not a Number*). Nous ne détaillerons pas ici la représentation de ces nombres stockés sous une forme dite dénormalisée.

Je crois qu'il est temps de donner quelques exemples concrets pour s'assurer que tout le monde suit...

Exemple 1 : En simple précision, la valeur 0,5 est représentée par la suite des 32 bits suivants :

0 01111110 00000000000000000000000.

En effet :

- le bit de signe est 0 : le nombre est positif ;
- l'exposant réel est égal à la valeur de l'exposant stocké (01111110) – le décalage soit $2^6 + 2^5 + 2^4 + 2^3 + 2^2 + 2^1 - 127 = 126 - 127 = -1$;
- la mantisse réelle est égale à 1 + la mantisse stockée (une suite de 0) soit 1 + 0 = 1.

Ainsi, on obtient $1.2^{-1} = 0,5$.

Exemple 2 : Représentations de 0,1.

Comme nous l'avons vu, ce nombre ne peut pas être représenté en base 2, il existe donc une représentation de sa valeur par excès et une par défaut.

En simple précision, la suite 0 01111011 10011001100110011001101 donne la valeur de 0,1 par excès. En effet :

- le bit de signe est 0 : le nombre est positif ;
- l'exposant réel est égal à la valeur de l'exposant stocké (01111011) – le décalage soit $2^6 + 2^5 + 2^4 + 2^3 + 2^1 + 2^0 - 127 = 123 - 127 = -4$;
- la mantisse réelle est égale à 1 + la mantisse stockée (10011001100110011001101) soit $1 + 2^{-1} + 2^{-4} + 2^{-5} + ... = 1,600000023841858$.

Ainsi, on obtient $1,600000023841858.2^{-4} = 0,100000001490116$.

La valeur de 0,1 par défaut est représentée par 0 01111011 10011001100110011001100. Il s'agit de la valeur juste inférieure à la valeur précédente. Pour l'obtenir, on a juste remplacé le dernier bit de la mantisse, celui de droite, appelé bit de poids faible (qui valait 1) par 0. Le calcul mène à $1,5999999046325684.2^{-4}$ soit 0,09999999403953552.

Conclusion

Vous savez désormais que les réels flottants (ceux de nos ordinateurs) ne représentent qu'un sous-ensemble des vrais réels. On comprend alors que les vraies valeurs ne peuvent pas, dans la grande majorité des cas, être obtenues par les ordinateurs qui ne travaillent que sur ce sous-ensemble.

Ayant défini la représentation des réels flottants, nous sommes maintenant prêts à comprendre les mécanismes mis en œuvre lors des calculs. C'est ce que nous ferons dans la deuxième partie de cet article (le mois prochain) en détaillant deux opérations (l'addition et la soustraction) et en expliquant ce qu'il se passe pour les autres opérations. Vous n'êtes pas au bout de vos surprises... ■

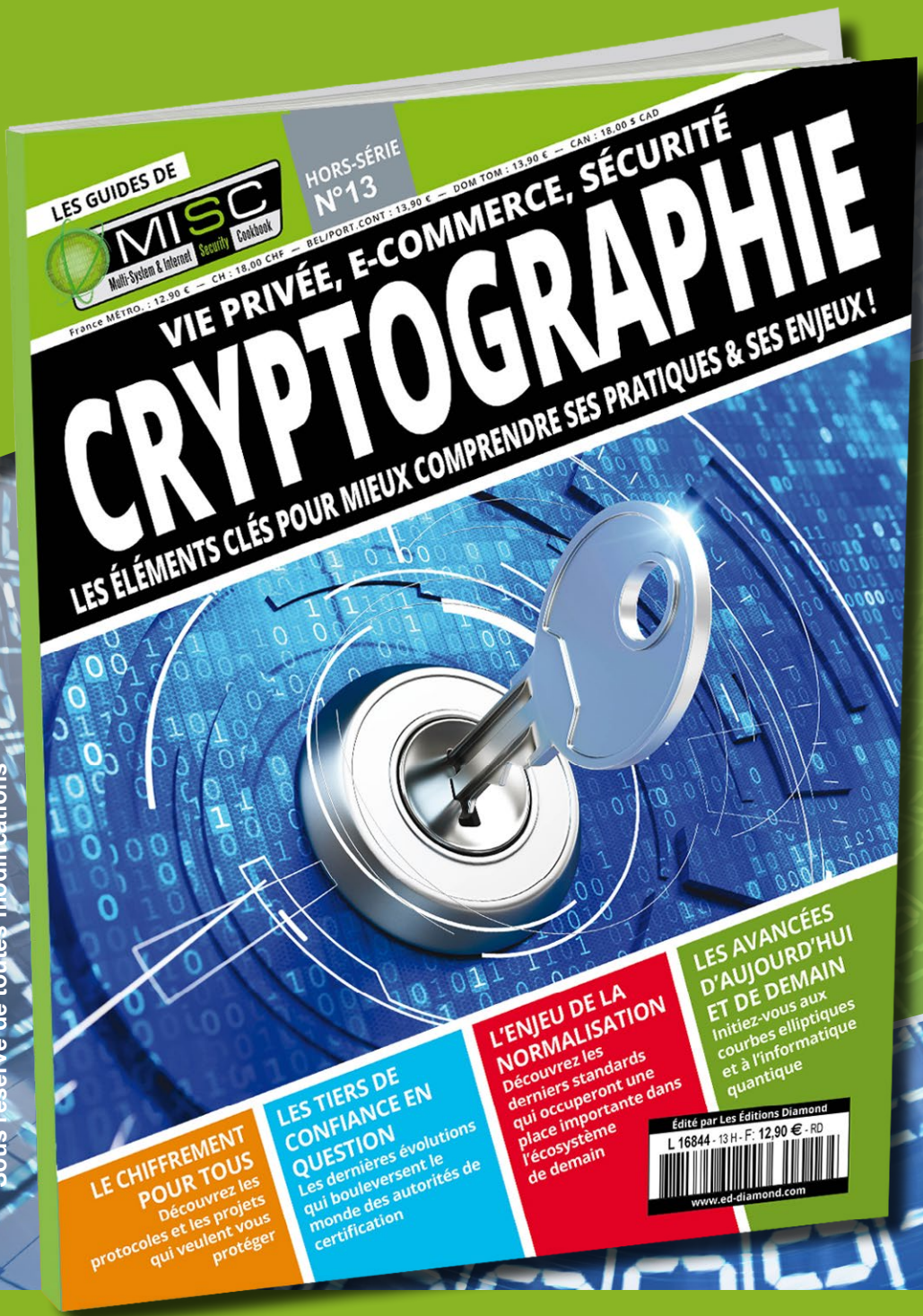
Pour aller plus loin

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, p. 51 à 65) revient sur le traitement de cette thématique lors des JDEV (Journées du DEVeloppement 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.

ACTUELLEMENT DISPONIBLE

MISC HORS-SÉRIE N°13 !



LE GUIDE POUR COMPRENDRE

LES PRATIQUES ET LES ENJEUX ACTUELS

DE LA CRYPTOGRAPHIE !

NE LE MANQUEZ PAS

CHEZ VOTRE MARCHAND DE JOURNAUX ET SUR :

www.ed-diamond.com


EXPLIQUER UN CODE QR

Nicolas Patrois [Enseignant dans le secondaire]

Vous en avez assez de voir des codes QR sans comprendre leur fonctionnement ? Voici comment lire leur contenu, qu'il soit entaché d'erreurs ou non et comment en créer.

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

Résumé

Cette série de huit articles présente une étude presque complète des codes QR et pour commencer, une présentation du projet suivie d'une description du format et des premières méthodes de notre classe. Les deux articles suivants déchiffreront l'image en un tableau de bits puis en liront toutes les informations nécessaires à son décodage, c'est-à-dire écrire le message caché dans le magma de carrés noirs et blancs. Toutes les informations de format pourront également être affichées. Les codes QR entachés d'erreurs ou munis d'un petit dessin verront leur contenu corrigé à l'aide de codes de Reed-Solomon (et une brève étude d'un corps fini ainsi que son codage en Python seront nécessaires). Enfin, tout ceci nous servira pour créer un code QR fonctionnel.

On voit de plus en plus de carrés pourvus de petites cibles dans les coins, remplis de carrés noirs et blancs disposés presque aléatoirement. Il ne s'agit pas d'œuvres d'artistes de rue qui veulent remplacer les populaires petits *space invaders* en céramique mais, et c'est moins joli, de codes QR, destinés à remplacer les codes à barres. Ils contiennent une vraie correction d'erreur contrairement à ces derniers et permettent de stocker des données plus variées et en plus grande quantité. Ils sont protégés par des brevets **[1]** donc cet article est à usage non professionnel, seulement pour *geeker* entre amis. Ces codes se limitent à traduire une image carrée faite de carrés, pas à reconnaître un code QR vu en perspective.

1 Préliminaires

1.1 Présentation du projet

Huit articles sont prévus en trois parties, trois articles sur la lecture, trois sur la correction et deux sur la création :

- le premier présente le projet, le format du code QR et l'affichage en console ;
- le deuxième déchiffre l'image de la figure 1 qui ne contient pas d'erreur en vue ;
- le troisième interprète le contenu binaire pour afficher le message ;
- le quatrième présente les mathématiques des corps finis ;
- le cinquième construit la classe mathématique utilisée dans la correction de Reed-Solomon ;
- le sixième détaille le mécanisme de correction des erreurs ;
- le septième prépare les données à remplir ;
- le dernier présente comment écrire un code QR.

Nous ne nous intéresserons qu'aux codes QR de modèle 2 dits standards, pas au micro QR ni au modèle 1, plus ancien et abandonné.

Fig. 1 : Le code QR à décoder.

Ce premier article commence par présenter brièvement le format.

1.2 Organisation du code

Le code décortiqué dans cet article est organisé en cinq fichiers non nettoyés en Python 3 : vous bénéficierez de code mort et de tonnes de codes ratés en commentaires, avec les images et exemples sur le GitHub du magazine **[2]** :

- **qrdecode.py**, le code principal qui affiche le contenu de l'image du code QR, éventuellement décoré ou avec des erreurs ;
- **qrencode.py**, le code principal qui crée une image du code QR dont le contenu est un fichier ;
- **qrcodeoutils.py**, une petite bibliothèque d'utilitaires communs aux autres mais non spécifique aux codes QR ;
- **qrcodestandard.py**, une bibliothèque qui contient les fonctions et constantes spécifiques aux codes QR ;
- **qrcorps.py**, une bibliothèque qui permet de manipuler le corps fini $\mathbb{F}_{256}$ et les polynômes à coefficients dans ce corps.

L'organigramme est présenté en figure 2.

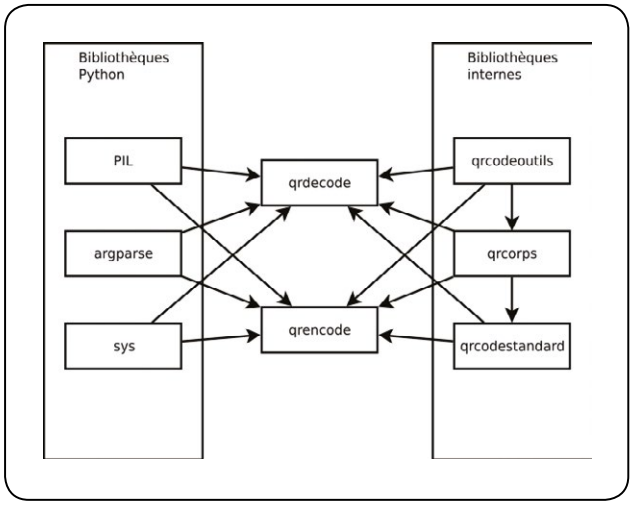


Fig. 2 : L'organigramme des bibliothèques utilisées.

Par exemple, **qrcorps.py** nécessite seulement **qrcodeoutils.py** et est utilisé par **qrdecode.py**.

Enfin, **qrdecode.py** et **qrencode.py** ont besoin de la bibliothèque d'images **PIL** **[3]**, d'**argparse** pour le traitement des options de la ligne de commande et de **sys** pour **exit** et **sdterr**.

PIL s'installe avec le paquet **python3-pil** de Debian, **argparse** avec **python-argparse** (plutôt pour Python 2 mais fonctionne très bien ici) et **sys** est fourni d'office.

Je me suis principalement servi des deux articles parus sur wikiversity **[4]**, parfois corrigés silencieusement ici, les autres sources sont données dans le texte.

2 Description du format

La description en détail du format se fera pendant l'explication du code en Python, il est cependant utile d'avoir un aperçu du vocabulaire employé et de voir globalement où sont les données utiles au décodage.

2.1 Vocabulaire

Un code QR est un carré formé d'un contour blanc de quatre modules d'épaisseur et d'un grand carré central qui contient les données codées sous forme de petits carrés noirs et blancs — les modules — plus quelques zones vierges de données pour calibrer la reconnaissance de l'image, les cibles et deux segments pointillés.

La version d'un code QR est, en gros, sa taille : un code de version 1 a 21 modules de côté (sans le contour) et 4 modules supplémentaires augmentent la version d'une unité.

Le format contient le niveau de correction des erreurs (on peut corriger plus ou moins d'erreurs) et le masque qui sert à éviter des motifs trompeurs dans l'image (par exemple des carrés concentriques où il ne faudrait pas ou des zones monochromes trop grandes). On en détaillera le principe lors de la création d'un code QR.

Enfin, le type désigne le type de données que le code contient parmi les quatre suivants :

- un format numérique ;
- un format alphanumérique qui contient 45 symboles ;
- un format binaire qui peut contenir du texte ASCII ou non mais aussi, soyons fous, un exécutable (s'il est bref sinon sur plusieurs codes QR consécutifs, chose possible) ;
- un format de texte kanji (non codé dans cet article) puisqu'il vient du Japon.

Les formats sont compressés sauf le format binaire.

2.2 Différentes zones

Les différentes zones d'un code QR de version 3 (donc de 29×29 modules) sont illustrées dans la figure 3, page suivante.

Un module noir vaut **1** et sera représenté par une couleur pure ; un module blanc vaut **0** et sera représenté par une couleur pastel. Les couleurs d'une même teinte sont de la même zone.

En rouge vif et rose, les cibles pour la reconnaissance et le placement correct de l'image plus un module nécessairement noir au-dessus de la barre verticale verte inférieure. Seul un code QR de version 1 n'a pas la petite cible en bas à droite, en revanche, un code QR de version 7 ou plus a des petites cibles complémentaires, réparties régulièrement dans l'image plus deux zones de vérification de la version de 3×6 modules chacune.

Fig. 3 : Les quatre différentes zones d'un code QR.

En bleu, les deux zones pointillées qui servent aussi au calibrage, par exemple à la détermination des dimensions des modules.

En vert pomme, les deux zones de format concaténées avec leur correction d'erreur : elle est présente d'une part autour de la cible en haut à gauche en tournant dans le sens trigonométrique, d'autre part en partant du bas et en remontant sur la cible à droite en tournant dans le sens horaire. Regardez bien et observez les deux endroits distincts où est écrite la suite de bits **111110110101010** (et comment). Si vous restez perché comme un guitariste psychédélique après l'avoir lue de tête, vous êtes mûr pour la partie grisée sans les articles suivants.

Enfin, en gris et blanc, la zone de données proprement dite qui contient un court en-tête (dont sa longueur exacte et le type de données), le message et sa correction. Ces données se lisent à partir du coin en bas à droite en suivant un petit serpent qui zigzague que nous repréciserons deux articles plus loin.

3 Construction du tableau de bits

Mon premier code était un pur code impératif qui nécessitait un suivi minutieux des données d'une fonction à la suivante. Bref, c'était laid et peu pratique. Allez, je me lance, j'essaie la programmation orientée objet à l'acronyme si amusant en anglais **[5]**.

J'ai finalement utilisé une unique classe **qrdecode** dont un membre de la classe — une instance — **essai**, contient tous les éléments — ses attributs — du nom du fichier au contenu textuel du message. Chaque fonction — ou méthode — est une étape dans le décodage, de la lecture du fichier à la production du message. Elles s'appellent les unes les autres, à la manière de poupées russes si un attribut (ou donnée) de l'instance n'est pas défini (c'est-à-dire s'il vaut **None**) au début de l'exécution de la méthode qui en a besoin. Bien entendu, à des fins de jeu ou de débogage, on peut toujours remplir soi-même ces attributs au sein du code.

Après l'initialisation de l'instance et son affichage, les méthodes transforment progressivement l'image brute en un tableau de bits : chargement et en passant recadrage si nécessaire, puis recherche des dimensions d'un module. Enfin, l'image est tournée si elle est mal orientée. Ce paragraphe détaille le contenu de l'article suivant.

3.1. Initialisation

Le code sera expliqué méthode par méthode en commençant par l'initialisation. Par ailleurs, il sera renuméroté à partir de **01** à chaque section.

```
01: #!/usr/bin/python3

03: from PIL import Image
04: from sys import exit,stderr
05: from argparse import *
06: from qrcorps import *
07: from qrcodestandard import *
08: from qrcodeoutils import *

10: class qrdecode:
11:   def __init__(self):
12:     [self.fichier,self.corrige,self.tout,self.image,self.module,self.qr,
13:       self.esttourne,self.form,self.formatok,self.nivcor,self.
masque,self.dim,
14:       self.version,self.gris,self.code,self.clair,self.redondant,self.
messageok,
15:       self.mode,self.longueur,self.longclair,self.message] = [None]*22
```

Il s'agit du constructeur d'une instance de la classe **qrdecode.py** et pour le moment, chaque attribut vaut **None**. Les noms des attributs sont presque tous parlants par eux-mêmes, ils sont présentés dans l'ordre d'apparition dans le code. Pour des commentaires plus lisibles, j'omettrai systématiquement le préfixe **self.** dans les noms des attributs et des méthodes.

3.2 Affichage

Avant de s'intéresser aux méthodes particulières à la classe (voir article suivant), commençons par la sortie de notre programme et la méthode **__str__**.

```
01:   def __str__(self):
```

Si on décide d'afficher toutes les informations du code :

```
02:     if self.tout:
03:       ch=""
04:       if self.fichier is not None:
05:         ch=ch+"Fichier : "+self.fichier+"\n"
06:       if self.formatok is not None and self.formatok is not True:
07:         ch=ch+"Il y a eu besoin de corriger la zone de format.\n"
08:       if self.nivcor is not None:
09:         ch=ch+"Niveau de correction : "+nomnivcor[self.nivcor]+"\n"
```

nomnivcor (dans **qrcodestandard.py**) contient les noms longs des niveaux de correction :

```
nomnivcor={"L":"Low","M":"Medium","Q":"Quality","H":"High"}
```

On dessine le masque utilisé :

```
10:       if self.masque is not None:
11:         ch=ch+"Masque :\n"
12:         ch=ch+dessine([[self.masque(i,j) for j in range(6)]
for i in range(6)])
```

La fonction **dessine** est dans **qrcodeoutils.py** (voir à la fin de cette section).

Le petit bout de code suivant permet d'afficher les huit masques :

```
for m in sorted(masques):
  print(m)
  print(dessine([[masques[m](j,i) for i in range(6)] for j in range(6)]))
```

Par exemple, le masque **(0,0,1)** inverse les lignes de rang (pythonique) pair alors que **(0,1,1)** inverse une diagonale sur trois.

```
13:       if self.dim is not None:
14:         ch=ch+"Dimensions : "+str(self.dim)+"x"+str(self.dim)+"\n"
15:       if self.version is not None:
16:         ch=ch+"Version : "+str(self.version)+"\n"
17:       if self.messageok is not None and self.messageok is not True:
18:         ch=ch+"Il y a eu besoin de corriger %d erreur(s) dans la
zone de données.\n"%self.messageok
19:       if self.mode is not None:
20:         ch=ch+"Mode : "+modes[self.mode]+"\n"
```

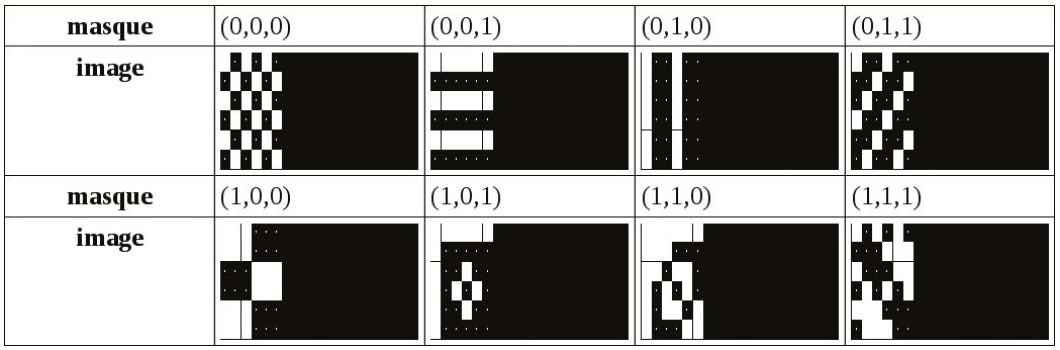
modes est dans **qrcodestandard.py** avec la définition suivante :

```
modes={0:"Numeric",1:"Alphanumeric",2:"Byte",3:"Kanji"}
```

Et enfin :

```
21:       if self.longclair is not None:
22:         ch=ch+"Longueur du message : "+str(self.longclair)+"\n"
23:       if self.message is not None:
24:         ch=ch+"Message :\n"+self.message+"\n"

26:       return ch[:-1]
```



Si on choisit de tout afficher avec l'option **-a 1**, tout ce qui est utile et défini (donc distinct de **None**) est ajouté à la queue leu leu (tout le monde s'amuse) dans la chaîne de caractères **ch**, qui est retournée ligne 26 privée du dernier retour à la ligne. Eh oui, sinon il y a un retour à la ligne en trop et c'est laid.

Si on choisit de n'afficher que le contenu :

```
28:     else:
29:       return self.message
```

Cette méthode **doit** retourner une chaîne et non l'afficher.

Note

Il peut être utile de taper **dir(2)**, **dir([2])** ou **dir({2})** dans une console Python pour regarder différentes méthodes entourées de doubles _. Ainsi, **__str__** sert par exemple aux fonctions **print** et **str**. **__add__** (qui n'est pas utile ici) sert à ajouter une instance à gauche de quelque chose (à gérer dans le code de la méthode) en utilisant tout simplement l'opérateur **+**. Ceci nous servira plus loin quand nous créerons de toutes pièces les opérations dans le corps fini $\mathbb{F}_{256}$, en clair pour prolonger **2*3-4** avec nos futurs nombres bizarres.

Voici le détail de la fonction **dessine** :

```
def dessine(tableau):
  dic={0:"·",1:"█",False:"·",True:"█"}
  c=""
  for l in tableau:
    c=c+"".join(dic[i] for i in l)+"\n"
  return c
```

Elle sert à afficher le masque de décodage (voir le troisième article de cette série) en noir et blanc dans la console (**True** ou **1** par un rectangle noir ■ et **False** ou **0** par le petit point ·) et pourrait aussi servir à dessiner le code QR. La méthode **join** du type **str** permet de concaténer directement les éléments d'un type itérable (liste, ensemble...) si ce sont des chaînes. Le séparateur est ici vide mais il peut être n'importe quelle autre chaîne comme une espace ou un tiret.

```
31:   def __repr__(self):
32:     if self.message is not None:
33:       return self.message
34:     return "Rien n'est défini."
```

La méthode **__repr__**, au cas où, ne retourne que le **message**.

Conclusion

Le décor est planté, l'article suivant va transformer l'image de notre code QR en un tableau de bits dans le but de finalement lire effectivement son contenu. ■

Références

[1] Les codes QR sont protégés par neuf brevets au Japon et au États-Unis : http://www.qrcode.com/en/patent.html.

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

[3] La bibliothèque PIL est ici : http://effbot.org/imagingbook/.

[4] 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/.

[5] *Also sprach Winnie the POO...*

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

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,-

Prix TTC en Euros / France Métropolitaine

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

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,-

Prix TTC en Euros / France Métropolitaine

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 :

.....

.....

DÉCHIFFRER UN CODE QR

Nicolas Patrois [Enseignant dans le secondaire]

Après les présentations de l'article précédent, on continue notre voyage en lisant l'image du code QR pour la traduire en un tableau de 0 et de 1.

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

Résumé

Dans cet article, on transformera l'image d'un code QR sans erreur ni joli petit dessin en un tableau de bits. Pour cela, on doit déterminer la taille d'un module, la position des cibles et éventuellement l'orientation de l'image.

Avant de lire l'image, un petit rappel s'impose. Nous allons étudier le code QR de la figure 1, le même que celui de l'article précédent. J'appelle module un petit carré blanc ou noir, cible les motifs carrés des coins, elles nous serviront pour déterminer la taille en pixels d'un module et l'orientation de l'image.

Fig. 1 : Le code QR à décoder.

Le cadre est clair, on peut commencer à lire l'image fournie en entrée.

1 Chargement et recadrage de l'image

Nous y voilà, la première méthode charge et stocke les pixels de l'image dans l'attribut **image** à partir de l'attribut **fichier** qui contient — tadaaa — le nom du fichier. Elle gère aussi les options de la ligne de commande. Je renuméroterai le code à partir de 01 à chaque section.

```
01:  def chargeim(self):
02:      parser=ArgumentParser(description="Lit et corrige un code QR.")
03:      parser.add_argument("-i",required=True,metavar="image",
help="Image d'entrée.")
04:      parser.add_argument("-c",choices=["1","0"],default="1",
help="Corrige ou non les erreurs.")
05:      parser.add_argument("-a",choices=["1","0"],default="0",
help="On affiche tout ou seulement le message.")
06:      arguments=parser.parse_args()
07:      self.fichier=arguments.i
08:      self.corrige=int(arguments.c)
09:      self.tout=int(arguments.a)
```

argparse est une bibliothèque qui permet de manipuler simplement les arguments de la ligne de commande sans réinventer la roue. Ici :

- **-i** spécifie l'image utilisée en entrée, de format bitmap ;
- **-c** (dés)active la correction des erreurs ;
- **-a** affiche la totalité des informations ou seulement le contenu du code QR.

Voici ce que donne l'option **-h**, automatiquement fournie par la bibliothèque :

```
> ./qrdecode.py -h
usage: qrdecode.py [-h] -i image [-c {1,0}] [-a {1,0}]

Lit et corrige un code QR.

optional arguments:
```

```
  -h, --help  show this help message and exit
  -i image    Image d'entrée.
  -c {1,0}    Corrige ou non les erreurs (peut planter si non).
  -a {1,0}    On affiche tout ou seulement le message.
```

Et sans option :

```
> ./qrdecode.py
usage: qrdecode.py [-h] -i image [-c {1,0}] [-a {1,0}]
qrdecode.py: error: the following arguments are required: -i
```

La suite du code :

```
11:     try:
12:       im=Image.open(self.fichier)
13:     except IOError:
14:       print("Fichier "+self.fichier+" inexistant.",file=stderr)
15:       exit(1)
```

On essaie d'ouvrir l'image, en cas d'échec on sort en se plaignant sur la sortie d'erreur.

```
17:     im=im.convert('1')
18:     pix=im.load()
19:     self.image=[]
```

On convertit l'image en noir et blanc à l'aide de la bibliothèque de traitement d'images **PIL** — sans John Lydon.

```
17:     for i in range(im.height):
18:       self.image.append([0+(pix[j,i]==0) for j in range(im.width)])
```

On stocke l'image dans l'attribut **image** qui est un tableau de tableaux puis, à la fin de la méthode **chargeim**, au cas où elle aurait été mal cadrée, on la recadre pour que le contour n'ait que deux pixels d'épaisseur. Python convertit automatiquement la valeur **True** à **1** et **False** à **0** en cas d'opération entre un booléen et un entier donc si un pixel est noir, la valeur de l'expression **0+(pix[j,i]==0)** sera **1**. On peut remarquer que les coordonnées dans l'objet **pix** sont écrites de manière mathématique et non de manière pythonique.

```
20:     while sum(self.image[0])==0:
21:       self.image=self.image[1:]
22:     while sum(self.image[-1])==0:
23:       self.image=self.image[:-1]
24:     self.image=[[0]*len(self.image[0])]*2+self.
image+[[0]*len(self.image[0])]*2
```

On retaille le haut puis le bas avant d'ajouter deux lignes blanches en haut puis en bas.

```
25:     while True:
26:       s=0
27:       for l in self.image:
28:         s+=l[0]
29:       if s==0:
30:         for i in range(len(self.image)):
31:           self.image[i]=self.image[i][1:]
32:       else:
33:         break
34:     while True:
35:       s=0
36:       for l in self.image:
37:         s+=l[-1]
38:       if s==0:
39:         for i in range(len(self.image)):
40:           self.image[i]=self.image[i][:-1]
41:       else:
42:         break
43:     for i in range(len(self.image)):
44:       self.image[i]=[0,0]+self.image[i]+[0,0]
```

Et de même avec les colonnes de gauche ligne 31 où on enlève le premier élément blanc de chaque ligne (si la colonne est entièrement blanche) puis de même avec le dernier blanc ligne 40 avant d'ajouter deux colonnes blanches de chaque côté.

2 Recherche de la dimension d'un module

2.1 Méthode directe

On n'utilise pas les petits pointillés bleus d'un module d'épaisseur (voir figure 2 pour rappel), on va reconnaître le motif blanc-noir-blanc à partir du coin supérieur gauche et du coin opposé. En effet, au moins l'un des deux contient une cible. Bien entendu, si l'attribut **image** est vide (un démon de minuit ?), on appelle la méthode précédente.

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

```
01:   def dims(self):
02:     if self.image is None:
03:       self.chargeim()
04:     i=0
05:     etat0,etat1=0,0
06:     no0,no1=[],[]
```

On va reconnaître l'expression rationnelle **0+1+0** (au moins un **0** suivi d'au moins un **1** et d'un **0** pour finir, voir figure 3) pour laquelle une cible a une dimension minimale. Pour cela il faut détecter deux changements d'état : une fois pour passer de **0** à **1** et une fois pour revenir à **0**. Par exemple pour le coin supérieur gauche, ligne 09, si la valeur courante **image[i][i]** est différente de la valeur précédente **etat0**, on stocke la coordonnée du changement dans la liste **n0** et on modifie l'état courant jusqu'à avoir repéré deux changements (ie **n0** ou **n1** a deux éléments).

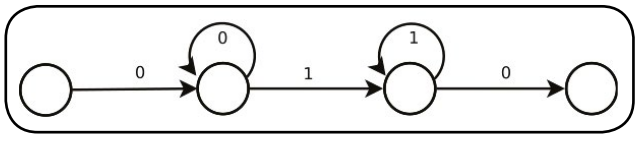


Fig. 3 : Notre petite machine qui reconnaît le motif.

```
08:     while not(len(no0)==2 or len(no1)==2):
09:       if self.image[i][i]!=etat0:
10:         etat0=self.image[i][i]
11:         no0.append(i)
12:       if self.image[-i-1][-i-1]!=etat1:
13:         etat1=self.image[-i-1][-i-1]
14:         no1.append(i)
14:       i+=1

16:     if len(no0)<len(no1):
17:       self.module=no1[0]-no1[1]
18:     else:
19:       self.module=no0[1]-no0[0]
```

Le premier entre **n0** et **n1** qui a une longueur de 2 a rencontré une cible, c'est donc lui qui va nous indiquer la dimension d'un module. En fait, ligne 16, on teste si la cible est en bas à droite.

Ici, un **module** fait 6×6 pixels.

2.2 Graphe étiqueté

On peut aussi utiliser un petit parcours de graphe étiqueté :

```
01:   def dims(self):
02:     if self.image is None:
03:       self.chargeim()

05:     graphe=[[None,0,None,None],[None,0,1,None],[None,None,1,
0],[None]*4]
06:     motif=[self.image[i][i] for i in range(len(self.image))]
07:     _,coord1=regraph(graphe,motif,0,3)
08:     _,coord2=regraph(graphe,motif[::-1],0,3)
09:     coord=min(coord1,coord2)
10:     self.module=coord[2]-coord[1]
```

On reconnaît une matrice de notre graphe à quatre sommets, une ligne correspond à un sommet, la position au sommet où on va et la valeur au caractère reconnu.

Ainsi au sommet **1**, **[None,0,1,None]** signifie que si on a un **0**, on reste sur le sommet **1** mais que si on a un **1**, on passe au sommet **2**. Tout autre caractère fait arrêter la reconnaissance.

On envoie donc la diagonale descendante (**motif**) puis la montante (**motif[::-1]**), **regraph** retourne si le motif a été reconnu entièrement (aucune importance ici) et les positions des modules de chaque changement de couleur. Ligne 09, celle qui reconnaît le plus tôt le premier changement est celle qui permet d'obtenir la taille d'un module.

regraph est la fonction suivante (dans **qrcodeoutils.py**) :

```
def regraph(g,l,deb,fin):
  pos=deb
  coord=[]
  for i in range(len(l)):
    if l[i] in g[pos]:
      npos=g[pos].index(l[i])
      if npos!=pos:
        coord.append(i)
      pos=npos
    elif i!=len(l)-1:
      return False,coord
  return pos==fin,coord
```

On parcourt le motif envoyé (la liste **l**) et on essaie de parcourir le graphe étiqueté **g** selon les valeurs des éléments de **l**. À chaque changement d'état, c'est-à-dire de sommet, on stocke sa coordonnée dans **coord**. Le sommet courant est **npos**, le sommet précédent est **pos**.

3 Le tableau

3.1 Transformation en tableau

Dans cette méthode, on se contente de prendre la valeur du pixel de coordonnées **2+i*module** en ordonnée et **2+j*module** en abscisse pour compléter le tableau. Le début vaut **2** parce qu'on a retaillé ainsi l'image dans la méthode **chargeim**. Si l'image contenait des modules arrondis, il faudrait tester ligne 09 si un module contient par exemple plus de 60% de noir ou étudier les pixels en son centre pour le déclarer noir et revoir la méthode précédente.

```
01:   def stockeim(self):
02:     if self.module is None:
03:       self.dims()
04:     debut,taille=2,self.module
05:     self.qr=[]
06:     for i in range(debut,len(self.image)-debut,taille):
07:       self.qr.append([])
08:       for j in range(debut,len(self.image[0])-debut,taille):
09:         self.qr[-1].append(self.image[i][j])
```

On stocke le tableau dans l'attribut **qr**, il s'agit bien du dessin en noir et blanc du code QR.

3.2 Orientation du tableau

Enfin, il reste à vérifier que le coin sans grande cible est bien en bas à droite du tableau. Si ce n'est pas le cas, on le tourne autant de fois que nécessaire.

```
01:   def placeim(self):
02:     if self.qr is None:
03:       self.stockeim()
```

Comme d'habituuudeuh.

On a besoin du motif des carrés concentriques caractéristique des grandes cibles :

```
cible=[[1,0,1,1,1,0,1] for _ in range(7)]
cible[1][2:5]=[0]*3
cible[5][2:5]=[0]*3
cible[0]=[1]*7
cible[6]=[1]*7
```

Ce code est dans **qrcodestandard.py**.

```
11:     for i in range(7):
12:       if self.qr[i][:7]!=cible[i]:
13:         coin=2
14:         break
15:       if self.qr[-i-1][:7]!=cible[i]:
16:         coin=1
17:         break
18:       if self.qr[i][-7:]!=cible[i]:
19:         coin=3
20:         break
21:       if self.qr[-i-1][-7:]!=cible[i]:
22:         coin=0
23:         break
```

On teste si, dans un des coins, un segment horizontal de sept modules ne correspond pas au segment de la cible attendu. Le numéro du **coin** est le nombre de quarts de tour directs nécessaire pour orienter correctement le tableau. Si le travail de création de l'image est bien fait, un seul ~~canard~~ coin doit se plaindre et comme je collectionne des canards (vivants)... plus ils sont contents, plus je le suis.

La disposition des coins est donc la suivante : $\begin{matrix} 2 & 3 \\ 1 & 0 \end{matrix}$.

```
def tourner90(matrice):
  nl=len(matrice)
  nc=len(matrice[0])
  m=[[0 for _ in range(nl)] for _ in range(nc)]
  for i in range(nl):
    for j in range(nc):
      m[j][i]=matrice[i][-j-1]
  return m
```

Cette fonction **tourner90**, dans **qrcodeoutils.py**, retourne la matrice envoyée en argument tournée d'un quart de tour direct, où $\begin{matrix} a & b \\ c & d \end{matrix}$ donne $\begin{matrix} b & d \\ a & c \end{matrix}$.

```
25:     for _ in range(coin):
26:       self.qr=tourner90(self.qr)
27:     self.esttourne=True
```

On tourne **coin** fois le tableau (zéro fois éventuellement) et on signale que la matrice est bien orientée.

Conclusion

Le tableau de bits (Jamie Mc Cartney n'y est pour rien) est prêt, il reste à l'interpréter dans le dernier article de cette première partie. ■

METTRE EN PLACE UN BUREAU VIRTUEL SUR SON SERVEUR DÉDIÉ

Bruno Dubois [Linux Power User]

Profitez des tarifs très abordables des serveurs dédiés Linux pour installer un serveur X et un bureau complet accessibles en permanence depuis votre tablette, votre laptop, le PC d'un ami...

Mots-clés : Session HTTP, Tomcat, Stateless Web Tier, Java/J2EE

Résumé

De nos jours, il est de plus en plus facile de se passer d'un « gros ordi » chez soi. En effet, hormis quelques cas spécifiques (les jeux vidéo 3D, les applications de montage vidéo, etc) qui nécessitent un maximum de puissance, les ordinateurs d'entrée de gamme en vente actuellement ont des spécifications techniques bien supérieures à ce dont nous avons besoin pour envoyer un email ou même faire un tableau croisé dynamique dans notre tableur favori. De plus, les nouveaux usages se sont progressivement déplacés sur le *Cloud*. C'est-à-dire que notre *laptop* se transforme petit à petit en un simple terminal qui permet d'accéder à nos données, et même à nos applications, qui sont, elles, sur le net. Ce qui nous amène à nous contenter d'une tablette, ou même d'un smartphone, pour gérer notre vie numérique, car un *laptop* aussi devient trop puissant pour notre besoin.

1 Qui n'a pas son dédié personnel ?

En tant que geek qui se respecte, j'aime bien toujours avoir un petit serveur Linux sous la main. Sur celui-ci, je peux faire tourner mon petit **Apache**, je peux stocker mes fichiers personnels, je peux tester ce que j'ai envie de tester, développer des prototypes d'applications, etc.

Je pourrais installer un petit boîtier sous Linux dans mon salon, avec un disque dur qui héberge mes données et auquel je pourrais accéder depuis n'importe où grâce à l'adresse IP fixe que pourrait me fournir mon opérateur Internet et ma « box » Internet qui renverrait les requêtes HTTP et SSH sur celui-ci.

Mais je vais préférer payer une mensualité ridicule (moins de 10 euros) pour qu'une société professionnelle me fournisse un serveur dédié (dédié : qui n'est pas partagé avec d'autres clients) ou virtuel (les VPS sont encore moins chers) sur lequel je suis root, j'ai des Gio de stockage, de la bande passante, je peux mettre la distribution de mon choix, je suis prévenu en cas de plantage, le disque est remplacé s'il y a un problème, il ne fait pas de bruit dans mon salon, ne consomme pas mon électricité et un cambrioleur ne pourra pas l'emporter avec lui ! C'est propre, c'est beau.

2 Un bureau virtuel sur son dédié

Dans ce monde numérique « cloudifié » où quasiment tout peut être réalisé depuis un Android ou un Chrome (merci Google), il devient compliqué de lancer une bonne

vieille application graphique, surtout si on a besoin de la garder ouverte pendant plusieurs jours.

Grâce à la combinaison de **Xvfb** et **x11vnc**, nous allons voir qu'il est facile de démarrer un environnement graphique complet à distance et profiter à la fois de tout ce qu'apporte **X** (un **Gnome** complet par exemple et pouvoir exécuter n'importe quelle appli graphique), de ce qu'apporte un serveur dédié (bande passante pour télécharger, hébergement professionnel, silence, etc.) et de ce qu'apporte le contrôle à distance (pouvoir piloter tout cela depuis votre smartphone lors d'une soirée IRL en plein cœur de Paris, malgré les déconnexions intempestives).

Voici les programmes principaux qui constituent la base de notre solution :

- Xvfb, une implémentation de serveur X11 sans carte graphique ;
- x11vnc, qui nous permettra de nous connecter au serveur X depuis Internet ;
- **openbox**, un *window manager* léger, indispensable pour gérer les fenêtres ;
- **wine**, l'émul^Wle truc qui permet de lancer des applications windows, si besoin ;
- **screen**, un gestionnaire de console très pratique pour conserver des sessions shell.

3 Xvfb, le serveur X virtuel

X virtual frame buffer, à la différence des autres serveurs X, n'a pas besoin de carte graphique. En fait, il ne saurait que faire d'un tel périphérique, il n'est pas capable d'afficher quoi que ce soit. Si je démarre un xterm dans un Xvfb, le processus va se lancer et fonctionner comme si de rien n'était, mais en réalité il n'existera qu'en mémoire, il n'ira pas plus loin, en tous cas pas sans logiciel supplémentaire.

Xvfb sert généralement pour faire des tests de non-régression d'applications graphiques. Dans ce type de test, on vérifie que le scénario se déroule comme prévu mais personne ne vérifie manuellement le résultat, car ce n'est pas important à ce stade. Par exemple on peut tester le comportement d'une application avec des résolutions improbables.

On peut également utiliser Xvfb pour exécuter des robots. Par exemple un programme de supervision de site Web qui effectue des clics automatisés à des positions précises et procède à des captures d'écrans pour vérifier que le résultat est conforme.

Par exemple, on pourrait parfaitement lancer le robot de Tristan Colombo (GNU/Linux Magazine n°187), qui clique tout seul sur des zones spécifiques du jeu Forge of Empires, dans un Xvfb afin de ne pas monopoliser un écran/clavier/souris.

Bref, c'est un vrai serveur X, mais sans écran.

La subtilité, c'est que nous pouvons utiliser le protocole X11 pour nous connecter à notre serveur X. C'est ainsi qu'on va pouvoir lancer un xterm, un chrome, ou un x11vnc.

Pour installer Xvfb :

```
# apt-get install xvfb
```

Pour démarrer Xvfb :

```
$ Xvfb :0 -nolisten tcp -screen 0 1200x700x16
```

Explication des arguments :

- **:0** est le numéro du serveur X sur le serveur. En effet vous pourrez en lancer plusieurs en changeant ce numéro. Il vous permettra ensuite de lancer une application sur le serveur de votre choix.
- **-nolisten tcp** est une mesure de sécurité qui empêche un intrus d'utiliser notre serveur X à distance. Nous utiliserons VNC pour nous connecter au serveur, qui est plus sécurisé et qui permet de laisser les applications ouvertes lors d'une déconnexion.
- **-screen 0 1200x700x16** définit un écran virtuel et la résolution souhaitée de cet l'écran. Choisissez une résolution qui laisse suffisamment de place à vos applications mais qui rentre aisément dans votre écran physique à vous. Ici **1200x700** est volontairement légèrement inférieur à **1366x768**, la résolution de mon *laptop*.

> **Note**
>
> Xvfb n'a pas besoin des droits root puisqu'il n'interagit pas avec la carte graphique.

Lorsque vous aurez démarré le serveur X, il ne se passera pas grand-chose. Du moins pas grand-chose de visible. En effet, tout est en mémoire et on ne sait pas encore comment y accéder.

4 x11vnc, la passerelle entre VNC et X11

x11vnc, comme son nom et le titre de la section l'indiquent, permet de se connecter à un serveur X via le protocole VNC.

VNC est un protocole spécialisé dans l'accès et le contrôle à distance d'applications graphiques. Il fonctionne

notamment sous Windows, sous X, sous Mac OS X. Il est semblable à *Remote Desktop Protocol* qui est utilisé sous Windows par l'outil d'accès à distance **mstsc**. Pour se connecter à un serveur VNC, il faut... un client VNC.

Grâce à x11vnc, nous allons pouvoir nous connecter à notre bureau à distance, avec un mot de passe, si besoin en SSL, depuis n'importe quel périphérique qui peut exécuter un client VNC. Comme vous vous en doutiez, il y a des clients VNC pour Linux, Windows, Android et même une App Chrome. Peu importe, si vous vous déconnectez, si votre périphérique passe sous un tunnel, s'il reboote, les applications continueront de fonctionner sur le Xvfb de votre dédié. En fait, vous pourriez même arrêter complètement x11vnc que le serveur X ne s'en apercevrait pas.

Pour installer x11vnc :

```
# apt-get install x11vnc
```

Pour démarrer x11vnc :

```
$ x11vnc -usepw -many -rfbport 5900 -display :0
```

Explication des arguments :

- **-usepw** indique que vous souhaitez protéger l'accès par un mot de passe. Au premier démarrage, on vous proposera de le définir ;
- **-many** permet d'être connecté plusieurs fois. Par défaut, un seul client VNC à la fois peut se connecter à x11vnc. En pratique, lors de déconnexions dues au réseau par exemple, x11vnc va mettre un certain temps pour comprendre que vous avez perdu la session. Pendant ce laps de temps la reconnexion vous sera refusée si vous ne spécifiez pas cette option ;
- **-rfbport 5900** indique le port TCP sur lequel cette instance de x11vnc va attendre les clients VNC. **5900** est le port par défaut. Pour des raisons de *sécurité par l'obscurité*, je vous recommande de le modifier ;
- **-display :0** indique sur quel serveur X local x11vnc va se connecter pour en partager l'accès. Puisque vous pouvez lancer plusieurs Xvfb sur le même serveur, vous devez lancer un x11vnc pour chacun d'entre eux que vous souhaitez partager.

Une fois le serveur VNC démarré et le mot de passe positionné, vous pourrez enfin accéder à votre serveur X à distance.

Pour cela, munissez-vous d'un client VNC, par exemple sur votre *laptop* Debian :

```
# apt-get install xvnc4viewer
```

Et connectez-vous à votre serveur dédié :

```
$ vncviewer 17.18.19.21:0
```

Au niveau de la syntaxe de vncviewer, **17.18.19.21:0** indique au client VNC sur quelle adresse IP et quel port il doit se connecter. Par défaut, il y a une correspondance entre le numéro du *display* VNC et le port TCP. Ainsi **:0** correspond au port **5900**, **:1** et **:2** correspondraient aux ports **5901** et **5902** et ainsi de suite. Si vous mettez un port farfelu du type **32145**, vous pouvez simplement indiquer **17.18.19.321:32145**.

Vous voilà connecté à votre nouveau bureau virtuel distant ! Enfin, si tout se passe bien, vous obtiendrez... un grand écran noir. C'est normal, car nous n'avons encore rien démarré.

5 Premiers pas

Pour tester que tout fonctionne bien, nous allons démarrer un **xterm** dans notre nouveau bureau. Normalement le paquetage est déjà installé, mais si ce n'était pas le cas, je vous fais confiance pour savoir comment utiliser la commande **apt-get install xterm**.

Pour l'instant, nous devons démarrer les programmes graphiques dans le shell, en ligne de commande. La variable d'environnement **DISPLAY** permet d'indiquer sur quel écran le programme peut s'afficher. Ainsi, depuis notre serveur dédié :

```
$ export DISPLAY=:0
$ xterm
```

:0 spécifie le numéro du serveur X sur lequel les programmes doivent se lancer.

Si vous avez plusieurs serveurs X et plusieurs programmes à démarrer en tâche de fond, il vous suffit de spécifier le bon **DISPLAY** avant chaque démarrage de commande :

```
$ export DISPLAY=:1
$ xterm &
$ xterm &
$ DISPLAY=:2 xterm &
```

Vous obtiendrez ainsi deux xterms sur **:1** et un xterm sur **:2**.

Depuis votre client VNC, vous devriez voir apparaitre ces terminaux parfaitement utilisables (en fait vous n'en verrez qu'un car ils sont superposés). Vous remarquerez également que vous ne pouvez pas déplacer la fenêtre, ni l'agrandir, ni même la fermer autrement qu'en quittant le terminal (**exit**) ou en terminant le programme (**<Ctrl>** + **<C>** depuis votre premier shell). C'est parce qu'il manque encore un gestionnaire de fenêtre (voir figure 1).

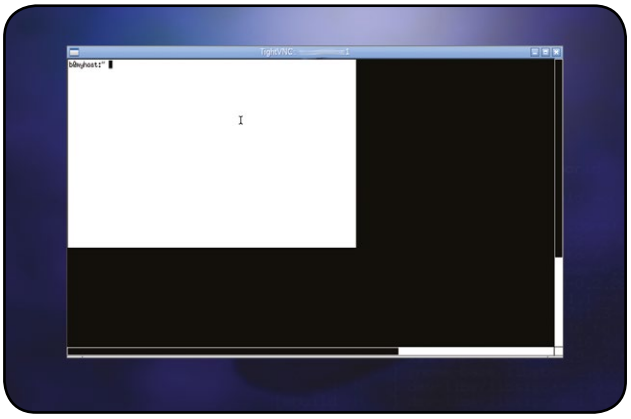


Fig. 1 : Connexion à distance à notre bureau virtuel dans lequel nous avons démarré un xterm.

6 Installer le minimum vital

Alors ici, chacun ses goûts et surtout chacun ses besoins. Pour ma part un gestionnaire de fenêtre léger me suffit amplement :

```
# apt-get install openbox
```

Mais si vous voulez l'artillerie lourde, aucun souci :

```
# apt-get install gnome
```

Ensuite, il ne reste qu'à le démarrer pour rendre votre bureau bien plus agréable :

```
$ export DISPLAY=:0
$ openbox &
```

Immédiatement, vos xterms précédemment démarrés se seront fait coiffer d'une bordure et d'une barre de titre avec ses boutons, qui vous permettront de les déplacer, redimensionner, fermer, etc (voir figure 2). Notez qu'à ce moment-là, vous obtiendrez probablement plusieurs bureaux virtuels que vous pouvez utiliser (ou pas) pour répartir vos applications.

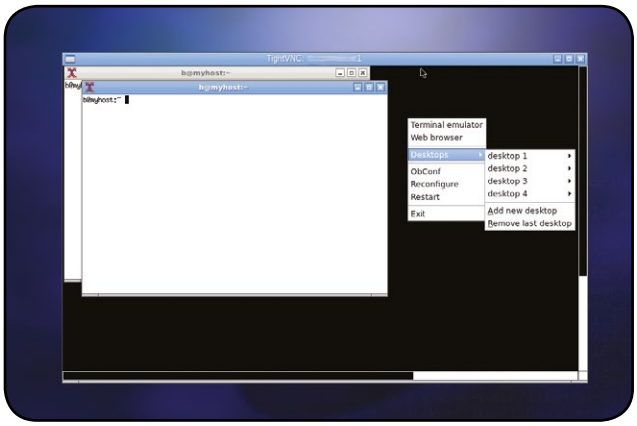


Fig. 2 : Notre bureau virtuel devient utilisable.

7 Vos applications

Vous pouvez enfin installer les applications que vous désirez, que ce soit un **Firefox** pour butiner à distance, un client Bittorrent (attention, c'est interdit chez certains fournisseurs de dédiés), ou même une application Windows avec l'aide de Wine.

Vous pouvez démarrer vos applications directement depuis le bureau X ou bien depuis une session SSH, auquel cas il ne faudra pas oublier de spécifier **export DISPLAY=:0** comme vu précédemment avant de démarrer quoi que ce soit (sinon cela ne marchera pas).

8 Screen pour mieux contrôler tous ces processus

Personnellement, je préfère démarrer le maximum de choses via SSH et voir les logs directement pour corriger de suite si besoin.

Pour cela, j'utilise **screen** (qui, malgré son nom, n'a rien d'un outil graphique). Screen, que tout sysadmin utilise quotidiennement, permet de créer des terminaux texte virtuels, un peu comme les bureaux virtuels de votre *window manager*, mais pour la console.

Les commandes principales sont :

- Démarrer un screen et le nommer « bureau » pour s'y référer facilement plus tard : **screen -S bureau** ;
- Par défaut un nouveau terminal est créé ;
- Créer un terminal supplémentaire dans le screen : **<Ctrl>** + **<A>** **c** ;

- Passer au terminal suivant : **<Ctrl> + <A> Espace** ;
- Revenir au terminal précédent : **<Ctrl> + A p** ;
- Lister les terminaux ouverts : **<Ctrl> + A "** ;
- Renommer un terminal : **<Ctrl> + A A** (majuscule) ;
- Se déconnecter du screen : **<Ctrl> + A d** ;
- Se reconnecter au screen : `screen -rd bureau` ;
- Naviguer dans le buffer du terminal : **<Ctrl> + A <Échap>**, puis utilisez les flèches, **<PageUp/Down>** et **<Échap>** pour sortir.

Ainsi, dans notre screen, nous allons créer quatre terminaux et les nommer de la façon suivante :

1. `xvfb` ;
2. `x11vnc` ;
3. `openbox` ;
4. `firefox`.

Ensuite, chacun de ces terminaux permettra de démarrer le programme correspondant en *foreground*. Ainsi nous pourrons voir les logs d'exécution en direct et nous pourrons facilement l'arrêter si besoin avec **<Ctrl>** + **<C>** : pas besoin de retrouver le PID ou faire un `ps`...

Conclusion

Pour résumer l'ensemble de l'article, une fois que tout est installé (`apt-get`) :

- démarrer un screen : `screen -S bureau` ;
- lancer xvfb dans le premier terminal, que vous nommez « xvfb »: `Xvfb :0 -nolisten tcp -screen 0 1200x700x16` ;
- créer un deuxième terminal, nommé « x11vnc », et lancer `x11vnc` : `x11vnc -usepw -many -rfbport 5900 -display :0` ;
- créer un troisième terminal, nommé « openbox », puis : `export DISPLAY=:0 ; openbox` ;
- créer un quatrième terminal, nommé « firefox », puis : `export DISPLAY=:0 ; firefox` ;
- vérifier que tout se passe bien en « cyclant » avec **<Ctrl> + <A> Espace** ou directement avec **<Ctrl> + <A> "** ;
- se connecter avec votre client VNC et profiter !

Si contrôler la situation ne vous intéresse pas, mettez simplement les commandes précédentes dans un script :

```
#!/bin/bash
Xvfb :0 -nolisten tcp -screen 0 1200x700x16 &
sleep 5
x11vnc -usepw -many -rfbport 5900 -display :0 &
export DISPLAY=:0 ; openbox &
export DISPLAY=:0 ; firefox &
```

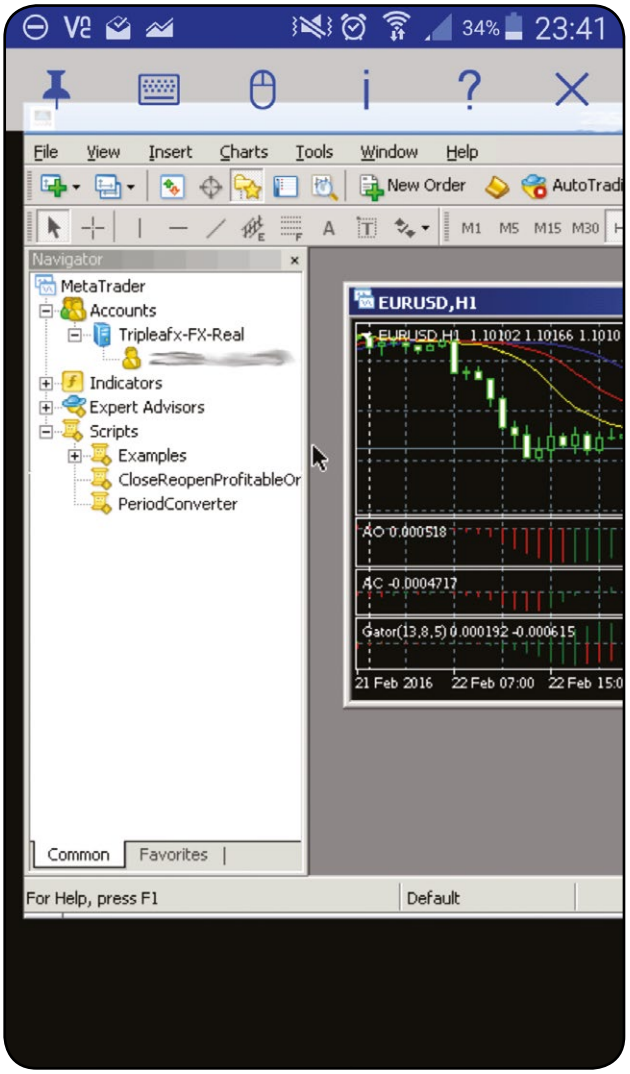


Fig. 3 : Depuis mon Android, je surveille mon robot Metatrader qui tourne sous Wine sur mon dédié Debian.

Le `sleep 5` est une petite sécurité, car les applications ne démarreront pas si le serveur X n'a pas fini de démarrer. Et pour conclure sur un exemple d'utilisation, la figure 3 montre mon robot Metatrader sous Wine sur un serveur dédié Debian. ■

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

TOMCAT EN VERSION « STATELESS »

Pierre-Emmanuel Gros [Responsable Innovation, Neuresys]

Le but de cet article est de faire les modifications nécessaires afin de rendre Tomcat « clusterisable » en mode stateless pour des applications Java Rich Internet Application. Au menu de l'article, on ouvrira le ventre d'un serveur J2EE pour faire les modifications indispensables au processus de sauvegarde de session.

Mots-clés : Session HTTP, Tomcat, Stateless Web Tier, Java/J2EE

Résumé

L'application phare de notre société est écrite en Java et gère un nombre important d'utilisateurs. Chaque utilisateur étant représenté par une session assez grosse, l'aventure de cet article a débuté lorsque j'ai dû reconfigurer le serveur Tomcat qui se mettait à gérer plus de 300 Mio de session. Nous aborderons donc dans cet article la programmation d'un module Tomcat permettant d'optimiser la gestion mémoire des sessions.

Vaadin est mon amour de développeur Java. Ce *framework* permet une mise en place aisée de ce que l'on appelle les RIA (*Rich Internet Application*), à savoir des applicatifs vivants dans un navigateur internet via un serveur J2EE. Par-delà le concept, Vaadin offre une potentialité de création d'interfaces graphiques riches entièrement basées sur la technologie Ajax et permettant de programmer très (trop?) rapdement.

Mon problème avec Vaadin était donc le suivant : plus mon nombre d'utilisateurs augmentait, plus Vaadin sauvegardait l'interface RIA dans des sessions du serveur Web, plus la taille mémoire nécessaire à l'applicatif augmentait. Bref, la mémoire du serveur était, non plus destinée à gérer des calculs complexes, mais à conserver des sessions utilisateurs et imposait par conséquent une surconsommation de temps de calcul pour le *garbage collector*.

Parce que ce problème de gestion de mémoire pour une session n'est pas une problématique purement Vaadin, j'ai décidé de mettre dans cet article la version « résumée » du long travail que nous avons mis en place sur Tomcat.

1 Serveur Web et Session HTTP

Avant de rentrer dans le vif du sujet, il est nécessaire de comprendre ce qu'est un serveur Web et ce qu'on appelle une session HTTP. Dans un premier temps, nous regarderons ce qu'est une session HTTP et dans un second temps, comment nous utilisons les sessions.

1.1 Qu'est-ce qu'une session HTTP ?

Les applicatifs/site Web sont hébergés sur un serveur Web utilisant le protocole HTTP. Ce dernier est un protocole Stateless ; on utilise la notion de session qui, via un numéro unique (dit numéro de session), permet le suivi des utilisateurs.

Deux méthodes de stockage de cette « session » sont possibles :

- soit via un cookie coté client. On parle dans ce cas de « Cookie Session Server » ;
- soit cet espace mémoire est conservé sur le serveur et l'identifiant de la session est renvoyé à l'utilisateur.

Dans cette dernière méthode, on retrouve côté client un cookie (nommé **PHPSESSID** pour les applicatifs PHP ou encore **JSESSIONID** pour les applicatifs Java) contenant l'identifiant de session.

Regardons maintenant comment manipuler ces sessions en Java.

1.2 Manipulation de sessions

L'idée est de voir maintenant comment « jouer » avec les sessions Java. La création de sessions se fait via l'objet **HttpServletRequest**, qui représente la requête HTTP. Cet objet passé à la Servlet contient une méthode **getSession** prenant un booléen. Ce booléen est là pour dénoter qu'il faut créer une session, si d'aventure elle n'existait pas déjà pour un utilisateur.

La manipulation d'une session en Java est proche de la manipulation d'un tableau associatif type **Map**. On trouve des méthodes :

- pour savoir si la session est fraîchement créée **isNew** ;
- pour placer un objet Java à l'emplacement d'une clef : **setAttribute** ;
- pour obtenir un objet Java potentiellement positionné sur une clef : **getAttribute**.

2 Tomcat et les sessions

Tomcat gère les sessions via une instance de « Session Manager » ayant la compétence pour créer et maintenir des sessions. Il est à noter que deux types de Session Manager sont possibles : le « Standard Persistent Manager », qui conserve les sessions en mémoire et le « Persistant Session Manager », qui est capable de sauvegarder les sessions inactives. La différence entre les deux est qu'avec les classes dérivées des « Persistant Session Manager », et en utilisant un *timer* bien placé, les sessions peuvent être actives en mémoire, inactives en mémoire et inactives mais non-résidentes en mémoire.

Cette manière de faire est un bon début de solution pour les applicatifs Web nécessitant de restreindre la taille mémoire de sessions. Il suffit de configurer, par exemple, pour une application X dans le répertoire **META-INF**, un fichier **context.xml** contenant :

```
<Manager className= " org.apache.catalina.session.
PersistentManager " saveOnRestart= " true ">
<Store className= " org.apache.catalina.session.FileStore "
directory= " /tmp "/>
</Manager>
```

Ce fichier XML, souvent suffisant pour des sites Web, ne l'a pas été pour nos applicatifs Web qui sont très sollicités. De ce fait, nous sommes obligés de construire un **SessionManager** spécifique.

3 Algorithmique de gestion de session

Le concept de « Stateless » indique souvent que la sauvegarde des sessions n'est pas faite dans le serveur mais en dehors du serveur. L'idée derrière ceci est d'avoir un réceptacle de session partagée par un cluster de serveur applicatif. Les membres du cluster étant affiliés à un utilisateur (on parle de « stickiness session ») ou pas (et dans ce cas on a des « sticky sessions »).

L'avantage de cette méthode de séparation entre la gestion de session et les serveurs est que la taille du cluster de serveurs applicatifs peut être diminuée ou augmentée facilement.

C'est approximativement cette approche que nous allons utiliser dans notre algorithmique :

- lorsqu'un utilisateur se connecte à Tomcat, soit il a déjà une session, soit nous lui en créons une ;
- cette session est sauvegardée en dehors du serveur (par exemple le système de fichiers) lorsque la requête a fini d'être traitée.

Pour réussir notre algorithme, nous avons besoin de quelques points essentiels :

- une compréhension des différentes classes de Tomcat gérant les sessions ;
- une capture de la fin d'une requête afin de pouvoir lancer la procédure de sauvegarde de session ;
- la capacité à « serializer/deserializer » une session afin de la sauvegarder/restaurer.

4 Management de session et Tomcat

Afin de résoudre notre problème de taille de sessions sous Tomcat, il faut d'abord voir quel est le comportement par défaut du manager de session de base.

4.1 ManagerBase

Le composant gérant les sessions se situe dans la classe « ManagerBase » du serveur applicatif qui implémente l'interface Manager de Tomcat.

Ce manager de sessions contient principalement les méthodes suivantes :

- **SessionIdGenerator getSessionIdGenerator**, qui a pour but de fournir un générateur d'identifiants uniques ;
- **Session createEmptySession**, qui doit fournir une session vide ;
- **Session createSession(String)** qui fournit une session avec un identifiant passé en paramètre (invoquer avec le paramètre **null** revient à invoquer **createEmptySession**) ;
- **Session findSession(String)** qui doit retrouver une session avec l'identifiant passé en paramètre ;
- **add(Session)** qui enregistre dans le manager une session.

Enfin, outre les méthodes de gestion de sessions, un manager de sessions se doit d'implémenter les méthodes **load** et **unload** qui sont invoquées au chargement/déchargement du manager.

Il est intéressant de regarder dans le code **ManagerBase** comment sont implémentées ces différentes méthodes et l'on remarquera qu'il s'agit principalement de manipulation d'une tâche de « haschage sessions », que l'on retrouve parmi les attributs de la classe. La définition de cette dernière est ainsi faite :

```
protected Map<String, Session> sessions = new ConcurrentHashMap<>();
```

La méthode **findSession** est implémentée via un accès direct à la **HashMap** en utilisant l'identifiant de session comme clef.

La création de sessions via **createSession(String)** entraîne l'exécution du code suivant :

```
@Override
   public Session createSession(String sessionId) {
```

Tomcat vérifie en premier que le nombre maximal de sessions possibles n'est pas encore créé via le code suivant , puis invoque la création de sessions via **createEmptySession** avec, soit l'identifiant passé en paramètre, soit un générateur d'identifiants.

```
        Session session = createEmptySession(); session.setNew(true);
        ///Suite du code d'iniatlisation d'une session
        String id = sessionId;if (id == null) {id =
generateSessionId();} session.setId(id);
```

L'appel à **createEmptySession** invoque assez directement la méthode **getNewSession** qui va créer une session standard Tomcat :

```
   protected StandardSession getNewSession() {return new
StandardSession(this);}
```

L'invocation de cette méthode met en lumière le deuxième acteur de la gestion de sessions, qui est la session proprement dite, soit la classe **StandardSession**.

4.2 StandardSession

La classe **StandardSession** est donc la classe représentant une session et implémentant l'interface **HttpSession** utilisée côté applicatif. Cette dernière est relativement simple, mais nous allons attirer l'attention sur deux points :

1. Le constructeur de **StandardSession** prend en paramètre le manager de sessions. La session dans Tomcat est donc gérée, à la fois par le manager de sessions pour la construction et la suppression, mais aussi par l'objet **Session** lui-même.
2. La session va s'enregistrer elle-même dans le manager et potentiellement demander sa suppression.

Ceci est visible dans la méthode **setId** de la session. Cette méthode, comme nous l'avons explicitée plus haut, sert à positionner l'identifiant de la session sur cette dernière. Néanmoins, et c'est le troisième point remarquable, après observation du code source, la méthode enregistre également la session dans le manager :

```
public void setId(String id, boolean notify) {
    if ((this.id != null) && (manager != null)) manager.remove(this);
    this.id = id;
    if (manager != null) manager.add(this);
```

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

5 Capture d'une requête HTTP dans Tomcat

Une requête HTTP dans un serveur applicatif peut être capturée de plusieurs façons : par les « webfilters », qui sont définis dans la norme J2EE et par les « Valves », qui sont le système interne de Tomcat.

Le but de ces deux technologies est le même : capturer une requête HTTP avant/après que cette dernière n'accède aux serveurs applicatifs. Les valves peuvent donc être vues comme une sorte de programmation par Aspect coté WebServer,et nous donnons ici une manière de les programmer. La démarche consiste à hériter de la classe **ValveBase** et implémenter la méthode suivante :

```
public void invoke(Request request, Response response) throws
IOException, ServletException
```

Cette méthode permet, avant de traiter la requête et la réponse HTTP, d'invoquer la valve suivante :

```
getNext().invoke(request, response);
```

6 Jeux d'interfaces afin de sauvegarder/restaurer des sessions

Notre jeu d'interfaces a pour but de transformer des sessions en tableaux d'octets (et vice-versa) afin de les sauvegarder.

Pour sauvegarder ces résultats, nous introduisons une interface modélisant un *pool* de connexions. Le but de ce *pool* est d'obtenir des connexions, chaque connexion représentant en emplacement de stockage en mot-clé/valeur.

L'interface des connexions est assez simple et permet de mettre/supprimer/obtenir un tableau d'octets (dans notre cas une session « serialize ») :

```
public interface MyConnection {
    // place le tableau d'octets à la clef id
    void set(String id, byte[] serializeFrom);
    //supprime les données de la clef id
    void delete(String id);
    //obtient les données stockées dans la clef id. Renvoie null
si aucune donnée n'est présente
    byte[] get(String id);
    void flush();
}
```

Dans le cadre d'une sauvegarde de sessions dans le système de fichiers, la méthode **set** peut s'écrire par « sérialization » du paramètre **serializedFrom** dans un fichier. L'interface de la **ConnectionPool** est aussi limpide que celle des connexions :

```
public interface ConnectionPool {
    //obtient une connection
    public MyConnection getConnection() ;
    // ferme une connection
    public void close(MyConnection connection);
    //supprime le pool
    public void destroy();
}
```

Maintenant que nous avons de quoi sauvegarder nos données, il est important de voir comment « sérializer » une session vers un tableau d'octets.

La « serialization » en Java est assez simple et peut passer par les API **readObjectData** et **writeObjectData** de l'objet **StandardSession** dans Tomcat. Afin de faire les choses proprement, nous introduisons une interface permettant de mettre en place ces opérations :

```
public interface Serializer {
    byte[] serializeFrom(StandardSession session) throws
IOException;
    void deserializeInto(byte[] data, StandardSession session)
throws IOException, ClassNotFoundException;
```

Les méthodes sont définies de façon on ne peut plus classique en utilisant la « sérialization » de base en Java ainsi que les méthodes **writeObjetData** et **readObjetData** de la classe **StandardSession**. Ces deux méthodes ont pour but de « serializer/deserializer » une session dans un flux.

7 Création de MyManager, MyValve et MySession

Nous allons maintenant créer notre manager de session **MyManager** et nos sessions instances de classes **MySession**. Ces créations viennent par héritage des classes **ManagerBase** et **StandardSession**.

Ces classes sont assez longues et dans cet article, nous allons mettre en avant les points saillants.

En premier lieu, abordons la classe **MyManager** qui est un héritage de **ManagerBase**. Cette classe contient comme attributs principaux un objet implémentant **MyConnection** et un objet implémentant **Serializer**. Enfin nous conser-

vons dans le manager la session courante au regard du *thread* qui utilise la session.

C'est en cela que nous introduisons les attributs suivants :

```
protected ThreadLocal<MySession> currentSession = new
ThreadLocal<>();
protected ThreadLocal<String> currentSessionId = new
ThreadLocal<>();
```

La première méthode à implémenter est la méthode **startInternal** qui est invoquée à l'initialisation du manager par Tomcat. Cette dernière méthode appelle l'initialisation de la classe parente et passe en revue les différentes **Valves**. Lorsqu'un objet de classe **MyValve** est rencontré, nous le capturons :

```
@Override
  protected synchronized void startInternal() throws
LifecycleException {
    super.startInternal();
    setState(LifecycleState.STARTING);
    for (Valve valve : getContainer().getPipeline().getValves())
{
        if (valve instanceof MyValveBase) {
            this.handlerValve = (MyValveBase) valve;
            this.handlerValve.setMyManager(this);
            break;
        }
    }
```

Cette capture permet de présenter la classe **MyValve** qui, en digne descendante de **ValveBase**, sera invoquée autour des requêtes HTTP utilisant **MyManager**. L'idée de notre **Valve** est simple : on lui demande de signaler au manager que la requête vient de se finir, car c'est à ce moment-là que nous sauvegarderons les sessions utilisateurs :

```
public class MyValve extends ValveBase  {
    private MyManager manager;
    public void setMyManager(MyManager manager) {this.manager =
manager;}
    @Override
      public void invoke(Request request, Response response)
throws IOException, ServletException { try {getNext().
invoke(request, response);} finally {manager.afterRequest();}}
```

Dans cette classe **MyValve**, il faut donc voir avec attention le bloc **finally** de la méthode **invoke** qui va appeler après chaque appel HTTP la méthode **afterRequest** de notre **Manager**.

Nous avons donc dans notre manager une méthode à écrire pour créer des sessions et une méthode pour les sauvegarder (**afterRequest**), invoquée par la **Valve**.

La liaison entre la **Valve** et le manager se fait via la méthode **startInternal** du manager. Nous profitons également de cette méthode pour créer deux objets permettant la « sérialization » et la persistance :

```
this.connectionPool=new FileConnectionPool();
this.serializer=new JavaSerializer();
```

La classe **FileConnectionPool**, classe implémentant l'interface **ConnectionPool** définie plus haut, est un *pool* de connexion vers le système de fichiers. Enfin, la classe **JavaSerializer** implémente l'interface **Serializer** que nous avons déjà présentée via le mécanisme de « sérialisation » standard de Java.

Ces deux objets étant triviaux à écrire, et afin de rester le plus synthétique possible, nous n'en donnons pas ici le code.

Le dernier acteur de notre code est la classe **MySession** héritant directement de **StandardSession**.

La méthode de création de sessions vides dans le manager est donc triviale :

```
    @Override
    public Session createEmptySession() {
        return new MySession(this);
    }
```

Attaquons-nous à la méthode de création de sessions nommées :

```
    @Override
    public Session createSession(String requestedSessionId) {
        MySession session = null; String sessionId = null;
```

La première chose à faire est de gérer le cas où l'identifiant de la session demandé est **null** en créant un identifiant ex-nihilo via la méthode **generateSessionId** de Tomcat :

```
      if (requestedSessionId) {sessionId=generateSessionId()}
else sessionId=requestedSessionId ;
```

Avec cet identifiant, nous créons la session proprement dite en l'informant qu'elle est valide, nouvelle, avec une date de création et son identifiant **sessionId** :

```
        session = (MySession)createEmptySession();
        session.setNew(true);session.setValid(true);
        session.setCreationTime(System.currentTimeMillis());
        session.setMaxInactiveInterval(getMaxInactiveInterval());
        session.setId(sessionId);session.tellNew();
```

Cette dernière session devient la session courante du manager et nous faisons le choix de la sauvegarder :

```
        currentSession.set(session);currentSessionId.
set(sessionId);
        MyConnection connection = connectionPool.getResource();
        saveInternal(connection, session);
```

La méthode **saveInternal** est la méthode de notre manager pour sauvegarder une session dans une connexion.

Cette dernière est naturellement invoquée dans la méthode appelée par la **Valve afterRequest**. Celle-ci prend la session courante (obtenue via **currentSession**) et appelle la méthode **saveInternal**.

```
public void afterRequest() {
    MySession redisSession = currentSession.get();
    MyConnection connection = connectionPool.getResource();

    if (redisSession != null) {
        try {
            if (redisSession.isValid()) {
                saveInternal(connection, session);
            }
```

La méthode **saveInternal** est assez simple et a pour but de « serializer » la session passée en paramètre et de la sauvegarder dans la connexion. Ceci donne :

```
protected boolean saveInternal(MyConnection jedis, Session
session, boolean forceSave) throws IOException {
    MySession mysession = (MySession)session;
    String binaryId = redisSession.getId();
    MyConnection connection = connectionPool.getResource();
    connection.set(binaryId, serializer.
serializeFrom(mysession));
```

La dernière chose à voir avec attention est la relecture de sessions qui se fait via la méthode **findSession** :

```
public Session findSession(String id) throws IOException
```

Nous allons utiliser l'identifiant passé en paramètre pour demander la session à la connexion. Soit cette session est **null** et dans ce cas, il n'existe pas de session portant l'identifiant (Tomcat invoquera alors la méthode **createEmptySession**), soit elle existe et nous la renvoyons. En premier lieu, nous récupérons une connexion et un potentiel tableau d'octets correspondant à la session :

```
    MyConnection connection = connectionPool.getResource();
    MySession session=null ;byte[] data = connection.get(id);
```

Si ce tableau n'est pas vide, nous créons une session vide et nous demandons à « déserializer » ce tableau dans la session !

```
    if (data != null) {  session = (MySession)
createEmptySession();
    serializer.deserializeInto( data,session);}
```

Nous signalons à cette session créée ex-nihilo qu'elle n'est pas nouvelle (utilisation de **setNew** à **false**), qu'elle est valide et accédée par une application (Tomcat mettant à jour les différents **TimeOut**).

```
    session.setId(id);
    session.setNew(false);
    session.setMaxInactiveInterval(getMaxInactiveInterval());
    session.access();
    session.setValid(true);
    currentSession.set(session);
    currentSessionId.set(id);
```

Nous avons donc créé les méthodes permettant à notre manager de :

- créer/retrouver une session ;
- sauvegarder la session après chaque requête HTTP, via l'appel d'une **Valve** ;
- faire fonctionner ces méthodes grâce à deux éléments tiers et « serialization » de la persistance d'une session Java.

Test et Conclusion

Afin de pouvoir faire les tests, nous allons rajouter un fichier **context.xml** dans le répertoire **META-INF** de notre application. Le but de ce fichier est de signaler à Tomcat qu'il doit prendre en compte notre **Valve** et notre manager :

```
<Context>
    <Valve className="com.neuresys.manager.MyValve" />
    <Manager className="com.neuresys.manager.MyManager"/>
</Context>
```

Dans cette fin d'article qui laisse supposer un suivant, nous abordons la vraie problématique de cette situation. Lors des tests, nous nous sommes aperçus de la présence d'un *bottelneck* sur les performances (principalement due à la « serialization/deserialization »). L'idée est donc de reprendre notre *framework* afin de voir s'il n'est pas possible de l'augmenter en changeant de couche de « serialization » et de méthode de stockage. ■

GIT INIT /ETC

Cyrille Colin [Direction du numérique – Université de Lorraine]

Quels fichiers de configuration ont été modifiés, quels paquets ont été installés ? Nous allons voir comment, avec l'outil etckeeper, nous pouvons obtenir les réponses à ces questions.

Mots-clés : Etckeeper, Versionning, Git, Etc, Configuration, Administration système, Gitlab, Tickets

Résumé

Conserver et consulter les modifications de configuration d'un serveur ? L'emploi des outils de gestion de version, comme git, permettent d'assister l'administrateur. Etckeeper permet lui d'adapter un système de gestion de versions à la sauvegarde d'un système de fichiers comme `/etc`. Couplé à un système de tickets il permet une traçabilité des actions, de visualiser les modifications apportées au système.

1 Problématique

La problématique n'est pas nouvelle, chaque changement sur nos systèmes (modification de fichiers de configuration, mise à jour, installation de nouveaux paquets) apporte son lot de surprises et les éternelles questions qui vont avec : « mais j'ai changé quoi ? » ou « qu'est-ce qui a changé ? ». Bon d'accord, c'est souvent la première hypothèse, un petit changement hâtif aura eu raison de la santé de notre serveur.

Dans l'idéal, il serait intéressant de savoir quels fichiers ont été modifiés, de connaître leurs anciens contenus, et pour aller plus loin, les changements de droits ou de propriétaires dont ils ont fait l'objet. Nous pourrions avoir une image de notre système de fichier, ici `/etc`, à des moments différents et ainsi de savoir qui a changé quoi entre deux ou plusieurs versions. Il faudrait également pouvoir connaître les paquets qui ont été mis à jour ou nouvellement installés, car oui nous ne sommes pas toujours à l'origine du problème.

La solution sera d'utiliser etckeeper, un outil open source développé par Joey Hess, qui permet d'adapter un système de gestion de versions à la sauvegarde d'un système de fichiers comme `/etc` et à la gestion d'un système d'exploitation et son utilisation. Pour cela, il met en œuvre un mécanisme simple de conservation des droits sur les fichiers, ainsi que des *hooks* sur les systèmes de gestion de paquets pour automatiser la conservation des changements et des installations sur le système.

2 Les gestionnaires de versions

Sans rentrer dans les détails, les gestionnaires de versions, VCS en anglais (*Version Control System*), permettent de conserver la chronologie des modifications sur un système de fichier. Dans notre cas, nous nous intéresserons aux systèmes décentralisés (ou distribués) sur lesquels s'appuie etckeeper.

Etckeeper supporte les principaux : git, **mercurial**, **bazaar** et **darcs**. Comme le laisse supposer le titre, nous allons pour la suite de cet article utiliser git en tant que gestionnaire de versions. Ce n'est pas le choix par défaut du paquet etckeeper sur une Ubuntu Server 14.04 LTS qui me sert de support pour cet article, mais bazaar, le système de gestion de versions libres sponsorisées par Canonical Ltd (Ubuntu).

2.1 Git

Si git n'est pas déjà installé sur votre système, vous pouvez l'installer avec la commande suivante :

```
$ sudo apt-get install git-core
```

Il faut configurer deux variables globales qui vous identifieront lors des modifications et de leurs consignations (*commit*) :

```
$ git config --global user.name "Cyrille Colin"
$ git config --global user.email cyrille.colin@univ-lorraine.fr
```

Vous pouvez retrouver ces informations dans le fichier **~/.gitconfig** ou avec **git** :

```
$ git config --global --get user.name
$ git config --global --get user.email
```

Git peut aussi être paramétré par des variables d'environnements : **GIT_COMMITTER_NAME**, **GIT_COMMITTER_EMAIL**, **GIT_AUTHOR_NAME** et **GIT_AUTHOR_EMAIL**, que vous pouvez initialiser dans le **~/.bashrc** :

```
export GIT_AUTHOR_NAME="Cyrille Colin"
export GIT_AUTHOR_EMAIL=cyrille.colin@univ-lorraine..fr
```

Avec etckeeper, il est aussi possible, via son fichier de configuration **/etc/etckeeper/etckeeper.conf**, de passer des paramètres à git :

```
GIT_COMMIT_OPTIONS='--author="Cyrille Colin <cyrille.colin@univ-lorraine.fr>"'
```

2.2 Cas particulier du multi-utilisateur root

Il arrive parfois que les serveurs soient gérés à plusieurs sur le compte root. « Bien ou pas bien ? Telle est la question... » ; mais du coup, il est encore plus intéressant de savoir qui a fait quoi, bien que tout le monde se retrouve au final en root sur le serveur. Une petite astuce consiste à utiliser une capacité du client ssh qui permet d'envoyer des variables d'environnement !

En premier lieu, modifiez sur votre machine le **~/.bashrc** pour y déclarer vos variables git :

```
$ vi ~/.bashrc
...
export GIT_AUTHOR_NAME="Cyrille Colin"
export GIT_AUTHOR_EMAIL="cyrille.colin@univ-lorraine.fr"
```

Ensuite, il faut demander au client ssh d'envoyer ces variables à l'établissement d'une connexion, en éditant ou en créant le fichier de config **~/.ssh/config** et en y insérant la ligne suivante :

```
$ vi ~/.ssh/config
...
SendEnv GIT_*
```

Si la variable **SendEnv** existait déjà, vous pourriez la compléter en séparant les expressions régulières par un espace.

Il ne reste plus qu'à autoriser le serveur ssh à accepter ces variables d'environnement. Pour cela, il faut se rendre sur le serveur, éditer le fichier **/etc/ssh/sshd_config** et compléter la ligne **AcceptEnv** avec l'expression **GIT_*** :

```
# vi /etc/ssh/sshd_config
# Allow client to pass locale environment variables
AcceptEnv LANG LC_* GIT_*
```

Une petite vérification pour s'assurer que tout est correct :

```
$ ssh root@stargate
# echo $GIT_AUTHOR_EMAIL
cyrille.colin@univ-lorraine.fr
```

C'est tout bon !

Attention !

Si vous utilisez les variables d'environnement, attention au **sudo** qui exécute une commande en tant que super utilisateur et qui ne conservera pas par défaut celle de l'utilisateur courant. Pour reporter les variables d'environnement, utilisez **sudo -E**.

3 Installation d'etckeeper

L'installation du paquet se fait comme à l'accoutumée :

```
$ sudo apt-get install etckeeper
```

À noter le message d'alerte en fin d'installation qui peut varier suivant que vous ayez ou non bazaar d'installé, car par défaut, l'installation crée le dépôt en utilisant bazaar (**bzr**) comme gestionnaire de version.

```
etckeeper init not ran as bzr is not installed
```

Si vous avez bazaar d'installé mais que vous ne souhaitez pas pour autant l'utiliser pour etckeeper, vous pouvez toujours supprimer le dépôt avec :

```
$ sudo etckeeper uninit
```

Cette commande est valable, quel que soit le gestionnaire choisi.

3.1 Utilisation de git comme gestionnaire

Pour qu'etckeeper utilise git, nous devons éditer son fichier de configuration et remplacer bazaar par git :

```
$ vi /etc/etckeeper/etckeeper.conf
```

Configuration par défaut :

```
# The VCS to use.
#VCS="hg"
#VCS="git"
VCS="bzr"
#VCS="darcs"
```

À remplacer par :

```
# The VCS to use.
#VCS="hg"
VCS="git"
#VCS="bzr"
#VCS="darcs"
```

3.2 Initialisation d'etckeeper

```
$ sudo etckeeper init
Initialized empty Git repository in /etc/.git/
```

Cette commande a pour effet de créer notre dépôt git dans **/etc**... c'est le but recherché. Rendons-nous dans **/etc** et regardons les premiers changements avec **git status** :

```
$ cd /etc
$ sudo git status
# On branch master
#
# Initial commit
#
# Changes to be committed:
#   (use "git rm --cached <file>..." to unstage)
#
# new file:   .etckeeper
# new file:   .gitignore
... (longue liste de nos fichiers présent dans /etc)
```

Nous pouvons voir la liste de tous les fichiers du répertoire **/etc** en attente d'être consignés, ce que nous effectuons avec la commande suivante :

```
$ sudo etckeeper commit -m "Creation du depot"
```

Ensuite, nous pouvons vérifier notre « commit » :

```
$sudo git log
commit d4e3a99681d49fbf5056f8c97f8da15cc3ab9f7f
Author: Cyrille Colin <cyrille.colin@univ-lorraine.fr>
Date:   Mon Feb 1 14:52:52 2016 +0000

    Creation du depot
```

Ou plus en détail :

```
root@stargate:/etc# sudo git log --summary |more
commit d4e3a99681d49fbf5056f8c97f8da15cc3ab9f7f
Author: Cyrille Colin <cyrille.colin@univ-lorraine.fr>
Date:   Mon Feb 1 14:52:52 2016 +0000

    Creation du depot

 create mode 100755 .etckeeper
 create mode 100644 .gitignore
```

Plusieurs choses sont à relever ici :

- etckeeper crée un fichier **.gitignore** afin de ne pas « sauvegarder » les fichiers inutiles ;
- les droits et les propriétés sur les fichiers sont archivés.

3.3 Rappel sur la sécurité

Comme le rappelle l'auteur d'etckeeper, en créant un dépôt, vous faites une copie de fichiers sensibles qui doivent rester secrets, comme **/etc/shadow** par exemple. Il faut garder à l'esprit, qu'en dupliquant un fichier, vous l'exposez forcément à une plus grande vulnérabilité.

Etckeeper prend soin de garder les droits associés aux fichiers pour ne permettre leur accès qu'au compte root. Cependant, vous devez être très vigilant, en copiant ou en clonant ce dépôt, à la possibilité qu'il soit accessible à d'autres personnes, sachant que forcément, il contient des données sensibles, voire très sensibles.

4 Utilisation d'etckeeper

Pour ceux qui sont aguerris à l'utilisation de git, il n'y a pas de changement. J'invite les autres à consulter le livre « ProGit » de Scott Chacon en français sur http://git-scm.com/book/fr/v2

ou pour comprendre l'essentiel, je conseille le site http://try.github.io/ qui vous propose, dans une console interactive, de prendre connaissance avec git (site en anglais).

4.1 Modifications dans /etc

Exemple de changement d'un mot de mot de passe :

```
root@stargate:/etc# passwd cyrille
Enter new UNIX password:
Retype new UNIX password:
passwd: password updated successfully
root@stargate:/etc# git status
On branch master
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git checkout -- <file>..." to discard changes in working directory)

  modified:   shadow

no changes added to commit (use "git add" and/or "git commit -a")
```

Nous voyons que le fichier **shadow** a bien été modifié, il faut ensuite consigner nos modifications :

```
root@stargate:/etc# git commit -a -m "changement de mot de passe
du compte cyrille"
[master 3dca290] changement de mot de passe du compte cyrille
 1 file changed, 1 insertion(+), 1 deletion(-)
```

Confirmation du log, un commit a bien été crée :

```
root@stargate:/etc# git log
commit 3dca290890c1d3bf89182a177f8cd6c176d70691
Author: Cyrille Colin <cyrille.colin@univ-lorraine.fr>
Date:   Mon Feb 1 15:00:12 2016 +0000

    changement de mot de passe du compte cyrille
```

4.2 Ignorer des fichiers

Par défaut, etckeeper crée un fichier **.gitignore** dans **/etc**. Celui-ci contient un ensemble de règles permettant d'exclure des fichiers. Cependant ,si vous constatez que des fichiers sont gérés et qu'ils sont sans importance, vous pouvez les retirer du dépôt et les déclarer « à ignorer ».

```
# git rm --cached monfichier
```

Attention à bien préciser **--cached**, qui ne fait que supprimer le fichier du dépôt git et non du système de fichier.

```
# echo monfichier >> .gitignore
```

Validation des changements :

```
# git commit -a -m "Arrêt du suivi de monfichier"
```

4.3 Gestionnaires de paquets

Un des avantages d'etckeeper est son intégration avec les gestionnaires de paquets qui lui permet d'enregistrer automatiquement les modifications effectuées sur **/etc** en cas d'installation, de suppression ou de mise à jour d'un programme. Exemple avec l'installation de **Monit**, un logiciel de surveillance de services locaux :

```
root@stargate:/etc# apt-get install monit
Lecture des listes de paquets... Fait
Construction de l'arbre des dépendances
Lecture des informations d'état... Fait
Paquets suggérés :
  exim4 postfix mail-transport-agent
Les NOUVEAUX paquets suivants seront installés :
  monit
0 mis à jour, 1 nouvellement installés, 0 à enlever et 42 non mis à jour.
Il est nécessaire de prendre 286 ko dans les archives.
Après cette opération, 771 ko d'espace disque supplémentaires seront utilisés.
Réception de : 1 http://archive.ubuntu.com/ubuntu/ trusty/universe monit amd64
1:5.6-2 [286 kB]
286 ko réceptionnés en 0s (1 318 ko/s)
Sélection du paquet monit précédemment désélectionné.
(Lecture de la base de données... 13188 fichiers et répertoires déjà installés.)
Preparing to unpack .../monit_1%3a5.6-2_amd64.deb ...
Unpacking monit (1:5.6-2) ...
Processing triggers for ureadahead (0.100.0-16) ...
Paramétrage de monit (1:5.6-2) ...
invoke-rc.d: policy-rc.d denied execution of start.
Processing triggers for ureadahead (0.100.0-16) ...
[master 31b80f7] committing changes in /etc after apt run
 31 files changed, 745 insertions(+)
 create mode 100644 default/monit
 create mode 100755 init.d/monit
 create mode 100644 logrotate.d/monit
 create mode 100644 monit/monitrc
 create mode 100644 monit/monitrc.d/acpid
...
 create mode 120000 rc5.d/S99monit
 create mode 120000 rc6.d/K01monit
```

Vérification :

```
root@stargate:/etc# git log -1
commit 31b80f74323d893aeccd9f57b032462c5eea560a
Author: Cyrille Colin <cyrille.colin@univ-lorraine.fr>
Date:   Mon Feb 1 15:03:50 2016 +0000

    committing changes in /etc after apt run

    Package changes:
    +monit 1:5.6-2
```

4.4 Les grosses modifications

Avec un changement de version de programme, il arrive que les fichiers de configuration changent totalement, et vous savez que vous aurez tout intérêt à basculer sur cette

nouvelle version, sans pour autant à avoir à mettre en place une architecture de test ; ce qui implique la modification de nombreux fichiers avec une incertitude de réussite ! L'intérêt serait donc de suivre les modifications effectuées mais aussi d'avoir la possibilité de revenir en arrière.

Attention !

Attention à l'utilisation des branches, si vous étiez amené à vous connecter sur une branche dans **/etc** avec **git checkout mabranche**, ceci affecterait immédiatement votre système de fichier en y plaçant le contenu de la branche et vous devrez exécuter **etckeeper init** pour qu'il installe les permissions sur les fichiers.

4.4.1 Clone

Voici une technique parmi tant d'autres pour réaliser des modifications sans toucher directement à votre système de fichier **/etc**. N'oubliez pas que recopier son dépôt **/etc**, c'est toujours prendre un risque, aussi il faut prendre quelques précautions. Avant tout, il faut créer un répertoire accessible uniquement par soi :

```
# mkdir temp_etc/
# cd temp_etc/
# chmod 700 .
```

Puis, y cloner **/etc** :

```
# git clone /etc
```

Ensuite, il ne faut pas oublier de rétablir les permissions par défaut avec etckeeper :

```
# cd etc
# etckeeper init -d .
# chmod 755 ..
```

Maintenant nous sommes prêts à attaquer nos modifications. Pour cela nous allons créer une branche de travail, nommons-la **nouvelle_conf**.

```
# git branch nouvelle_conf
# git checkout  nouvelle_conf
```

Ceci aurait aussi pu s'écrire :

```
# git checkout -b nouvelle_conf
```

Vous effectuez toutes vos modifications dans cette nouvelle branche, modifications que vous consignez et vous vous préparez à les apporter à votre configuration en vous rendant dans **/etc**, en déclarant votre clone et en important vos changements. Les différentes étapes sont : ajout du dépôt distant, rapatriement du dépôt distant et fusion des données :

```
# cd /etc
# git remote add temp_repo temp_etc/etc
# git fetch temp_repo
# git merge temp_repo/nouvelle_conf
```

Ensuite il faut rétablir les droits sur les fichiers avec etckeeper :

```
# etckeeper init -d .
```

Vous pouvez supprimer le répertoire de travail.

5 Git push

Attention !

Attention à la duplication de dépôt, ce qui peut s'avérer pratique comme une gestion centralisée de configuration peut également s'avérer très risqué. Toutes les configurations sont exposées à l'utilisateur git, la moindre compromission sur le serveur serait gravement préjudiciable.

Une autre utilisation du clonage consiste en la sauvegarde des fichiers de configuration sur une machine distante. Pour ma part, comme j'utilise **GitLab** (voir figure 1) pour gérer mes développements, j'ai décidé d'y importer mes configurations de serveurs et de profiter du système de gestion de tickets, du wiki, etc. pour la gestion des machines.

Attention !

Gitlab est un projet intéressant. Il intègre parfaitement git à la manière de GitHub, qu'on ne présente plus. Cependant, il reste un logiciel réservé aux techniciens. Tout comme GitHub, il est uniquement disponible en anglais et par conséquent, son utilisation ne pourra souvent pas être étendue. De plus, il n'est pas « packagé », ce qui présente un gros frein à son installation, bien que celle-ci soit relativement aisée.

Pour ce faire, j'utilise un groupe nommé **etc**, dans lequel est crée un projet par machine, ce qui donne pour

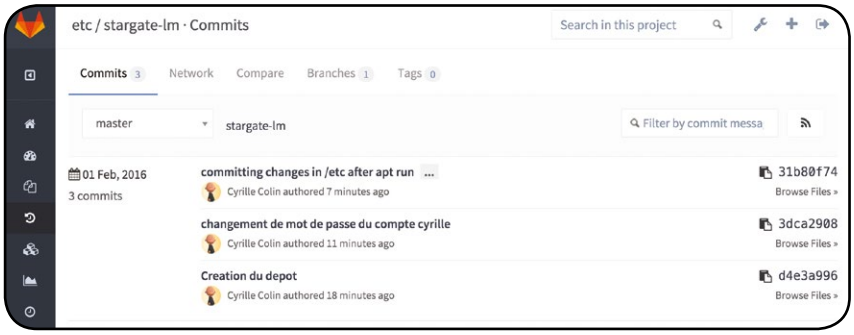


Fig. 1 : Vue de GitLab.

la machine **stargate**, une adresse de ce style : **git@monserveurdedépôt:etc/stargate.git** qui a l'avantage d'être assez lisible et facile à retenir.

Pour importer **/etc** dans GitLab :

```
git remote add origin git@monserveurdedépôt:etc/stargate.git
git push -u origin master
```

5.1 Gestion des demandes

GitLab propose un gestionnaire de tickets intégré, utile pour les demandes de changement de configuration, et sur un serveur : ajout d'une acl, d'un compte, installation d'un programme, etc ...

Pour l'exemple, la figure 2 montre un ticket pour rappeler qu'il faut mettre à jour le certificat d'Apache.

Comme souvent, il est pratique d'associer le commit au ticket et également de pouvoir le clore. Ceci est possible en utilisant dans le commentaire du commit la syntaxe **fixes #N**, où **N** est le numéro du ticket à fermer :

```
# git commit -a -m "Ajout des nouveaux certificats fixes #1"
[master 5828cc1] Ajout des nouveaux certificats fixes #1
 2 files changed, 1 insertion(+)
 create mode 100644 ssl/certificat
```

Et nous n'oublions pas de pousser notre commit sur le dépôt distant, sur lequel GitLab va interpréter le commentaire et apporter les modifications sur le ticket en le fermant :

```
# git push origin master
```

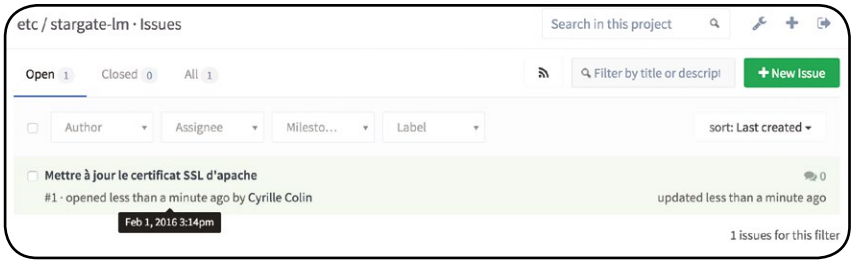


Fig. 2 : Ticket de rappel de mise à jour du certificat d'Apache.

6 Configuration

Le fichier de configuration est **/etc/etckeeper/etckeeper.conf** mais le plus intéressant se trouve dans les répertoires **command.d** dans **/etc/etckeeper** où il est possible pour chaque phase d'exécution de spécifier des scripts à lancer.

6.1 Push automatique

Si vous avez mis en place un dépôt distant, autant synchroniser automatiquement son contenu après chaque « commit ». Pour cela, il faut créer un script dont le seul but sera de lancer la commande **git push origin master** :

```
# cd /etc/etckeeper/commit.d
# (echo '#!/bin/sh' ; echo 'git push origin master') > 99git-push
# chmod +x 99git-push
```

Puis, comme nous avons modifié notre **/etc**, nous intégrerons ces modifications :

```
# git add .
# git commit -m "Ajout du push automatique pour etckeeper"
```

À noter que la commande **git commit** ne pousse (*push*) pas automatiquement à l'aide de la commande ci-dessus, qui ne sera déclenchée que par **etckeeper commit**. En l'occurrence, ici, il faut pousser en manuel.

Conclusion

Voir l'évolution d'un serveur au fil du temps change la vision « immédiate » de la configuration. Les serveurs ont une vie et il est même possible de l'observer dans de beaux graphiques. Git, et par extension etckeeper, sont des outils d'aide à l'amélioration de la qualité des services. Ils assistent les administrateurs systèmes dans une tâche de gestion où souvent seule la mémoire prévaut. ■

RÉDIGER SON ARTICLE AVEC ASCIIDOC[TOR]

Benoît Benedetti [Administrateur Système Linux, Université de Nice Sophia Antipolis]

Si vous rédigez des articles pour GNU/Linux Magazine France, vous en avez peut-être assez d'utiliser OpenOffice/LibreOffice et de passer plus de temps à vous battre avec l'éditeur pour ajouter des styles à votre article, plutôt que dans la rédaction propre de l'article. Je vous propose de découvrir un thème Asciidoctor que j'ai créé et qui va vous permettre de rédiger dans votre éditeur de texte préféré votre article au format Asciidoc, pour le générer au format ODT attendu par l'équipe de rédaction du magazine. Et même si vous n'avez jamais écrit d'article, peut-être que cet article vous mettra le pied à l'étrier, ou plus simplement vous donnera des idées pour utiliser Asciidoc pour créer votre thème personnalisé Asciidoctor pour générer vos propres documents.

Rédaction Asciidoc Asciidoctor LibreOffice

L'OBJECTIF

Dans les phases 1.X de cet article, nous allons voir comment récupérer et utiliser le dépôt Git qui contient le thème Asciidoctor nécessaire que j'ai développé, et le tester avec des fichiers exemples. À la fin de cette phase, vous aurez validé l'installation des prés requis, vous aurez installé le thème nécessaire, et aurez évalué le potentiel de cette solution.

Dans les phases 2.X de cet article, nous allons voir comment mettre en place et démarrer rapidement l'écriture d'un article à l'aide du thème installé dans l'étape précédente. Nous verrons aussi comment configurer votre environnement pour visualiser en direct le résultat de votre article.

LES OUTILS

- Un système GNU/Linux ;
- L'utilitaire Git, pour cloner un dépôt distant ;
- Un environnement Ruby ;
- Un environnement LibreOffice/OpenOffice ;
- Votre éditeur de texte préféré: Vim, Emacs, Acme, etc.

PHASE 1

Dans cette phase nous allons installer les prés requis. Nous avons besoin de **Ruby** (en version au moins égale à 1.9, ce qui est le cas sur la plupart des distributions) ; de **Bundler** pour installer et gérer les gemmes Ruby ; de **Git**

pour cloner le dépôt source; de **soffice** pour générer le fichier final ODT, et d'un environnement **LibreOffice** pour pouvoir bien sûr ouvrir ces fichiers.

Sous Debian, l'installation de ces différentes dépendances s'effectue comme suit :

```
$ sudo apt install ruby bundler git libreoffice-common
```

Phase 1.1

On peut maintenant cloner le dépôt contenant le thème :

```
$ git clone https://github.com/humboldtux/asciidoctor-diamonded
```

Ce dossier contient également des fichiers exemples. Pour les tester, placez-vous donc dans le dossier cloné, toutes les commandes suivantes de cette section seront à exécuter depuis ce dossier :

```
$ cd asciidoctor-diamonded
```

Phase 1.2

Il faut maintenant installer les différentes gemmes Ruby nécessaires. Pour cela, des fichiers **Gemfile** et **Gemfile.lock** vous permettent de les installer facilement grâce à la commande suivante :

```
$ bundle install
Using asciidoctor 1.5.4
Using temple 0.7.6
Using tilt 2.0.1
Using slim 3.0.6
...
Using bundler 1.7.4
Your bundle is complete!
Use `bundle show [gemname]` to see where a bundled gem is installed.
```

Phase 1.3

Le dossier cloné contient des fichiers ***.adoc** au format Asciidoc :

```
$ ls *adoc
gnulinuxmagazine.adoc gnulinuxmagazineHS.adoc gnulinuxmagazine_
recette.adoc modele_article_diamond.adoc modele_misc.adoc
```

D'après les noms, vous comprenez que ces différents fichiers correspondent aux modèles fournis par les rédacteurs en chef des différents magazines à l'adresse www.editions-diamond.fr/devenir-auteur/. Un fichier **Makefile** va vous permettre de générer leur équivalent au format ODT. La simple exécution de la commande suivante va le faire automatiquement pour vous :

```
$ make
...
```

Les différents fichiers ODT ont été générés dans le dossier **builds** :

```
$ ls builds/*.odt
builds/gnulinuxmagazineHS.odt builds/gnulinuxmagazine.odt builds/
gnulinuxmagazine_recette.odt builds/modele_article_diamond.odt
builds/modele_misc.odt
```

Vous pouvez les ouvrir pour voir qu'ils correspondent bien aux fichiers ODT originaux fournis par les différentes rédactions.

Ça vous plaît ? Ces différents fichiers **adoc** servent à tester le thème. En lisant la source de ces fichiers exemple vous pouvez voir comment utiliser le thème pour écrire vos propres articles.

Par contre, le dossier cloné sert vraiment à obtenir les fichiers du thème, mais pas à écrire votre article depuis ce dossier.

PHASE 2

J'ai élaboré un dépôt Git qui contient différents fichiers pour vous aider à démarrer et à rédiger rapidement votre article, en utilisant les fichiers du thème cloné précédemment. On commence par cloner ce dossier de base :

```
$ git clone https://github.com/humboldtux/template-diamonded
```

Les commandes dans la suite seront à exécuter depuis le dossier cloné :

```
$ cd template-diamonded
```

Ce dossier contient différents fichiers :

```
$ ls
article.adoc content.adoc Gemfile.lock Makefile tmp
builds Gemfile Guardfile template_diamonded_figure_01.png
```

Phase 2.1

Ce dossier contient lui aussi un **Makefile** qui automatise certaines tâches, comme la génération du fichier ODT final. La génération a besoin du thème Asciidoctor, dont vous avez récupéré les fichiers en clonant le dépôt à l'étape 1.1.

Le chemin dans lequel vous avez cloné le dossier ne correspond sûrement pas à celui que j'utilise, et qui est écrit en dur dans le **Makefile**, à vous de modifier la variable **TPLDIR** de ce fichier :

```
TPLDIR=~/dev/src/github.com/humboldtux/
asciidoctor-diamonded
...
```

Phase 2.2

Si tout va bien, vous pouvez exécuter la commande **make** :

```
$ make
```

Celle-ci va générer le fichier **builds/article.fodt**, raccourci de la cible **fodt**.

Le format FODT est le format FLAT ODT : c'est une version non compressée du format odt. Pourquoi ce format? Car Asciidoctor ne génère pas directement en ODT, seulement en FODT.

Vous pouvez néanmoins ouvrir ce format avec OpenOffice/LibreOffice. Cette cible est donc une cible par défaut pour valider le rendu de votre article avant sa finalisation.

Note

J'ai choisi une cible par défaut différente pour le **Makefile** du dossier des thèmes cloné en 1.X : les fichiers exemple adoc de ce dossier sont déjà écrits, donc pas besoin de valider le rendu FODT.

Phase 2.3

Pour générer le format ODT final attendu par la rédaction du magazine, vous utilisez la cible **odt** :

```
$ make odt
```

Cette cible utilise la cible **fodt** précédente, puis utilise la commande **soffice** pour la convertir au format odt dans **builds**.

Attention !

Pour pouvoir s'exécuter, la commande **soffice** nécessite que OpenOffice/LibreOffice soit fermé. Il faut donc que vous fermiez l'application, avant de pouvoir générer votre odt. Aucune erreur n'apparaît lorsque vous exécutez **soffice** et que l'application est ouverte, vous donnant l'impression que la commande **make** s'est correctement déroulée: si vous n'avez aucun fichier odt généré dans **build** en lançant la commande, c'est que vous avez oublié de fermer une fenêtre OpenOffice/LibreOffice !

Phase 2.4

Votre article est fini ? Une cible **tgz** du **Makefile**, utilisant la cible **odt** précédente, va créer une archive dans **builds**, incluant votre article au format odt et toutes les images (images à placer directement à la racine du dossier de travail) :

```
$ make tgz
```

Il ne vous reste plus qu'à envoyer cette archive à la rédaction !

Phase 2.5

Dans cette phase nous allons voir comment utiliser le thème html fourni en supplément: lors de la rédaction de votre article, il peut être fastidieux de générer le fichier final ODT, s'apercevoir que vous avez oublié d'ajouter un style ou fait une faute d'orthographe, etc. Il faut alors fermer le fichier ODT, modifier le fichier source **article.adoc**, régénérer le fichier final, l'ouvrir et le relire pour vérification.

Tout ce processus s'avère fastidieux et j'ai donc ajouté un thème html minimal, qui va vous permettre d'itérer plus vite dans la rédaction de votre article.

Pour générer ce html, il suffit d'exécuter la tâche **html** du **Makefile** :

```
$ make html
```

Un fichier **article.html** est généré dans le dossier **builds**. Étant donné que c'est un « brouillon » servant à la rédaction, le thème inclut les images dans le rendu html. Cela vous permet aussi de vérifier facilement que vous avez correctement indiqué les noms de fichiers images, leur contenu et leur ordre d'insertion dans votre article.

Phase 2.6

Le fichier **article.html** généré précédemment est intéressant, mais il vous faut encore rafraîchir manuellement votre navigateur à coup de **<Ctrl>** + **<F5>** à chaque modification des fichiers sources **adoc**.

Le site officiel de Asciidoctor possède toute une page très détaillée pour mettre en place **Guard** et **Livereload** pour vous éviter ce désagrément, accessible à l'adresse asciidoctor.org/docs/editing-asciidoc-with-live-preview/.

Le fichier **Gemfile** du dossier récupéré contient toutes les dépendances pour Guard indiquées sur cette page, que vous avez déjà installées dans la phase 1.2.

Comme dépendances, il ne vous reste plus qu'à installer l'extension Livereload pour votre navigateur, et configurer les options de l'extension dans votre navigateur pour accepter les dossiers locaux.

J'ai également inclus le fichier **Guardfile** qui va bien, et va surveiller les fichiers **.adoc**, exécuter **make html** à chaque modification, ainsi qu'une règle de type Livereload, qui va avertir tout navigateur connecté que le fichier **article.html** a été modifié.

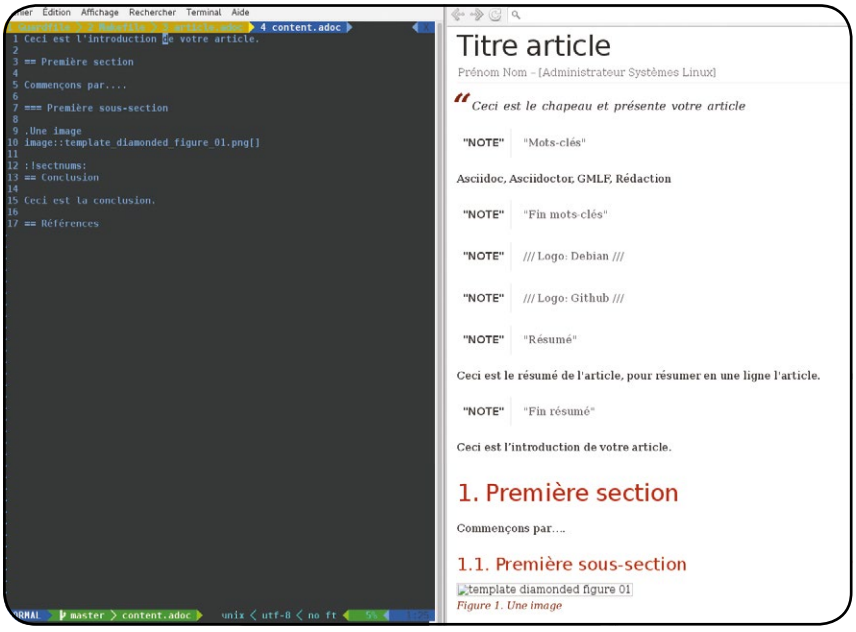


Figure 1: L'environnement du rédacteur agile

Enfin, j'ai créé dans le **Makefile** une règle **guard**. Cette règle va générer le fichier html, l'ouvrir automatiquement dans votre navigateur, puis lancer Guard :

```
$ make guard
...
bundle exec guard
18:58:40 - INFO - LiveReload is waiting
for a browser to connect.
18:58:40 - INFO - Guard is now watching
at 'template-diamonded'
[1] guard(main)>
```

Il ne vous reste plus qu'à activer l'extension Livereload dans votre navigateur, placer le navigateur d'un côté de l'écran, les fichiers source **adoc** dans votre éditeur préféré de l'autre, et apprécier la mise à jour automatique de votre article en temps réel (voir figure 1).

Phase 2.7

Vous vous passez de LibreOffice, et écrivez désormais votre article au format texte. Format qui vous permet de versionner votre article comme du code source. Pourquoi ne pas en profiter, surtout que votre dossier courant est un dossier Git, cloné dans la phase 2.0 ?

Par contre, vous êtes dans un dépôt cloné: vous n'êtes pas propriétaire de la branche distante source **origin**. Il va falloir changer de branche source pour que vous puissiez pousser votre code vers votre dépôt distant. Pour cela, la simple commande suivante fera l'affaire :

```
$ git remote set-url origin https://
exemple.fr/mon-nouveau-dépôt
```

Vous pourrez ensuite faire vos commits et exécuter un **push** vers votre dépôt distant.

Phase 2.8

Une dernière phase en guise de cadeau *Bonux*: vous avez écrit votre article, lu par des millions de lecteurs, il vous a apporté gloire et postérité, et vous voudriez pouvoir le conserver pour le lire plus tard et le montrer fièrement à vos petits-enfants.

Mais le lire dans OpenOffice/LibreOffice ne vous dit rien, surtout que les images ne sont pas incluses, et le format html ne vous intéresse pas non plus.

C'est pourquoi vous avez la possibilité de le générer aux formats pdf, epub3 et kf8/mobi : le **Gemfile** contient déjà toutes les gemmes nécessaires (**asciidoctor-epub3**, **asciidoctor-pdf**, etc.) qui ont donc été installées en phase 1.2.

Différentes cibles **make** sont incluses dans le **Makefile** pour générer votre article dans ces différents formats, et une cible **ebooks** générera tous ces formats en même temps :

```
$ make pdf
...
$ make epub3
...
$ make epub-kf8
...
$ make ebooks
...
```

Note

Des avertissements peuvent apparaître durant la génération des différents formats: vous pouvez les ignorer.

Attention !

De part la façon dont Asciidoctor gère et génère les fichiers EPUB3 et KF8/MOBI, il faut absolument que votre article soit découpé en au moins deux fichiers. Regardez les sources de **article.adoc**, qui incluent le fichier **content.adoc**. C'est une limitation, temporaire, de l'outil. Si vous ne comptez pas utiliser l'export EPUB3/KF8/MOBI, pas besoin de découper votre article en plusieurs fichiers.

LE RÉSULTAT

Nous sommes à la fin de ce « how-to », et ces deux étapes vous ont montré les outils de la solution et comment les utiliser pour débuter la rédaction de votre article. ■

ÉCRIRE SON THÈME ASCIIDOCTOR

Benoît Benedetti [Administrateur Systèmes Linux, Université de Nice Sophia Antipolis]

Vous avez testé la recette précédente pour générer un article à partir d'un fichier source Asciidoctor et vous aimeriez savoir comment écrire votre propre thème Asciidoctor pour générer vos documents personnalisés.

Mots-clés : Asciidoc, Asciidoctor, GMLF, Rédaction, Slim, (F)ODT

Résumé

Asciidoctor possède un *framework* de création de thèmes pour générer vos documents personnalisés que nous allons découvrir.

Dans cet article, nous allons voir comment utiliser un thème Asciidoctor pour pouvoir générer vos documents personnalisés, selon ce thème.

Nous prendrons comme exemple le thème présenté dans la recette précédente, qui permet de générer un document au format FODT respectant les styles de la rédaction de GLMF.

1 Introduction

1.1 Les thèmes sous Asciidoctor

Asciidoctor propose une *Stylesheet Factory* **[1]**, autrement dit tout un système basé sur **Compass**/**Sass** et **Foundation** pour développer votre thème et styliser le rendu généré en HTML5. Or, nous ne désirons pas générer du HTML5 mais du ODT...alors qu'Asciidoctor ne permet que de générer du FODT !

1.2 FODT?

Le FODT est du *Flat* ODT : c'est-à-dire une version non compressée d'un ODT, au format XML. C'est parce que ce fichier est un simple format texte/XML que Asciidoctor peut le générer. J'en parle déjà dans la recette précédente sur Asciidoctor : il faut ensuite utiliser la commande `soffice` de LibreOffice/OpenOffice pour convertir un FODT en ODT.

1.3 FODT de base

Quel document FODT pouvons-nous utiliser comme base pour savoir quoi générer avec notre thème ? C'est simple, il suffit de prendre un des modèles OTT ou ODT fournis par les éditions Diamond, par exemple `gnulinuxmagazine.ott`. Puis, depuis Libreoffice/OpenOffice, allez dans ***Fichier > Enregistrer sous > FODT***. Vous obtiendrez un fichier `gnulinuxmagazine.fodt`, au format FODT, que vous pourrez ouvrir, puis parcourir et analyser le contenu XML depuis n'importe quel éditeur de texte.

Note

Le fichier `fodt` généré n'est pas indenté correctement et difficile à lire. Vous pouvez utiliser l'outil **xmllint** (paquet `libxml2-utils` sous Debian) pour rendre sa lecture plus agréable :

```
$ xmllint --format gnulinuxmagazine.fodt
```

1.4 Slim

Nous n'utiliserons donc pas la *Stylesheet Factory* de Asciidoctor car elle est plus orientée stylisation de contenu HTML5. Mais nous n'allons pas non plus écrire en entier tout le contenu au format XML depuis notre thème, ce serait trop long et verbeux. Si vous regardez

les sources des *backends* de Asciidoctor **[2]**, vous verrez qu'Asciidoctor supporte plusieurs moteurs de *templates* qui permettent de générer un langage de balises comme XML : **Erb**, **Haml** et **Slim**.

Nous allons choisir Slim car il est simple d'utilisation et beaucoup plus lisible que les deux autres, surtout quand on veut faire de l'extrapolation de variables, ce qui arrive souvent depuis un fichier de *template* d'un thème Asciidoctor, pour pouvoir utiliser les variables et contenus générés par le préprocesseur Asciidoctor lorsqu'il parcourt un fichier Asciidoc.

Pour savoir comment générer du contenu en sortie, un fichier de *template* Slim se base, entre autres, sur l'indentation et les retours à la ligne. En particulier pour un langage de balises, cela permet de savoir quand s'arrête et se ferme une balise, qu'elle soit auto-fermante ou pas, juste via l'indentation et les retours à la ligne. Prenons l'exemple du fichier de *template* Slim suivant :

```
office:master-styles
  style:master-page style:name="Standard" style:page-layout-name="pm1"
  style:master-page style:name="Endnote" style:page-layout-name="pm2"
```

Celui-ci va générer le contenu XML/FODT suivant :

```
<office:master-styles>
  <style:master-page style:name="Standard" style:page-layout-name="pm1"/>
  <style:master-page style:name="Endnote" style:page-layout-name="pm2"/>
</office:master-styles>
```

1.5 Arborescence de fichiers de votre thème

Les fichiers de votre thème doivent être placés dans un dossier, selon une arborescence précise. Ce dossier du thème est à passer en paramètre de l'exécution de la commande **asciidoctor** qui traite un fichier au format Asciidoctor. Imaginons que vous ayez un fichier **exemple.adoc** à traiter ; vous lanceriez la commande :

```
$ asciidoctor --trace -T theme/slim -b fodt exemple.adoc -o exemple.fodt
```

L'option **--trace** permet d'activer la verbosité de la sortie de la commande. L'option **-T** sert à préciser le chemin du dossier du thème, qui pourrait ne pas être dans le dossier courant. On va créer ce dossier :

```
$ mkdir -p theme/slim
```

Pourquoi l'arborescence **theme/slim** et pas tout simplement un dossier **theme** ou **slim** ? Parce que vous pourriez décider de proposer des versions Erb ou Haml de votre thème comme alternatives, comme cela est visible dans les sources des *backends* officiels **[2]**.

Enfin, l'option **-b** permet de préciser le *backend* à utiliser dans votre thème, ici fodt, dont il faut créer le dossier :

```
$ mkdir theme/slim/fodt
```

Un *backend* est un format de sortie, généré par Asciidoctor. Vous pourriez avoir besoin de plus de *backends* pour votre thème : html5, pdf, mobi, etc. Pour chaque *backend*, vous mettez les fichiers de *templates* propres à ce *backend* dans son sous-dossier. Pour notre exemple, nous n'aurons que le *backend* fodt, donc tous les fichiers de *templates* Slim dont nous parlerons dans la suite seront à créer dans ce dossier **theme/slim/fodt**.

Maintenant que notre mise en place est prête, nous pouvons enfin passer à la création du thème.

2 Création du thème

Pour créer votre thème avec Asciidoctor, il vous faut procéder petit à petit : commencez à rédiger votre fichier **exemple.adoc** en utilisant les éléments de syntaxe Asciidoctor que vous aimeriez utiliser pour écrire votre document. Lancez la commande **asciidoctor** vue précédemment en 1.5 : Asciidocotor va traiter **exemple.adoc**, analyser son contenu, et lorsqu'il va trouver un élément de syntaxe Asciidoctor, va voir dans votre thème comment le traiter et quelle sortie générer. Si rien n'est disponible dans votre thème pour cet élément de syntaxe Asciidoctor, la commande va sortir en erreur, vous dire quel fichier est manquant dans votre thème pour traiter ce cas, et vous saurez quoi faire. Et ainsi de suite, jusqu'à ce que vous ayez fini d'implémenter dans votre thème tout ce dont vous avez besoin.

C'est exactement cette approche que j'ai adoptée pour créer le thème pour les articles des éditions Diamond : je suis parti d'un thème vide et d'un fichier **adoc** vide. J'ai ensuite vu que dans le fichier de modèle **gnulinuxmagazine.ott** il y avait par exemple le style gras à traiter : dans la syntaxe Asciidoctor, on met des éléments en gras en les encadrant par des *backquotes*. J'ai donc mis un mot entre *backquotes* dans mon fichier **adoc**, et lancé la commande **asciidoctor** précédente : Boom ! La commande m'affiche une erreur me disant que j'ai un fichier Slim à rajouter dans

mon thème. Quel contenu mettre dans ce fichier ? Il suffit d'aller dans le fichier **gnulinuxmagazine.fodt**, trouver un exemple XML/FODT d'un mot mis en gras, convertir cette syntaxe en Slim, puis l'ajouter dans le fichier manquant du thème et ainsi de suite.

C'est cette approche que nous allons aborder dans la suite, pour vous aider à comprendre la démarche.

2.1 Le fichier de squelette document.fodt.slim

Créez un fichier vide **exemple.adoc** :

```
$ touch exemple.adoc
```

Puis exécutez la commande précédente :

```
$ asciidoctor --trace -T theme/slim -b fodt exemple.adoc -o
exemple.fodt
find_converter: Could not find a converter to handle transform:
document (RuntimeError)
```

Boom! Nous avons une erreur : bien que notre fichier Asciidoctor soit vide, notre thème ne peut être utilisé car il lui manque, comme nous l'indique l'erreur, le fichier **document**, qui est le fichier squelette d'un thème Asciidoctor, qui doit avoir comme nom **document.BACKEND.MOTEUR-TEMPLATE**. Dans notre cas il doit donc s'appeler **document.fodt.slim**, fichier que l'on va commencer par créer vide :

```
$ touch theme/slim/fodt/document.fodt.slim
```

Si vous relancez la commande, vous n'aurez plus d'erreur et un fichier **exemple.fodt** aura été généré. Le fichier de sortie est bien sûr vide, car **document.fodt.slim** est vide, pour l'instant.

À quoi sert **document.fodt.slim** ? Voyez cela comme le squelette de tout document qui sera généré par votre thème. Si vous avez l'habitude de générer du HTML avec un système de *templates*, dans ce cas **document.fodt.slim** serait le fichier dans lequel vous définiriez tout ce qui est commun aux fichiers HTML à générer par votre thème : les balises **html** et **head**, et accessoirement l'entête et le pied de page. Et dans **exemple.adoc** vous auriez pour contenu la balise **body** de votre page.

Pour notre thème générant du FODT, il faut lire le code de **gnulinuxmagazine.fodt** et en extraire ce qui est commun pour le mettre dans **document.fodt.slim** au format Slim, en enlevant le contenu spécifique à cet article modèle. Par exemple, vous mettrez au format Slim le contenu des balises **<office:meta>**, **<office:setttings>**, **<office:scripts>** et surtout **<office:styles>** qui définit tous les styles personnalisés de votre document, propre à GLMF. Par exemple, dans **gnulinuxmagazine.fodt** on trouve la définition suivante pour le style gras :

```
<style:style style:name="gras" style:family="text">
  <style:text-properties fo:color="#0000ff" fo:font-weight="bold"
fo:background-color="transparent"/>
</style:style>
```

Où devez-vous vous arrêter dans ce que vous devez mettre dans **document.fodt.slim** depuis **gnulinuxmagazine.fodt** ? Le contenu d'un fichier FODT commence après la balise XML **< office:body><office:text>**, comme on le voit dans le document exemple **gnulinuxmagazine.fodt**, où le contenu de l'article commence après la déclaration de ces balises (vous pouvez ignorer les balises **office:forms** et **text:sequence-decls**) :

```
<office:body>
  <office:text text:use-soft-page-breaks="true">
    <office:forms form:automatic-focus="false" form:apply-design-mode="false"/>
    <text:sequence-decls>
      <text:sequence-decl text:display-outline-level="0"
text:name="Illustration"/>
      <text:sequence-decl text:display-outline-level="0" text:name="Table"/>
      <text:sequence-decl text:display-outline-level="0" text:name="Text"/>
      <text:sequence-decl text:display-outline-level="0" text:name="Drawing"/>
    </text:sequence-decls>
    <text:p text:style-name="Heading">Écrire son thème avec Asciidoctor</text:p>
...
```

La dernière ligne de ce listing est le titre d'un article GLMF, avec comme style ODT **Heading**. Convertie en Slim, cette partie commune sera à ajouter à **document.fodt.slim**. Dans notre exemple, pour le modèle d'article GLMF, mon fichier de *template* **document.fodt.slim** finit donc par :

```
office:body
  office:text text:use-soft-page-breaks='true'
    office:forms form:automatic-focus="false" form:apply-design-mode="false" /
    text:sequence-decls
      text:sequence-decl text:display-outline-level="0" text:name="Illustration" /
      text:sequence-decl text:display-outline-level="0" text:name="Table" /
      text:sequence-decl text:display-outline-level="0" text:name="Text" /
      text:sequence-decl text:display-outline-level="0" text:name="Drawing" /
    text:p text:style-name="Heading"
      =(attr :doctitle)
    =content
```

On voit que j'ai remplacé la balise **<text:p>** de style **Heading** pour générer le titre, par quelque chose de plus

dynamique, en récupérant le titre depuis **exemple.adoc**, comme nous le verrons plus tard.

De son côté, à quoi sert **=content** ? Quand vous exécutez la commande **asciidoctor** sur un fichier **exemple.adoc** en utilisant votre thème, le moteur d'Asciidoctor va commencer par créer le fichier de sortie **exemple.fodt** en collant le contenu de **document.fodt.slim**. Arrivé à la fin de ce *template*, le moteur tombe sur **=content**, et va ensuite lire le fichier **exemple.adoc** pour remplacer l'appel à **=content** par le contenu de **exemple.adoc** dans le fichier **exemple.fodt** généré.

Il nous reste maintenant à commencer à rédiger notre fichier de contenu **exemple.adoc**, et voir comment peupler notre dossier de thème avec des *templates*. On peut déjà commencer par ajouter un titre à notre fichier **exemple.adoc**, en ajoutant au tout début, la simple ligne suivante :

```
= Titre de mon document
```

Quand vous allez exécuter **asciidoctor**, le moteur Asciidoctor va récupérer **Titre de mon document** dans la variable **doctitle**. Arrivé à la directive **=(attr :doctitle)**, le moteur va remplacer cet appel par la valeur de **doctitle**, et générer le contenu ODT suivant dans **exemple.fodt** :

```
...
<text:p text:style-name="Heading">Titre de mon document</text:p>
...
```

Nous avons vu le fichier **document.fodt.slim**, squelette et fichier le plus important de votre thème. Voyons maintenant comment utiliser d'autres syntaxes Asciidoctor dans votre **exemple.adoc**, et pouvoir les traiter avec votre thème.

2.2 Sections

Vous aurez forcément besoin de créer des sections dans votre document. Dans la syntaxe Asciidoctor, pour définir une section et son titre on utilise au moins deux signes **=** (un unique signe **=** est réservé au titre principal du document), et un signe **=** supplémentaire pour une sous-section, jusqu'à quatre niveaux de sous-sections. Il ne nous reste plus qu'à essayer, en créant des sections suivant la syntaxe Asciidoc dans **exemple.adoc** :

```
= Titre de mon document
== Partie 1
=== Partie 1.1
==== Partie 1.1.1
=== Partie 1.2
== Partie 2
```

Exécutons **asciidoctor** :

```
$ asciidoctor --trace -T theme/slim -b fodt exemple.adoc -o
exemple.fodt
find_converter: Could not find a converter to handle transform:
section (RuntimeError)
```

Ok, il manque un fichier **section.fodt.slim** à notre thème, mais que doit-il générer? Si on regarde le fichier modèle **gnulinuxmagazine.fodt**, on voit que les différents titres de section sont stylisés comme suit :

```
...
<text:p text:style-name="Heading_20_1">Partie 1</text:p>
...
```

Et les sous-sections doivent utiliser des styles **Heading_20_2**, **Heading_20_3**, etc. Ce qui nous donne le fichier de *template* slim **section.fodt.slim** :

```
- style = "Heading_20_#{level}"
text:p text:style-name="#{style}"
  =title
=content
```

Dans ce fichier Slim, j'utilise deux attributs qui sont créés par le moteur Asciidoctor quand il traite votre fichier **exemple.adoc** : **level**, qui est le niveau de votre section, de 1 à 4, et **title**, qui contient le texte du titre de votre section (ce qui est placé après les caractères **=** dans le fichier source **adoc**).

On découvre aussi une nouvelle syntaxe dans ce fichier Slim, une ligne débutant par un signe **-** : cela indique que ce contenu ne doit pas être traité comme du texte à générer, mais comme du Ruby à évaluer. Dans notre fichier de *template*, je crée une variable **style**, qui va avoir comme valeur le style FODT **Heading_20_X**, où **X** est le niveau de la section. Ensuite, on crée un paragraphe avec comme style **Heading_20_X**, et comme titre le contenu de **title** (appel **=title**). La dernière ligne **=content** indique que l'on a fini de traiter notre section, et que l'on peut passer à la suite du contenu de **exemple.adoc**.

Si vous exécutez **asciidoctor**, le fichier de sortie **exemple.fodt** devrait désormais ressembler à :

```
...
<text:p text:style-name="Heading">Titre de mon document</text:p>
<text:p text:style-name="Heading_20_1">Partie 1</text:p>
<text:p text:style-name="Heading_20_2">Partie 1.1</text:p>
<text:p text:style-name="Heading_20_3">Partie 1.1.1</text:p>
<text:p text:style-name="Heading_20_2">Partie 1.2</text:p>
<text:p text:style-name="Heading_20_1">Partie 2</text:p>
...
```

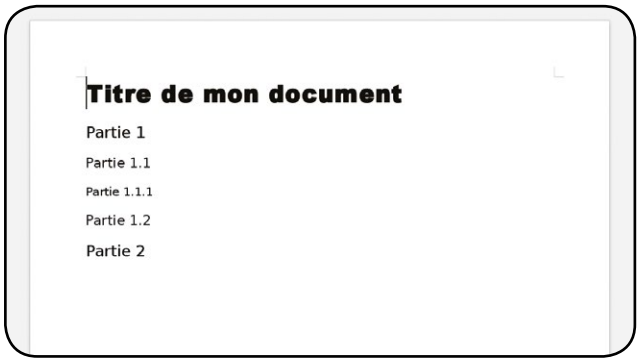


Fig. 1 : Création de sections.

Et en l'ouvrant dans LibreOffice/OpenOffice, vous devriez avoir quelque chose de similaire à la figure 1.

2.3 Blocs de texte

Maintenant que nous savons comment créer des sections, il faudrait commencer à ajouter du texte.

2.3.1 Paragraphe

Rajoutez un simple paragraphe, qui consiste en une ligne, à votre **exemple.adoc** :

```
...
== Partie 2

Ceci est un premier paragraphe.
```

Exécutons **asciidoctor** :

```
$ asciidoctor --trace -T theme/slim -b fodt exemple.adoc -o
exemple.fodt
find_converter: Could not find a converter to handle transform:
paragraph (RuntimeError)
```

On voit que notre thème a besoin d'un fichier de *template* Slim **paragraph.fodt.slim**. Dans un thème Asciidoctor, ce fichier est utilisé pour générer un paragraphe de texte tout simple, sans style, depuis un fichier source **adoc**.

Si nous lisons les sources de **gnulinuxmagazine.fodt**, un tel paragraphe doit utiliser le style **Normal**, et ressembler à cela :

```
<text:p text:style-name="Normal">Ceci est un premier paragraphe.</text:p>
```

Rien de bien compliqué, le fichier **paragraph.fodt.slim** de *template* Slim de notre thème est donc :

```
text:p text:style-name="Normal"
=content
```

Vous pouvez ré-exécuter **asciidoctor**, le fichier généré **exemple.fodt** devrait inclure notre modeste paragraphe :

```
...
<text:p text:style-name="Heading_20_1">Partie 2</text:p>
  <text:p text:style-name="Normal">Ceci est un premier
paragraphe.</text:p>
</office:text>
```

2.3.2 Styles de bloc

Votre thème aura sûrement besoin de mettre en forme des blocs de texte, comme j'en ai eu besoin pour mon thème afin de correspondre aux styles ODT de GLMF : **code**, **console**, **legende**, **note**, **pragma**, etc. Ça tombe bien, la syntaxe Asciidoctor propose une syntaxe variée pour délimiter des blocs de texte aux styles différents **[3]**.

Nous allons prendre l'exemple des affichages des listings de code source et de sortie de terminal. Les styles FODT que nous désirons appliquer sont respectivement **code** et **console**, comme on peut le voir dans **gnulinuxmagazine.fodt** :

```
<style:style style:name="code" style:family="paragraph" style:parent-
style-name="Normal" style:next-style-name="Normal" style:master-page-
name="">
  <loext:graphic-properties draw:fill="solid" draw:fill-color="#ffffcc"/>
  <style:paragraph-properties fo:margin-top="0cm" fo:margin-bottom="0cm"
loext:contextual-spacing="false" style:page-number="auto" fo:background-
color="#ffffcc"/>
  <style:text-properties style:font-name="Courier New" fo:font-
family="&apos;Courier New&apos;" style:font-family-generic="modern"
style:font-pitch="fixed" fo:font-size="8pt"/>
</style:style>

<style:style style:name="console" style:family="paragraph" style:parent-
style-name="Normal" style:next-style-name="Normal" style:master-page-
name="">
  <loext:graphic-properties draw:fill="solid" draw:fill-color="#000000"/>
  <style:paragraph-properties fo:margin-top="0cm" fo:margin-bottom="0cm"
loext:contextual-spacing="false" fo:line-height="100%" style:page-
number="auto" fo:background-color="#000000"/>
  <style:text-properties fo:color="#ffffff" style:font-name="Courier
New" fo:font-family="&apos;Courier New&apos;" style:font-family-
generic="modern" style:font-pitch="fixed" fo:font-size="8pt"/>
</style:style>
```

Il existe dans la syntaxe Asciidoctor des blocs de texte **listing** et **source**, que nous allons employer car leur utilisation dans la syntaxe Asciidoc correspond au résultat que

nous souhaitons obtenir. Pour définir et utiliser de tels blocs dans notre fichier **exemple.adoc**, on crée un bloc de texte délimité par quatre **-**, et de nom **source** ou **listing**. Pour reprendre les exemples de code source et sortie console donnés dans les fichiers exemples de GLMF, ça nous donne :

```
...
Ceci est un premier paragraphe.

[source]
----
# Ceci est du code
# Une entrée typique de menu.lst
# Le style employé est "code"
title= DEBIAN GNU/Linux 2.6.24 (hdc3 - ext3)
kernel= (hd1,2)/boot/vmlinuz-2.6.24-1-686 root=/dev/hdc3
vga=extended
initusrd (hd1,2)/boot/initrd.img-2.6.24-1-686
----

[listing]
----
$ cat dma
2: floppy
3: parportØ
4: cascade
----
```

Vous pouvez exécuter la génération de notre fichier **exemple.fodt** :

```
$ asciidoctor --trace -T theme/slim -b fodt exemple.adoc -o
exemple.fodt
find_converter: Could not find a converter to handle transform:
listing (RuntimeError)
```

Nous obtenons une erreur car le fichier **listing.fodt.slim** qui traite et transforme ces syntaxes de blocs **source** et **listing** de Asciidoctor n'est pas présent. Ce fichier de *template* Slim doit prendre le contenu du boc texte et lui appliquer le style adéquat.

Analysons, le fichier modèle **gnulinuxmagazine.fodt**, et regardons ce que nous devons générer :

```
<text:p text:style-name="code"># Le style employé est " code "</
text:p>
<text:p text:style-name="code">title= DEBIAN GNU/Linux 2.6.24 (hdc3 -
ext3)</text:p>
<text:p text:style-name="code">kernel= (hd1,2)/boot/
vmlinuz-2.6.24-1-686 root=/dev/hdc3 vga=extended</text:p>
<text:p text:style-name="code">initusrd (hd1,2)/boot/initrd.img-
2.6.24-1-686</text:p>
...
<text:p text:style-name="console">$ cat dma</text:p>
<text:p text:style-name="console">2: floppy</text:p>
<text:p text:style-name="console">3: parportØ</text:p>
```

On voit que dans le cas d'un listing **console** comme d'un listing **code**, chaque ligne du listing est rendue avec une balise **<text:p>**, et le style **code** ou **console**. Ce qui nous donne le fichier **listing.fodt.slim** suivant :

```
- if attr?(:style, 'source')
  - style = "code"
- else
  - style = "console"
- parts = content.split(/\n/)
- parts.each do |part|
  text:p text:style-name="#{style}"
    =part
```

Dans ce fichier de *template* Slim, l'attribut **style** d'un paragraphe est défini par le moteur Asciidoctor lorsqu'il traite un paragraphe. On découpe ensuite le paragraphe en lignes distinctes avec **content.split**, puis chaque contenu de ces lignes est rendu en **text:p text:style-name="#{style}"**.

Ré-exécutez **asciidoctor**, vous devriez avoir un résultat similaire à la figure 2.

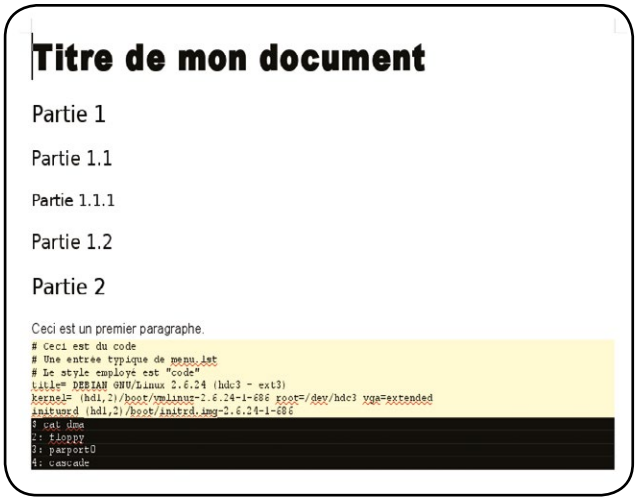


Figure 2 : Création de Listings.

Comme dit précédemment, il existe plusieurs types de blocs texte dans la syntaxe Asciidoctor qui vous permettront de traiter les différents cas de figure auxquels vous devriez avoir à faire face dans la création de votre thème. Souvent vous n'aurez pas une correspondance aussi nette entre le nom du bloc texte Asciidoctor et le style à générer. Par exemple, dans mon thème, pour générer le style ODT **legende** de GLMF, j'utilise de manière détournée un bloc de texte **quote** d'Asciidoctor, dont le style est à définir dans le fichier de *template* **quote.fodt.slim**. Autre exemple, pour générer le style ODT **pragma**, j'utilise la syntaxe d'avertissement Asciidoctor **NOTE**, à définir dans un fichier **admonition.fodt.slim**.

2.4 Formatage de texte

Dans votre thème vous aurez sûrement besoin de mettre en valeur des mots ou chaînes de caractères. Par exemple, dans un article GLMF, nous pouvons avoir besoin de mettre en gras ou encore en italique des mots. Ces différents styles sont définis dans **gnulinuxmagazine.fodt** :

```
<style:style style:name="code_5f_par" style:display-name="code_par" style:family="text">
  <style:text-properties fo:color="#dc2300" style:font-name="Bitstream Vera Sans Mono" fo:font-
family="'Bitstream Vera Sans Mono'" style:font-style-name="Gras" style:font-family-generic="modern"
style:font-pitch="fixed" fo:font-size="9pt" fo:font-weight="bold"/>
</style:style>
<style:style style:name="code_5f_em" style:display-name="code_em" style:family="text">
  <style:text-properties fo:color="#ff0000" style:font-name="Letter Gothic MT" fo:font-
family="'Letter Gothic MT'" style:font-style-name="Gras" style:font-family-generic="modern"
style:font-pitch="variable" fo:font-size="9pt" fo:font-weight="bold"/>
</style:style>
<style:style style:name="gras" style:family="text">
  <style:text-properties fo:color="#0000ff" fo:font-weight="bold" fo:background-
color="transparent"/>
</style:style>
<style:style style:name="italic" style:family="text">
  <style:text-properties fo:color="#6b2394" fo:font-style="italic"/>
</style:style>
<style:style style:name="exposant" style:family="text">
  <style:text-properties style:text-position="super 58%"/>
</style:style>
<style:style style:name="indice" style:family="text" style:parent-style-name="exposant">
  <style:text-properties style:text-position="sub 58%"/>
</style:style>
```

On a donc plusieurs styles de mots et chaînes de caractères que l'on veut appliquer dans un modèle GLMF : **code_par**, **code_em**, **gras**, **italic**, **exposant** et **indice**. Ce sont des styles classiques, et Asciidoctor propose dans sa syntaxe des éléments de syntaxe équivalents (sauf pour **code_par** ou **code_em**) que l'on peut utiliser. Imaginons que dans notre **exemple.adoc** nous ajoutions les lignes suivantes :

```
...
Ce mot sera en *gras*.
Ce mot sera en _italique_.
Ce mot sera en `code_par`.
Voici une puissance 2^n^.
Voici une utilisation d'un indice: H~2~O.
```

Si vous exécutez **asciidoctor** :

```
$ asciidoctor --trace -T theme/slim -b fodt exemple.adoc -o exemple.fodt
find_converter: Could not find a converter to handle transform: inline_quoted
(RuntimeError)
```

Par chance, tous ces différents styles de mots et chaînes de caractères sont à définir dans un unique fichier de template slim **inline_quoted.fodt.slim**. Voyons dans **gnulinuxmagazine.fodt** ce que l'on doit générer comme code FODT :

```
<text:span text:style-
name="gras">TOUJOURS</text:span>
```

Plutôt simple : pour chaque mot ou chaîne de caractères à styliser, on l'encadre dans une balise **<text:span>** avec pour style l'un des styles définis dans ce modèle GLMF (**gras**, **italic**, etc.).

Ce qui nous donne le fichier de *template* **inline_quoted.fodt.slim** suivant :

```
- case @type
- when :strong
  - style = "gras"
- when :monospaced
  - style = "code_par"
- when :emphasis
  - style = "italic"
- when :subscript
  - style = "indice"
- when :superscript
  - style = "exposant"
text:span text:style-name="#{style}"
  =text
```

Dans ce fichier de *template* Slim, on commence par faire un test sur la valeur **type** du mot : **type** est attribué par le moteur Asciidoctor à un mot ou chaîne de caractères quand il est encadré par un élément de syntaxe. Par exemple, **type** vaut **strong** lorsque vous encadrez un mot avec des *backquotes* pour le mettre en gras, **type** vaut **emphasis** lorsque vous mettez un mot en italique en l'encadrant par des _. Dans **inline_quoted.fodt.slim**, il suffit donc de

Figure 3 : Mise en forme de mots.

voir quel est le type du mot en cours de traitement par Asciidoctor, de faire la correspondance avec le nom du style GLMF du fichier ODT et de le mémoriser dans une variable `style`, puis enfin de générer le mot ou chaîne de caractères avec une balise `<text:span>` et le style adéquat.

Ré-exécutez `asciidoctor`, vous devriez avoir un résultat similaire à la figure 3 en ouvrant le fichier obtenu dans OpenOffice/LibreOffice.

Conclusion

Asciidoctor possède un système très avancé pour créer votre thème. Système dont j'ai à peine utilisé une petite partie pour créer le thème qui génère des documents dans le format attendu pour les articles GLMF. Et j'ai à peine présenté une partie des fonctionnalités de ce thème ici, car vous pouvez traiter images, URL, tables, etc. Je vous laisse imaginer l'étendue des possibilités offertes.

Si vous devez créer votre thème, faites-le par étape, en suivant l'approche décrite en début de partie 2. Dans les phases de débogage de vos *templates*, vous aurez peut-être besoin d'afficher tous les attributs disponibles via le moteur d'Asciidoctor en suivant les conseils sur le thème qui permet d'utiliser **RevealJS** comme *backend* d'Asciidoctor **[4]**.

Vous pouvez jeter un œil au thème **[5]** que j'ai développé pour les articles GLMF, afin de vous donner des idées. ■

Références

[1] Asciidoctor Styleheet Factory : http://asciidoctor.org/docs/produce-custom-themes-using-asciidoctor-stylesheet-factory/

[2] *Backends* officiels Asciidoctor : https://github.com/asciidoctor/asciidoctor-backends

[3] Les différents blocs de texte dans la syntaxe Asciidoctor : http://asciidoctor.org/docs/asciidoc-writers-guide/#building-blocks-in-asciidoc

[4] Aide au débogage de template, page du *backend* officiel RevealJS : https://github.com/asciidoctor/asciidoctor-reveal.js/blob/master/HACKING.adoc

[5] Thème Asciidoctor pour les articles GLMF : https://github.com/humboldtux/asciidoctor-diamonded

IL NE LUI MANQUE QUE LA PAROLE...

Tristan Colombo

Nous avons tous déjà entendu cette phrase généralement prononcée par une personne âgée en flattant un chien : « il ne lui manque que la parole ! ». Mais c'est un chien... déjà que son maître parle, on ne va pas en plus devoir entendre l'animal ! Et un ordinateur, ça peut parler ?

Mots-clés : Synthèse vocale, Espeak, SVOX Pico TTS, Google TTS, GNUSpeech, Web Speech, ResponsiveVoice.JS

Résumé

Les solutions de synthèse vocale sont nombreuses sur le marché, qu'il s'agisse de solutions open source ou propriétaires. Pour pouvoir effectuer un choix il faut les tester ! C'est ce qui est fait dans cet article avec le test de six outils qui pourront être utilisés simplement dans vos programmes.

La synthèse vocale, ou TTS pour *Text To Speech* en anglais, est un ensemble de techniques combinant traitement des langues et traitement du signal. Mon objectif n'est pas ici de rentrer dans la théorie de la synthèse vocale mais de comparer les différentes solutions auxquelles nous avons accès de manière à identifier celles procurant les meilleurs résultats en anglais... et si possible également en français. Tous les logiciels présentés dans cet article peuvent être utilisés sur Raspberry Pi sous Raspbian. Pour chacun d'eux je détaillerai leur installation, leur utilisation et bien entendu la qualité des résultats obtenus bien que cette notion soit subjective.

1 Espeak

Commençons par le plus simple de tous, **Espeak**. Présent dans les paquetages des systèmes basés sur Debian, vous pourrez l'installer simplement par :

```
$ sudo apt install espeak
```

Utilisée simplement en lui passant une chaîne de caractères en paramètre, cette commande va la lire en utilisant les paramètres par défaut à savoir langue anglaise et voix masculine :

```
$ espeak "Hello, I am your computer"
```

La voix est un peu métallique mais on comprend ce qui est dit. Tentez maintenant l'expérience en français :

```
$ espeak "Salut, je suis ton ordinateur"
```

Là c'est la catastrophe ! Les phonèmes ne sont pas les mêmes. Pour nous en rendre compte nous allons demander à **espeak** de nous afficher les phonèmes détectés dans la phrase :

```
$ espeak -x "Salut, je suis ton ordinateur"
 s@l'Vt
 dZ'i: s'u:iz t'Vn 'O@dI2na#t'3:
```

Et la même chose en indiquant que la phrase est en français :

```
$ espeak -v fr -x "Salut, je suis ton ordinateur"
 sal'y
 Z@- sy,i tO~n Ordinat'Wr
```

Les phonèmes ne sont pas les mêmes et vous noterez au passage que cette fois-ci, grâce au paramètre **-v fr**, la phrase est plus compréhensible (bien que l'on ait l'impression de se retrouver face à un robot de film de science-fiction des années 1980).

1.1 Réglage de la langue

C'est donc le paramètre **-v** suivi du code ISO 639-1 de la langue sur deux lettres (ou ISO 639-3 sur trois lettres quand le code ISO 639-1 n'existe pas). Pour l'anglais vous pouvez jouer avec les accents grâce à :

- **en** : anglais « standard » ;
- **en-us** : américain ;
- **en-sc** : accent écossais ;
- **en-n** : accent du nord ;
- **en-rp** : prononciation britannique ;
- **en-wm** :accent des Midlands de l'Ouest (Birmingham).

Ne cherchez pas d'équivalents en français, vous ne pourrez pas faire parler votre ordinateur avec l'accent parisien ou marseillais.

L'ensemble des langues disponibles est visible par :

```
$ espeak voices
...
 5  vi-sgn         M  vietnam_sgn          asia/vi-sgn
 5  zh             M  Mandarin             asia/zh
 5  zh-yue         M  cantonese            asia/zh-yue   (yue 5)(zhy 5)
```

1.2 Réglage de la voix

Plusieurs voix sont disponibles et il est possible d'afficher leur liste à l'aide de la commande suivante :

```
$ espeak --voices=variant
Pty Language Age/Gender VoiceName          File          Other Languages
 5  variant       F  female2              !v/f2
 5  variant       F  female3              !v/f3
 5  variant       F  female4              !v/f4
 5  variant       F  female5              !v/f5
 5  variant       F  female_whisper       !v/whisperf
 5  variant       -  klatt                !v/klatt
 5  variant       -  klatt2               !v/klatt2
 5  variant       -  klatt3               !v/klatt3
 5  variant       -  klatt4               !v/klatt4
 5  variant       M  male2                !v/m2
 5  variant       M  male3                !v/m3
 5  variant       M  male4                !v/m4
 5  variant       M  male5                !v/m5
 5  variant       M  male6                !v/m6
 5  variant       M  male7                !v/m7
 5  variant       M  whisper              !v/whisper
 5  variant     70F  female1              !v/f1
 5  variant     70M  croak                !v/croak
 5  variant     70M  male1                !v/m1
```

Pour utiliser ces voix, il faut indiquer le code choisi après l'option **-v**. Si vous souhaitez employer la voix de femme **f2**, il faudra alors taper **-v f2**. Si en plus vous avez modifié la langue, alors il faut séparer le code de la langue de celui de la voix par un signe plus : **-v fr+f2**.

1.3 Réglages divers

1.3.1 Vitesse d'élocution

Il est possible d'accélérer ou de ralentir le débit avec l'option **-s** qui indique le nombre de mots par minute (**160** par défaut). Voici un exemple d'utilisation :

```
$ espeak -v fr+whisper -s 100 "Salut, je suis ton ordinateur"
```

1.3.2 Amplitude

Avec l'option **-a** vous pourrez jouer sur le volume de la voix. La documentation (le man) indique une valeur comprise entre **0** et **20**, par défaut **10**. Après tests la valeur par défaut est en fait de **100** et on peut donc aller bien au-delà. Pour obtenir un sorcier qui chuchote :

```
$ espeak -v fr+whisper -s 100 -a 20 "Salut, je suis ton ordinateur"
```

1.3.3 Pitch

Enfin, l'option **-p** permet de régler le pitch, c'est-à-dire le type d'intonation. Le pitch varie de **0** à **100** (valeur par défaut **50**) et va du plus grave au plus aigu.

```
$ espeak -v fr -s 150 -a 20 -p 2 "Salut, je suis ton ordinateur"
```

1.4 Travailler avec des fichiers

1.4.1 Lire un fichier

L'option **-f** permet d'indiquer un fichier à lire. Créons par exemple le fichier **a_lire.txt** suivant :

```
Ceci est un petit texte à lire. Nous pouvons
tester la ponctuation: les points, les virgules.
```

L'appel se fait ensuite à l'aide de l'option **-f** :

```
$ espeak -v fr+m3 -s 150 -f a_lire.txt
```

1.4.2 Écrire dans un fichier

L'option **-w** permet d'écrire dans un fichier au format wave :

```
$ espeak -v fr+m3 -s 150 -f a_lire.txt -w lu.wav
$ mplayer lu.wav
```

1.5 API

Il n'existe pas d'API dédiée et, quel que soit le langage que vous employez, vous devrez passer par un appel système. En Python on peut faire cela de la manière suivante :

```
01: import subprocess
02:
03: def readText(text, lang='en', voice='m1', speed='160',
amplitude='100', pitch='50'):
04:     options = {
05:         'voice' : '-v' + lang + '+' + voice,
06:         's' : '-s' + speed,
07:         'a' : '-a' + amplitude,
08:         'p' : '-p' + pitch,
09:     }
10:     subprocess.call(['espeak', text, options['voice'],
options['s'], options['a'], options['p']])
11:
12: if __name__ == '__main__':
13:     readText('Salut, je suis ton ordinateur', lang='fr',
voice='f3', speed='150')
```

2 SVOX Pico TTS

Il s'agit d'un moteur TTS sous licence Apache 2.0 qui fonctionne plutôt bien. C'est pratiquement l'équivalent du Google TTS que nous verrons par la suite mais en local, sans avoir besoin d'effectuer une requête sur internet. L'installation de Pico TTS est très simple :

```
$ sudo apt install libttspico-utils
```

Vous disposez alors d'une commande **pico2wave** qui va générer un fichier wave depuis du texte :

```
$ pico2wave -w hello.wav "Hello, I am your computer"
$ mplayer hello.wav
```

Au premier test on s'aperçoit que la diction est tout de suite beaucoup plus fluide qu'avec Espeak. Voyons maintenant s'il est possible d'obtenir un fichier son correct en français.

2.1 Réglage de la langue

L'option **-l** permet de régler la langue en fournissant en paramètre le code souhaité (ISO 639-1). Pour nous il s'agit de **fr-FR** :

```
$ pico2wave -l fr-FR -w salut.wav "Salut, je suis ton ordinateur"
$ mplayer salut.wav
```

C'est beaucoup mieux !

La seule façon d'obtenir les langues disponibles est de faire afficher une erreur :

```
$ pico2wave -l fr-CA -w salut.wav "Salut, je suis ton ordinateur"
Unknown language: fr-CA
Valid languages:
en-US
en-GB
de-DE
es-ES
fr-FR
it-IT
```

Peu de langues sont reconnues... mais pour une fois que le français fonctionne assez bien sans problème, on ne va pas se plaindre !

2.2 S'affranchir du fichier wav

pico2wave ne dispose pas de beaucoup d'options de réglage et, comme son nom l'indique, génère un fichier wave qu'il faut ensuite lire. Si vous souhaitez retrouver le comportement de **espeak** permettant de dire un texte sans avoir à taper plusieurs commandes... il va falloir créer un script **picoTTS** qui exécutera les commandes pour nous :

```
01: #!/bin/bash
02:
03: pico2wave -l fr-FR -w /tmp/test.wav "$1"
04: aplay -q /tmp/test.wav
05: rm /tmp/test.wav
```

Nous le rendons exécutable :

```
$ chmod ugo+x picoTTS
```

Il n'y a plus qu'à l'utiliser sur une seule ligne de commande sachant que le script est préconfiguré pour le français et que nous n'avons plus à effacer le fichier **.wav** :

```
$ picoTTS "Salut, je suis ton ordinateur"
```

2.3 API

Nous nous retrouvons dans le même cas qu'Espeak : il faut réaliser un appel système. Le code de la section 1.5 peut être très facilement adapté :

```
01: import subprocess
02:
03: def readText(text):
04:     subprocess.call(['picoTTS', text])
05:
06: if __name__ == '__main__':
07:     readText('Salut, je suis ton ordinateur')
```

3 GNUSpeech

Petit nouveau dans la famille des moteurs TTS (première version officielle le 14 octobre 2015 bien que le projet soit bien plus ancien), GNUSpeech a l'avantage d'être un projet GNU et d'avoir un objectif louable et somme toute parfaitement réalisable : *« The goal of the project is to create the best speech synthesis software on the planet »* (L'objectif du projet est de créer le meilleur programme de synthèse vocale de la planète). Par contre, que donne la comparaison avec Pico TTS, l'objectif n'étant pas déjà forcément atteint... En effet, GNUSpeech revient de loin puisqu'il s'agit d'un port de TTS_Server pour NeXTSTEP. Nous allons tout de même le tester...

3.1 Installation

L'installation est un brin plus compliquée que pour les deux premiers outils :

```
$ wget http://ftp.gnu.org/gnu/gnuspeech/gnuspeechsa-0.1.5.tar.gz
$ tar -zxvf gnuspeechsa-0.1.5.tar.gz
```

Nous vérifions que tous les pré-requis sont installés :

```
$ sudo apt install gcc cmake
```

Nous pouvons ensuite lancer la compilation :

```
$ pkg_dir=$PWD
$ mkdir ../GnuspeechSA-build
$ cd ../GnuspeechSA-build
$ cmake -D CMAKE_BUILD_TYPE=Release $pkg_dir
$ make
```

L'exécutable est **GnuspeechSA-build/gnuspeech_sa**.

3.2 Test

Pour un premier test nous allons utiliser la langue anglaise :

```
$ gnuspeech_sa -c $pkg_dir/data/en -p /tmp/test_param.txt -o /
tmp/test.wav "Hello, I am your computer"
$ aplay -q /tmp/test.wav
Equation Diphone without formula (ignored).
Equation Tetraphone without formula (ignored).
Duplicate word: [articulate]
Duplicate word: [associate]
Duplicate word: [attribute]
Duplicate word: [charro]
...
```

Comme vous le remarquez, tout comme avec Pico TTS, nous enregistrons un fichier wave que nous jouons par la suite. La suite des tests a nécessité quelques recherches, la documentation étant, à l'heure actuelle, plus que succincte (pour ne pas dire totalement absente).

3.3 Réglage de la langue

Vous y avez cru ? GNUSpeech a déjà du mal à se débrouiller avec l'anglais, n'imaginez donc pas qu'une autre langue soit disponible !

3.4 Réglage de la voix

Pour changer de voix, éditez le fichier **gnuspeechsa-0.1.5/data/en/trm_control_model.config**, commentez la ligne **voice_name = male** et dé-commentez une autre voix :

```
# Configuration file for the TRM Control Model.

# Hz
control_rate = 250.0

#voice_name = male
#voice_name = female
#voice_name = large_child
#voice_name = small_child
voice_name = baby

#
tempo = 1.0
...
```

Le test passe toujours par la même commande :

```
$ ./gnuspeech_sa -c $pkg_dir/data/en -p /tmp/test_param.txt -o
/tmp/test.wav "I'm blue, dabadi dabada"
$ aplay -q /tmp/test.wav
```

C'est malheureusement toujours aussi ridicule... voire pire !

Le fichier **trm_control_model.config** vous permet de régler d'autres paramètres (tempo, etc.). Le fichier **gnuspeechsa-0.1.5/data/en/trm.config** permet d'effectuer d'autres réglages.

3.5 Lire le contenu d'un fichier

L'option **-i** permet de spécifier le nom d'un fichier texte à lire. À ne tenter que par les plus courageux d'entre vous ou par ceux désireux de faire un saut dans le passé de la synthèse vocale.

4 Web Speech API en html5

La Web Speech API est en statut *draft* **[1]** et pour pouvoir la tester nous allons installer la version *nightly build* de Firefox :

```
$ wget https://archive.mozilla.org/pub/firefox/nightly/latest-
mozilla-central/firefox-48.0a1.en-US.linux-x86_64.tar.bz2
$ bunzip2 firefox-48.0a1.en-US.linux-x86_64.tar.bz2
$ tar -xvf firefox-48.0a1.en-US.linux-x86_64.tar
$ cd firefox
$ ./firefox
```

Vérifiez que le paramètre **media.webspeech.synth.enabled** soit à **true** en vous rendant sur l'url about:config et en tapant le nom du paramètre dans la barre de recherche.

Attention !

En installant la *nightly build* de Firefox, si vous utilisez déjà Firefox en tant que navigateur et que vous avez configuré des groupes d'onglets, ceux-ci seront transformés en marques-pages... Sauvegardez votre répertoire **~/.mozilla** pour pouvoir le restaurer par la suite.

Comme il ne s'agit que d'un test consistant à montrer que cette solution ne donne pas de résultat particulièrement convaincant, je vais insérer le code javascript directement dans ma page html (oui, c'est sale mais on ne s'en resservira que lorsque le système sera un peu plus au point et d'ici là l'API aura certainement été modifiée...) :

```
01: <doctype html>
02: <html lang="fr">
03:   <head>
04:     <title>Test Web Speech API</title>
05:     <meta charset="utf-8" />
06:   </head>
07:   <body>
08:     Petit test de synthèse vocale...
09:     <script>
10:       var msg = new SpeechSynthesisUtterance();
11:       var voices = window.speechSynthesis.getVoices();
12:       console.log(voices);
13:       msg.voice = voices[33];
14:       msg.voiceURI = 'native';
15:       msg.volume = 1;
16:       msg.rate = 0.8;
17:       msg.pitch = 1;
18:       msg.text = 'Salut, je suis le navigateur';
19:       msg.lang = 'fr-FR';
20:
21:       speechSynthesis.speak(msg);
22:     </script>
23:   </body>
24: </html>
```

En ouvrant cette page, vous constaterez que le résultat est très mauvais avec une voix robotisée. En ce qui concerne le code, nous devons créer une instance de **SpeechSynthesisUtterance** (ligne 10) qui nous permettra de paramétrer la synthèse vocale : type de voix (ligne 13), emplacement du service de synthèse vocale (ligne 14), volume (compris entre **0** et **1**, ligne 15), débit de parole (compris entre **0.1** et **10**, ligne 16), intonation (compris entre **0** et **2**, ligne 17), le message à dire en ligne 18 et la langue en ligne 19. Le type de voix est sélectionné depuis le tableau **voices** obtenu en ligne 11 et affiché dans la console développeur grâce à la ligne 12. Cela vous permettra d'explorer ce tableau pour découvrir les différentes langues proposées. Enfin, la ligne 21 permet de déclencher la diction.

Une dernière petite remarque avant d'en finir avec cette solution : on pourrait croire que la ligne 19 permet de sélectionner la langue et la ligne 13 uniquement le type de voix (homme, femme, chien, chat, etc.). Il n'en est rien ! Le type de voix est associé à une langue (en position **33** dans le tableau **voices** nous avons le français) et l'attribut **lang** de l'objet **SpeechSynthesisUtterance** ne sert pas dans ce cas (vous pouvez modifier sa valeur, cela n'impactera pas le programme). Tout comme GNUSpeech, cette solution mérite quelques améliorations...

5 ResponsiveVoice.JS : une alternative à la Web Speech API

Il s'agit ici d'un projet Javascript utilisable dans des projets non commerciaux sous licence CC-BY-NC-ND (*Creative Commons Attribution - Non Commercial - No Derivates 4.0*). Un lien vers le site de ResponsiveVoice.JS doit être ajouté sur la page (notez la correction par rapport au code indiqué sur le site officiel) :

```
<div><a href="http://responsivevoice.org">ResponsiveVoice-
NonCommercial</a> licensed under <a href="http://creativecommons.org/
licenses/by-nc-nd/4.0/"><img title="ResponsiveVoice Text To Speech"
src="https://responsivevoice.org/wp-content/uploads/2014/08/95x15.
png" alt="95x15" width="95" height="15" /></a></div>
```

Les différentes langues disponibles sont listées sur la page http://responsivevoice.org/text-to-speech-languages/. Nous allons utiliser la seule voix française : **French Female**.

```
01: <doctype html>
02: <html lang="fr">
03:   <head>
04:     <title>Test de ResponsiveVoice.JS</title>
05:     <meta charset="utf-8" />
06:     <script src="http://code.responsivevoice.org/responsivevoice.
js"></script>
07:     <script>
08:         function speak()
09:         {
10:          responsiveVoice.speak("Bonjour, je suis le navigateur",
"French Female");
11:         }
12:     </script>
13:   </head>
14:
15:   <body onload="speak()">
16:     <div><a href="http://responsivevoice.org">ResponsiveVoice-
NonCommercial</a> licensed under <a href="http://creativecommons.org/
licenses/by-nc-nd/4.0/"><img title="ResponsiveVoice Text To Speech"
src="https://responsivevoice.org/wp-content/uploads/2014/08/95x15.png"
alt="95x15" width="95" height="15" /></a></div>
17:   </body>
18: </html>
```

Le véritable code déclenchant la diction est succinct puisqu'il s'agit uniquement de la ligne 10. La fonction **speak()** prend en paramètre une chaîne de caractères correspondant au texte à lire, le nom de la voix à utiliser et, éventuellement, en troisième paramètre, un dictionnaire permettant de régler les pitch, rate et volume. Voici un exemple d'utilisation de ce troisième paramètre :

```
responsiveVoice.speak("Bonjour, je suis le navigateur", "French
Female", {volume: 1, rate: 1, pitch: 0.2});
```

Cette solution n'est pas libre et nécessite l'utilisation de technologies Web mais c'est le meilleur rendu que nous ayons pu obtenir dans tous nos tests précédents. Pour pouvoir l'utiliser hors ligne, il suffit de télécharger localement le fichier **responsivevoice.js** :

```
$ wget http://code.responsivevoice.org/responsivevoice.js
```

Vous devrez bien entendu modifier la façon dont le script est chargé en ligne 6 :

```
06:     <script src="responsivevoice.js"></script>
```

6 Google TTS

Il est possible d'utiliser le moteur TTS de Google à l'aide d'une astuce sur l'url du service « translate » : https://translate.google.com/translate_tts?ie=UTF-8&q=Bonjour,%20je%20suis%20le%20navigateur&tl=fr-FR&client=tw-ob. Toutefois, il est plus intéressant d'utiliser un programme tel que **google_speech**. Pour le faire fonctionner, il faudra installer **SoX** (un ensemble d'outils de traitement du son) :

```
$ sudo apt install sox libsox-fmt-mp3
```

Ensuite, **pip3** nous permettra d'installer google_speech :

```
$ pip3 install google_speech
```

L'option **-l** permet de spécifier la langue à employer (**en** pour l'anglais, **fr** pour le français, etc.). Différents réglages sont disponibles via l'option **-e**. Le réglage le plus utile sera celui de la vitesse (**speed**). Voici un exemple d'utilisation :

```
$ google_speech -l fr "Bonjour, je suis ton ordinateur" -e speed 1.1
```

La diction est très bonne, c'est le système employé par Google que vous entendez dans l'application de navigation si vous utilisez un smartphone sous Android. Comme avec Espeak et Pico TTS, il faudra passer par un appel système pour utiliser cet outil depuis un programme. Notez qu'ici la connexion à internet est indispensable !

7 Petite application comparant SVOX Pico TTS et Google TTS

Voici un exemple d'application avec un programme qui nous annoncera le temps du lendemain en recherchant les informations en ligne. Dans cet exemple j'ai utilisé les deux meilleurs moteurs que nous ayons testés soit Pico TTS et Google TTS ainsi que l'API du site de prévisions météorologiques wunderground.com (il faut créer un compte gratuit pour disposer d'une clé).

Avant de nous lancer, voyons à quoi ressemblent les données :

```
from urllib.request import urlopen
import json

f = urlopen('http://api.wunderground.com/api/CLE_A_INDIQUER/
forecast10day/q/France/Marseille.json')
data = f.read().decode('utf-8')
parsed_json = json.loads(data)
print(parsed_json)
```

Les données sont compactées et ressemblent à ça :

```
$ python3 weather.py
{'forecast': {'txt_forecast': {'date': '4:44 PM CET', 'forecastday': [{'icon_
url': 'http://icons.wxug.com/i/c/k/chancerain.gif', 'fcttext': 'Light rain
early. Lows overnight in the mid 40s.', 'fcttext_metric': 'Periods of light
rain early. Low 7C.', 'pop': '100', 'period': 0, 'title': 'Wednesday', 'icon':
'chancerain'}, {'icon_url': 'http://icons.wxug.com/i/c/k/nt_chancerain.gif',
'fcttext': 'Becoming partly cloudy after some evening light rain. Low 42F.
Winds ESE at 10 to 20 mph. Chance of rain 100%.', 'fcttext_metric': 'Becoming
partly cloudy after some evening light rain. Low 6C. Winds ESE at 15 to 30
km/h. Chance of rain 100%.', 'pop': '100', 'period': 1, 'title': 'Wednesday
Night', 'icon': 'nt_chancerain'}...
```

Ce n'est guère lisible... Pour détecter les informations qui nous seront utiles, nous pouvons afficher ces données en les indentant :

```
print(json.dumps(parsed_json,indent=2))
```

Et là c'est beaucoup mieux :

```
$ python3 weather.py
{
  "response": {
    "features": {
      "forecast10day": 1
    },
    "termsofService": "http://www.wunderground.com/weather/api/d/terms.html",
    "version": "0.1"
  },
...
```

Visiblement nous recherchons dans la liste **parsed_json['forecast']['txt_forecast']['forecastday']** un élément contenant une clé **pop** ayant pour valeur **60**. La clé **fcttext** contient alors la description du temps du lendemain.

Nous pouvons maintenant écrire le code complet du programme :

```
01: import subprocess
02: from urllib.request import urlopen
03: import json
04:
05: def readText(text):
06:     subprocess.call(['picoTTS', text])
07:
08: def readTextgTTS(text):
09:     subprocess.call(['google_speech', '-len', text])
10:
11: def getWeather():
12:     f = urlopen('http://api.wunderground.com/api/CLE_A_INDIQUER/
forecast10day/q/France/Marseille.json')
13:     data = f.read().decode('utf-8')
14:     f.close()
15:     parsed_json = json.loads(data)
16:
17:     for weather_data in parsed_json['forecast']['txt_forecast']
['forecastday']:
18:         if weather_data['pop'] == '60':
19:             print(weather_data['fcttext'])
20:             print('Avec SVOX Pico TTS...')
21:             readText(weather_data['fcttext'])
22:             print('Avec Google TTS...')
23:             readTextgTTS(weather_data['fcttext'])
24:             print('Fini!')
25:             break
26:
27: if __name__ == '__main__':
28:     getWeather()
```

Pour ce code j'ai légèrement modifié **picoTTS** de manière à ce que la langue employée soit **en-US**.

La ligne 12 définit l'url du service et la ligne 13 lit les données. En ligne 15 les données de **data** (JSON) sont converties en dictionnaire. Il ne reste plus qu'à détecter la donnée que l'on souhaite faire prononcer (lignes 17 et 18).

Il faudrait rendre paramétrable la langue de **picoTTS** et de la fonction **readTextgTTS()**, la ville pour laquelle on souhaite obtenir la météo, éventuellement le jour et, bien sûr, nous pourrions faire appel à un service de traduction en ligne pour obtenir le texte en français. Cette expérience nous montre une chose : le moteur de Google est quand même au-dessus de Pico TTS...

Conclusion

Tout comme avec la reconnaissance vocale, il existe un fossé entre les solutions libres et les solutions propriétaires (voir http://www.acapela-group.com). L'usage de la synthèse vocale peut être très intéressant pour des applications éducatives s'adressant à de jeunes enfants, pour des applications pouvant s'adapter à un public mal voyant ou simplement pour économiser un temps précieux en générant les voix de personnages d'un jeu vidéo. Nous avons pu tester plusieurs solutions dans cet article et « ressentir » ainsi réellement les différences entre les solutions.

D'un point de vue Web il faut espérer que la prise en charge de la Web Speech API se généralise et s'améliore. En attendant, il faut utiliser des solutions telles que ResponsiveVoice.JS.

En local il y a clairement deux outils qui sortent du lot : Pico TTS et Google TTS. Pico TTS est un peu moins bon que Google TTS mais il est distribué sous licence Apache 2.0 et fonctionne sans connexion internet. Bien sûr, pour un rendu optimal il faudra passer par l'indétrônable Google mais peut-être que GNUSpeech parviendra à rattraper son retard... ■

Référence

[1] Spécifications (*draft*) de la Web Speech API : https://dvcs.w3.org/hg/speech-api/raw-file/tip/webspeechapi.html

Parrot jobs Fair

in Paris & Lausanne !

Ingénieur logiciel de modélisation de drones (h/f)

Développeur de simulateur de drones (h/f)

Ingénieur développement logiciel Wi-Fi bas niveau (h/f)

Technical leader iOS (h/f)

Ingénieur développement C bas niveau (h/f)

Ingénieur développement logiciel vidéo (h/f)

Ingénieur développement logiciel auto (h/f)

Ingénieur web Django / Python (h/f)

Embedded software engineer (h/f)

Ingénieur réseau Linux (Wi-Fi/Streaming) (h/f)

Ingénieur développement logiciel C/C++ (h/f)

Ingénieur développement iOS (h/f)

Ingénieur développement logiciel C drone (h/f)

Ingénieur développement logiciel bas niveau (h/f)

Image processing engineer (h/f)

Développeur front-end (h/f)

Développeur Django (h/f)

Parrot recrute !

Plus de 100 postes ouverts de développeurs R&D, retrouvez toutes nos offres sur :

www.recrute.parrot.com

Vous connaissez quelqu'un ?

Cooptez cette personne et gagnez un Drone Bebop 2 si le recrutement aboutit.

www.recrute.parrot.com/cooptation

Parrot®

Rejoignez les 900 passionné(e)s qui inventent les drones de demain

LE GRAAL À PORTÉE DE MAIN : ÉCRIRE UN INTERPRÉTEUR

Guillaume Saupin [Responsable R&D QosEnergy, Data Scientist]

Dans le précédent article, notre héros, le Lisp, a fait l'objet d'un rappel de ses innombrables qualités. Inspirés par ce modèle, nous avons présenté rapidement le caractère modulaire que nous souhaitions donner à notre langage, puis décrit la structure de données contenant le code parsé, et enfin écrit un parseur récursif descendant. Tout ça n'est pas très utile. Nous allons remédier à ça en ajoutant un interpréteur à notre langage.

Mots-clés : C++, Lisp, LLVM, Compilation, Parseur, Langage

Résumé

Dans ce second article, nous allons donner vie à notre langage en lui offrant un interpréteur, et un premier ensemble de fonctions arithmétiques, tout en revenant en détail sur la gestion de nos modules.

A l'issue de notre premier article, nous avions réussi à mettre au point un premier code nous permettant de transformer une chaîne de caractères en une représentation syntaxique de notre langage.

Très simplement, avec ce premier outil, nous sommes en mesure de transformer `(+ 1 2 (* 3 5))` en un vecteur C++ de `Sexp expr` contenant :

```
expr[0] // SymbolAtom +
expr[1] // RealAtom 1.0
expr[2] // RealAtom 2.0
expr[3] // Sexp contenant :  SymbolAtom *, RealAtom 3, RealAtom 4
```

Ce que nous aimerions faire à présent, c'est parcourir cette expression `expr` pour l'exécuter et obtenir le résultat suivant :

```
15 ;;; =(+ 1 2 (* 3 4))
```

C'est ce que va faire notre interpréteur.

1 Tout est dans l'interprétation

Si nous essayons de réfléchir à la manière dont nous allons exécuter l'expression de notre exemple, et si nous nous rappelons la sémantique du Lisp, c'est-à-dire que le premier terme d'une liste est une fonction et les suivants ses arguments, alors la méthode apparaît.

Il faut simplement évaluer les arguments pour obtenir leur valeur, puis appliquer la fonction à ces valeurs. C'est le fameux cycle *eval/apply* décrit dans SICP **[1]**.

Pour évaluer un argument, c'est-à-dire déterminer sa valeur, nous allons avoir besoin d'un environnement qui nous permettra d'associer à un symbole sa valeur.

1.1 L'environnement

L'une des fonctionnalités essentielles des langages de programmation est la capacité d'associer à un label une valeur. C'est-à-dire qu'il faut pouvoir supporter des variables.

Ces variables peuvent avoir une valeur différente suivant leur place dans le code. Dans l'exemple ci-dessous, la variable **lapin** vaut tantôt **666**, tantôt **42** :

```
(let ((lapin 42))
  (print lapin) ;;; 42
  (let ((lapin 666))
    (print lapin) ;;; 666
    ((+ lapin 33)))
```

La valeur associée à une variable, dans notre cas, un **SymbolAtom**, dépend donc de son environnement. Nous allons créer un objet **Environnement**, dont le rôle consistera à stocker la valeur associée à un symbole. Cet objet devra permettre de gérer des portées, et donc de permettre de lier localement un symbole avec une valeur alors même que ce symbole était déjà lié à une autre valeur.

Une fois sorti de cette portée locale, l'environnement devra être en mesure de restituer l'ancienne valeur au symbole.

On voit donc apparaître ici un mécanisme de pile, avec une carte symbole -> valeur qui viendrait s'ajouter à chaque nouvelle portée. Voici l'interface de cet environnement :

```
#pragma once
#include <map>
#include <vector>
#include <string>
#include <functional>
class Sexp;
class Cell;

template <typename Key, typename Val>
class Env
{
public:
  static std::map<Key,Val> func;
  std::vector<std::map<Key,Val>* > envs;

  std::map<Key,Val> top;

  Val& operator[] (const Key& k);
  Val& operator[] (Key& k);
  typename std::map<Key,Val>::iterator find(const Key& k);
  typename std::map<Key,Val>::const_iterator end() const;
  Env();
  ~Env();
  void addEnvMap(std::map<Key, Val>* env);
  void removeEnv();

  template<typename K, typename V>
    friend std::ostream& operator<< (std::ostream& stream, const
Env<K, V>& env);

};
```

Pour plus de généricité, ce code a été paramétré. La classe a pour paramètre un type pour la clef, et un type pour la valeur associée. Dans notre cas, le type de la clef sera une **string**, et le type de la valeur, une **std::shared_ptr<Cell>**.

Une valeur est associée grâce à l'opérateur **[]** :

```
myEnv['rabbit'] = std::dynamic_pointer_cast<Cell>(RealAtom::New(" 42 " )
```

La valeur associée à un symbole est récupérée de la même manière :

```
std::shared_ptr<Cell> cell = myEnv['lapin'] // contient 42
```

Et enfin, un nouvelle carte peut être ajoutée afin de masquer localement d'anciennes associations :

```
std::map<std::string, std::shared_ptr<Cell> > newEnv;
(*newEnv)['lapin'] = std::dynamic_pointer_
cast<Cell>(RealAtom::New(" 666 " ))
myEnv.addEnvMap(&newEnv);
```

Après quoi, si l'on essaye de récupérer la valeur associée avec lapin, on obtient :

```
std::shared_ptr<Cell> cell = myEnv['lapin'] // contient 666
```

On peut ensuite se débarrasser des associations de cette portée avec :

```
env.removeEnv();
```

Et récupérer à nouveau 42 :

```
std::shared_ptr<Cell> cell = myEnv['lapin'] // contient 42
```

Ce mécanisme de portée est implémenté à l'aide d'une pile, stockée dans le vecteur **envs** dans lequel une nouvelle carte symbole -> cellule peut être ajoutée avec **addEnvMap**. La carte en haut de la pile est utilisée en priorité pour récupérer une valeur. Une fois sortie de la portée, la carte est retirée avec **removeEnv**.

Voyons en détail le code :

```
#include "environment.h"
#include "cell.h"

template <typename Key, typename Val>
Env<Key, Val>::Env()
{
  envs.push_back(&top);
}
```

Le constructeur se contente d'empiler une carte par défaut, qui sera utilisée lorsque l'on est en dehors de toute portée, c'est à dire au niveau du *toplevel*.

```
template <typename Key, typename Val>
Env<Key, Val>::~Env()
{
}

template <typename Key, typename Val>
void Env<Key, Val>::addEnvMap(std::map<Key, Val>* env)
{
  this->envs.push_back(env);
}

template <typename Key, typename Val>
void Env<Key, Val>::removeEnv()
{
  this->envs.pop_back();
}
```

L'ajout et la suppression d'une carte se font à l'aide des méthodes standard des **std::vector**.

```
template <typename Key, typename Val>
Val& Env<Key, Val>::operator[] (const Key& k)
{
  for(auto envIt = envs.rbegin() ; envIt != envs.rend() ; envIt++)
    {
      auto it = (*envIt)->find(k);
      if(it != (*envIt)->end())
  return it->second;
    }
      return (*envs.back())[k];
}

template <typename Key, typename Val>
Val& Env<Key, Val>::operator[] (Key& k)
{

  for(auto envIt = envs.rbegin() ; envIt != envs.rend() ; envIt++)
    {
      auto it = (*envIt)->find(k);
      if(it != (*envIt)->end())
  return it->second;
    }
   return (*envs.back())[k];
}

template <typename Key, typename Val>
 typename std::map<Key,Val>::iterator Env<Key, Val>::find (const Key& k)
{
  for(auto envIt = envs.rbegin() ; envIt != envs.rend() ; envIt++)
    {
       typename std::map<Key,Val>::iterator lIt = (*envIt)->find(k);
  if(lIt != (*envIt)->end())
    return lIt;
    }
  return this->top.end();
}
```

La recherche d'une cellule associée à un symbole se fait, comme nous l'avons précisé plus haut, en commençant par la carte en haut de la pile. Notez la présence des **rbegin** et **rend** qui sont bien pratiques pour parcourir le vecteur en sens inverse.

```
template <typename Key, typename Val>
typename std::map<Key,Val>::const_iterator Env<Key, Val>::end() const
{
    return this->top.end();
}
```

Enfin, la méthode **end** nous sera utile pour déterminer si un symbole n'est associé à aucune valeur.

1.2 L'évaluation

Doté ce cet environnement, nous allons pouvoir passer à l'évaluation. Notre code va s'organiser simplement pour supporter ce mécanisme d'évaluation. La classe mère dont dérive tous nos objets se voit dotée d'une méthode purement virtuelle **eval(CellEnv& env)**. Cette dernière prend en paramètre l'indispensable environnement décrit ci-dessus.

```
struct Cell
{
...
  std::weak_ptr<Cell> weakRef;
  virtual std::shared_ptr<Cell> eval(CellEnv& env) = 0;
...
}
```

Cette méthode va ensuite être surchargée pour chacun de nos quatre types. Rappelez-vous que nous ne gérons que quatre types :

- Les nombres réels ;
- Les chaînes de caractères ;
- Les symboles ;
- Les **Sexp**.

Si l'argument est de type **RealAtom** ou **StringAtom**, alors la valeur associée est simplement et respectivement un réel ou une chaîne de caractères. On parle alors d'auto-évaluation.

Si le type de l'argument est un **SymbolAtom**, alors dans ce cas, la valeur associée est la valeur à laquelle est liée le symbole dans l'environnement courant.

Enfin, si l'argument est de type **Sexp**, alors il faut appeler récursivement la fonction d'évaluation afin d'obtenir sa valeur.

Voyons maintenant en détail comment cela s'articule.

1.2.1 Les cellules s'auto-évaluant

Parmi ces quatre types, il en est deux qui ne nécessitent pas d'environnement, et qui contiennent leur propre valeur. Il s'agit du **RealAtom** et du **StringAtom**.

Lors de leur construction, à l'aide de l'opérateur **New**, nous créons un **weak_ptr**, afin de conserver une référence **weakRef** sur l'objet. Cette référence étant un **weak_ptr**, elle n'incrémente pas le compteur de référence, et ne possède donc pas l'objet. Elle ne peut donc s'opposer à sa destruction.

Dans le cas du **RealAtom**, le code est le suivant :

```
std::shared_ptr<RealAtom> RealAtom::New()
{
  std::shared_ptr<RealAtom> atom(new RealAtom);
  atom->weakRef = atom;
  return atom;
}
```

Par contre, à l'aide de la méthode **lock**, cette référence peut être transformée en **shared_ptr**, qui lui, possède l'objet et s'oppose à sa destruction tant qu'il existe.

```
std::shared_ptr<Cell> RealAtom::eval(CellEnv& env)
{
  return weakRef.lock();
}

std::shared_ptr<Cell> StringAtom::eval(CellEnv& env)
{
  return weakRef.lock();
}
```

Ainsi, l'évaluation des **RealAtom** et des **StringAtom** retourne simplement un pointeur garantissant leur existence.

1.2.2 Les symboles

Les symboles, a contrario, n'ont de sens que par rapport à un environnement. Eux aussi contiennent une référence sous la forme d'un **weak_ptr**, qui est initialisé dans le constructeur **New**.

Le code de leur évaluation est direct et fait appel à notre objet environnement :

```
std::shared_ptr<Cell> SymbolAtom::eval(CellEnv& env)
{
  auto it = env.find(this->val);
  if(it != env.end())
    return it->second->eval(env);
  else
    {
      return weakRef.lock();
    }
}
```

Il cherche tout bêtement la valeur associée au label **val** de ce symbole et le retourne, le cas échéant. Sinon, il retourne le symbole en tant que tel.

1.2.2 Les Sexp

Reste maintenant le cas des **Sexp**. C'est la partie la plus compliquée, puisqu'il s'agit d'appliquer une fonction, précisée par le premier terme de la liste, aux arguments que constitue le reste de la liste.

Il faut donc dans un premier temps retrouver la fonction associée à ce premier symbole. Pour cela, normalement, il faudrait interroger notre environnement pour trouver la fonction associée au symbole.

Cependant, effectuer cette recherche à chaque fois qu'on évalue une fonction est coûteux, et impacte négativement les performances dès que l'on fait beaucoup d'appels.

Pour éviter ce problème, et être le plus efficaces possible, nous allons doter nos *cells* d'un nouvel attribut, **closure**, qui contiendra un pointeur vers la fonction associée à notre symbole.

```
struct Cell
{
...
  mutable std::function<std::shared_ptr<Cell> (Sexp*,
Cell::CellEnv&)> closure;
...
}
```

Cette fonction devra donc retourner un **shared_ptr** vers une **Cell**, et prendra en argument un pointeur vers une **Sexp**, et un environnement. Nous allons donc lui passer la **Sexp** et l'environnement courant.

L'évaluation se fera alors ensuite comme suit :

```
std::shared_ptr<Cell> Sexp::eval(CellEnv& env)
{
  if(this->cells.size() == 0)
    return Cell::nil;

  std::shared_ptr<Cell> cl = this->cells[0];
  //we don't look into env to find closure associated to function symbol in
order to be more efficient. (no lookup)
  if(cl->closure)
    {
      return cl->closure(this, env);
    }
  // this code cannot be placed in a module, as it is used to load module !
  if(cl->val.compare("load") == 0)
    {
      loadHandlers(this->cells[1]->eval(env)->val,
       this->cells[2]->eval(env)->val,
       env);
      std::shared_ptr<Cell> res(StringAtom::New());
      res->val =  "loaded";
      return res;
    }

  // there is no function associated with the first argument
  // => this is a list
  std::shared_ptr<Cell> res(Sexp::New());
  static_cast<Sexp*>(res.get())->cells = this->cells;
  return res;
}
```

Le premier test s'assure que la **Sexp** contient au moins une *cell*. Si ce n'est pas le cas, nous retournons le symbole spécial **nil**, qui n'est autre que le **null** du Lisp.

Si c'est bien le cas, nous récupérons la première cellule, ainsi que la fonction vers laquelle elle pointe. Si cette fonction est non nulle, alors nous l'invoquons et renvoyons son résultat.

Dans le cas où la première cellule contient une fonction nulle, alors cela signifie que nous sommes en présence d'une liste. Nous retournons donc une **Sexp** ayant le même contenu.

Note

Et là apparaît clairement le fait que le code est de la donnée, et inversement, puisqu'une liste est codée avec les mêmes objets C++ que le code. C'est la fameuse homoiconicité du Lisp.

Enfin, le lecteur attentif que vous êtes a bien noté que j'ai sauté, volontairement, quelques lignes de code. Ces quelques lignes vont nous permettre de répondre à une autre question que vous vous posez : mais comment est-ce que j'associe une fonction à mon pointeur **closure** ?

2 Extension modulaire du domaine du langage

En effet, comment ajouter des fonctions à notre langage, qui ne fait pour l'instant pas grand-chose ?

2.1 Chargement des modules

Nous allons ajouter des fonctions justement grâce à ce bout de code :

```
// this code cannot be placed in a module, as it is used to load module !
if(cl->val.compare("load") == 0)
  {
    loadHandlers(this->cells[1]->eval(env)->val,
     this->cells[2]->eval(env)->val,
     env);
    std::shared_ptr<Cell> res(StringAtom::New());
    res->val =  "loaded";
    return res;
  }
```

Celui-ci va nous permettre de charger nos modules, et ajouter entre autres des fonctions. Pour cela, il invoque la méthode **loadHandler**, dont le code suit, et qui prend en argument une librairie partagée et le nom d'une fonction chargeant le code supplémentaire. Côté Lisp, cela donne par exemple :

```
(load "./core.so" "registerCoreHandlers")
```

En C++ ce sera :

```
int loadHandlers(const std::string& lib, const std::string& handlerName,
Cell::CellEnv& env)
{
  void* handle = dlopen(lib.c_str(), RTLD_LAZY);

    if (!handle) {
      std::cout << "Cannot open library: " << dlerror() << '\n';
        return 1;
    }
    // load the symbol
    typedef void (*loader_t)(Cell::CellEnv& env);

    // reset errors
    dlerror();
    hello_t handler = (loader_t) dlsym(handle, handlerName.c_str());
    const char *dlsym_error = dlerror();
    if (dlsym_error) {
      std::cout << "Cannot load symbol: " << dlsym_error <<  std::endl ;
        dlclose(handle);
        return 1;
    }
    // use it to do the calculation
    handler(env);

    return 0;
}
```

Nous utilisons **dlopen** pour charger la bibliothèque, puis **dlsym** pour récupérer la fonction **handler** à invoquer et enfin, nous l'invoquons avec l'environnement.

Charge à cette fonction de faire le nécessaire pour ajouter des fonctionnalités à notre langage.

2.2 Un premier module

Nous avons maintenant un interpréteur, un moyen pour charger des extensions depuis notre Lisp. Cette base peut sembler bien faible, mais elle est en fait très puissante, et nous allons à partir d'elle construire tout notre langage, en allant jusqu'à intégrer un compilateur.

Mais pour l'instant, et pour conclure cet article en douceur, nous allons ajouter un premier module essentiel qui va nous permettre de boucler la boucle et de répondre à la question posée en introduction : que vaut **(+ 1 2 (* 3 5))** ?

C'est l'objet de la librairie **core.so**, dont voici l'essentiel du code :

```
#include <ostream>
#include "cell.h"

extern "C" void registerCoreHandlers(Cell::CellEnv& env)
{
  Cell::t = SymbolAtom::New(env, "t");
  Cell::t->real = 1.;
  Cell::t->val = "t";

  Cell::nil = SymbolAtom::New(env, "nil");
  Cell::nil->real = 0.;
  Cell::nil->val = "nil";
```

Nous ajoutons ci-dessus les symboles **nil** et **t**, qui sont les deux valeurs booléennes faux et vrai.

```
  std::shared_ptr<Atom> plus = SymbolAtom::New(env, "+");
  plus->closure = [](Sexp* sexp, Cell::CellEnv& env) {
    std::shared_ptr<Cell> res = RealAtom::New();
    res->real = 0;
    std::for_each(sexp->cells.begin()+1, sexp->cells.end(), [&]
(std::shared_ptr<Cell> cell){res->real += cell->eval(env)->real;});

    res->val = std::to_string(res->real);
    return res;
  };
```

La fonction **+** est ajoutée en créant un symbole du même nom, et dont la fonction associée, **closure**, itère sur les arguments de la **Sexp**, les évalue, puis les additionne. Noter que ce symbole est créé avec une variante de **New**, qui prend l'environnement ainsi que le nom de la fonction comme argument.

Note

C'est en fait ici que ce fait la partie *apply* du fameux *eval/apply*. La fonction **+** est appliquée au résultat de l'évaluation de ces arguments.

```
  std::shared_ptr<Atom> minus = SymbolAtom::New(env, "-");
  minus->closure = [](Sexp* sexp, Cell::CellEnv& env) {
    std::shared_ptr<Cell> res = RealAtom::New();
    res->real = sexp->cells[1]->eval(env)->real;
    std::for_each(sexp->cells.begin()+2, sexp->cells.end(), [&]
(std::shared_ptr<Cell> cell){res->real -= cell->eval(env)->real;});

    res->val = std::to_string(res->real);
    return res;
  };
```

La fonction **–** est du même tonneau. La seule différence est que le premier terme n'est pas zéro, mais la valeur du premier argument. On en soustrait ensuite tous les termes restants.

Le reste du code est à l'avenant, et permet facilement d'ajouter les opérations *****, **/**, **>** et **<**.

```
  std::shared_ptr<Atom> mult = SymbolAtom::New(env, "*");
  mult->closure = [](Sexp* sexp, Cell::CellEnv& env) {
    std::shared_ptr<Cell> res = RealAtom::New();
    res->real = 1.0;
    std::for_each(sexp->cells.begin()+1, sexp->cells.end(), [&]
(std::shared_ptr<Cell> cell){res->real *= cell->eval(env)->real;});

    res->val = std::to_string(res->real);
    return res;
  };

  std::shared_ptr<Atom> div = SymbolAtom::New(env, "/");
  div->closure = [](Sexp* sexp, Cell::CellEnv& env) {
    std::shared_ptr<Cell> res = RealAtom::New();
    res->real = sexp->cells[1]->eval(env)->real;
    std::for_each(sexp->cells.begin()+2, sexp->cells.end(), [&]
(std::shared_ptr<Cell> cell){res->real /= cell->eval(env)->real;});

    res->val = std::to_string(res->real);
    return res;
  };

  std::shared_ptr<Atom> inf = SymbolAtom::New(env, "<");
  inf->closure = [](Sexp* sexp, Cell::CellEnv& env) {
    std::shared_ptr<Cell> m1 = sexp->cells[1];
    std::shared_ptr<Cell> m2 = sexp->cells[2];

    double a1 = m1->eval(env)->real;
    double a2 = m2->eval(env)->real;

    std::shared_ptr<Cell> res = sexp->weakRef.lock();
    if (a1 < a2)
      res->real = 1.0;
    else
      res->real = 0.;

    res->val = std::to_string(res->real);
    return res;
  };

  std::shared_ptr<Atom> sup = SymbolAtom::New(env, ">");
  sup->closure = [](Sexp* sexp, Cell::CellEnv& env) {
    std::shared_ptr<Cell> m1 = sexp->cells[1];
    std::shared_ptr<Cell> m2 = sexp->cells[2];

    double a1 = m1->eval(env)->real;
    double a2 = m2->eval(env)->real;

    std::shared_ptr<Cell> res = sexp->weakRef.lock();
    if (a1 > a2)
      res->real = 1.0;
    else
      res->real = 0.;

    res->val = std::to_string(res->real);
    return res;
  };
}
```

Conclusion

À l'issue de ce second article, nous avons maintenant à notre disposition un interpréteur, un mécanisme de chargement de modules, et un premier module nous permettant d'exécuter les quatre opérations arithmétiques de base. Bref, nous avons sous la main une superbe calculatrice ;)

Dans le prochain article, nous verrons comment augmenter notre langage pour définir non seulement des fonctions, mais surtout des macros. ■

Référence

[1] ABELSON H., SUSSMAN G. J., et SUSSMAN J., *« Structure and Interpretation of Computer Programs »* : https://mitpress.mit.edu/sicp/full-text/sicp/book/.

Note

Le code de cet article a été développé sous Ubuntu et est disponible sur GitHub : https://github.com/kayhman/llisp

ANDROID CONTEXT : QUEL CONTEXT UTILISER DANS VOTRE APPLICATION ANDROID ?

Ramel Adjibi [Ingénieur en Systèmes d'information, spécialiste techno mobile Android]

Lors du développement d'une application mobile Android, on est souvent confronté au choix du Context afin d'initialiser tel ou tel composant; cela peut être une vue, un adapter etc. Il existe plusieurs Context mis à notre disposition par le framework Android; essayons de voir ensemble quel Context utiliser suivant la situation dans laquelle on pourrait se retrouver :-)

Mots-clés : Android, Java, Android Studio, Context, Activity, View

Résumé

Optimiser son code pour éviter au maximum les fuites de mémoire est un l'un des objectifs d'un bon développeur. Pour ce faire, plusieurs points sont à prendre en compte, comme le choix du `Context` à utiliser dans une situation donnée. Nous verrons ensemble dans la suite de l'article quels sont les `Context` mis à notre disposition et comment s'en servir à bon escient.

Nous pouvons définir le **`Context`** comme étant l'état de l'application à un moment donné ; c'est une entité qui représente un ensemble de variables d'environnement. Le **`Context`** permet par exemple à une activité de pouvoir accéder aux ressources de l'application, à des fichiers stockés en local, à la base de données, à certaines classes, etc.

1 Création d'un Context

Le **`Context`** est créé de façon différente en fonction du composant Android qui s'en sert, ce qui implique que le **`Context`** sera toujours différent suivant le composant qui y accède. Mais de quels composants s'agit-il ?

1.1 Application

Le composant **`Application`** est un singleton s'exécutant dans le processus de votre application. On peut y accéder depuis une activité ou un service via la méthode **`getApplication()`**. On peut aussi y accéder via la méthode **`getApplicationContext()`** lorsque l'on utilise un autre objet qui hérite de **`Context`**. Une seule instance du composant est retournée, quel que soit l'endroit où il est appelé.

1.2 Activité/Service

Ces composants héritent de **`ContextWrapper`**, qui implémente la même API, mais qui fait gérer tous les appels de méthodes par une instance interne de **`Context`**, aussi appelé **`Base Context`**. Toutes les Activités/Services créés ainsi que le **`Base Context`** sont tous uniques par instance.

1.3 BroadcastReceiver

Ce composant n'est pas un **`Context`** à lui tout seul mais le *framework* lui

fournit un **Context** via la méthode **onReceive()** à chaque fois qu'un événement lié est intercepté.

1.4 Le ContentProvider

Comme le **BroadcastReceiver**, ce composant n'est pas un **Context** et en reçoit un à sa création ; ce **Context** peut être accédé via la méthode **getContext()**.

Nous avions dit plus haut que le **Context** permet d'accéder aux ressources de l'application comme les **String** avec la méthode **getResources()** ou comme les *assets* avec la méthode **getAssets()**, ainsi que d'autres ressources système via la méthode **Context.getSystemService()**. Le **Context** est une classe abstraite et les méthodes citées précédemment sont abstraites, ce qui implique qu'elles doivent être définies par les classes qui étendent **Context**, à savoir :

- **ContextWrapper** ;
- **Application** ;
- **Activity**/**Service**.

En regardant l'API, on remarque la hiérarchie des classes présentée en figure 1.

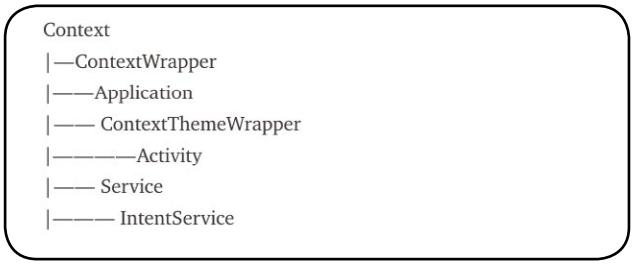


Fig. 1 : Hiérarchie des classes dans l'API

Les sources des classes **ContextWrapper** et **Application** ne nous révèlent aucun indice sur la manière dont est implémenté le **Context**, ce qui signifie que l'implémentation du **Context** est faite par l'OS et est cachée de l'API. Vous pouvez néanmoins voir l'implémentation du **Context** via la classe **ContextImpl** à l'adresse suivante : https://github.com/android/platform_frameworks_base/blob/master/core/java/android/app/ContextImpl.java.

2 Les types de Context

2.1 ContextThemeWrapper

Les choses deviennent intéressantes lorsque l'on regarde les sources de la classe **ContextThemeWrapper**. En voici un extrait :

```
@Override
public Resources getResources() {
 if (mResources != null) {
  return mResources;
 }
 if (mOverrideConfiguration == null) {
  mResources = super.getResources();
  return mResources;
 } else {
  Context resc = createConfigurationContext(mOverrideConfiguration);
  mResources = resc.getResources();
  return mResources;
 }
}
@Override public Resources.Theme getTheme() {
   if (mTheme != null) {
    return mTheme;
   }
   mThemeResource = Resources.selectDefaultTheme(
                         mThemeResource,
                          getApplicationInfo().targetSdkVersion);
   initializeTheme();
   return mTheme;
}
@Override public Object getSystemService(String name) {
 if (LAYOUT_INFLATER_SERVICE.equals(name)) {
  if (mInflater == null) {
   mInflater = LayoutInflater.from(
                      getBaseContext()).cloneInContext(this);
  }
  return mInflater;
 }
 return getBaseContext().getSystemService(name);
}
```

On remarque dans ce code que le **ContextThemeWrapper** utilise le thème de l'application et implémente ensuite la méthode **getBaseContext()** qui est utilisée dans le **Context**.

2.1.1 La méthode getBaseContext() ?

La méthode **getBaseContext()** retourne le **Context** encapsulé par la classe **ContextWrapper** ; cela peut être un composant **Activity** ou **Application**, ou autre... Comme le **ContextWrapper** est implémenté par des dizaines d'autres classes/composants, le résultat de la méthode **getBaseContext()** peut ne pas être connu par avance si l'on ne fait pas attention ; c'est pour cela que je recommanderais plutôt d'utiliser **getContext()** ou **l'Activity**, **l'AppCompatActivity**, etc. Comme cela on sait exactement quel **Context** est utilisé ; il est toujours bon de savoir :-) Cela permet aussi de savoir que l'on a sauvegardé une référence vers un objet dont on connaît la durée de vie et/ou que l'on peut diagnostiquer en cas de fuite de mémoire. Le fait de savoir quel **Context** est utilisé permet aussi de pouvoir se servir des méthodes de la classe concernée par le **Context**.

2.1.2 Le thème de l'application et le Context de votre composant

Si vous voulez toujours appliquer le thème de votre application à votre composant (par exemple **Activity**), il est préférable d'utiliser son **Context** avec **this** le plus souvent. Nous allons illustrer cela avec des exemples de code.

Créons une liste avec le **Context getApplicationContext()**. Choisissez un thème *Light* afin d'avoir le texte par défaut en noir. Créez *l'adapter* de votre liste et passez-lui comme **Context** le **getApplicationContext()** :

```
public class MainActivity extends ListActivity {
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        String[] values = new String[]{"Android", "iPhone", "WindowsMobile",
                "Blackberry", "WebOS", "Ubuntu", "Windows7", "Max OS X",
                "Linux", "OS/2"};
        ArrayAdapter<String> adapter = new ArrayAdapter<>(getApplicationContext(),
                android.R.layout.simple_list_item_1, values);
        setListAdapter(adapter);
    }
}
```

On remarque que les éléments de liste sont blancs (voir figure 2), ce qui n'est pas le résultat attendu avec le thème de l'application.

Changez le code de *l'adapter* et passez-lui plutôt le **Context** du composant, c'est-à-dire **this** au lieu du **getApplicationContext()**. Compilez et admirez le résultat (figure 3) :-)

```
public class MainActivity extends ListActivity {
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        String[] values = new String[]{"Android", "iPhone", "WindowsMobile",
                "Blackberry", "WebOS", "Ubuntu", "Windows7", "Max OS X",
                "Linux", "OS/2"};
        ArrayAdapter<String> adapter = new ArrayAdapter<>(this,
                android.R.layout.simple_list_item_1, values);
        setListAdapter(adapter);
    }
}
```

2.2 Le Context de l'application : getApplicationContext()

La méthode **getApplicationContext()** retourne une instance de la classe **Application**. Cette classe est utilisée pour maintenir un état à un instant « t » de l'application. Alors, quand utiliser le **getApplicationContext()** ?

- Si l'objet qui a besoin du **Context** est par exemple un **Activity** qui doit vivre longtemps, garder une référence d'un tel objet peut amener à des fuites de mémoire; dans ce cas, utiliser le **getApplicationContext()**.
- Si vous avez besoin d'accéder à des ressources utilisant le thème de l'application, comme utiliser des vues ou des *drawables*, vous ne devez pas utiliser le **getApplicationContext()**.

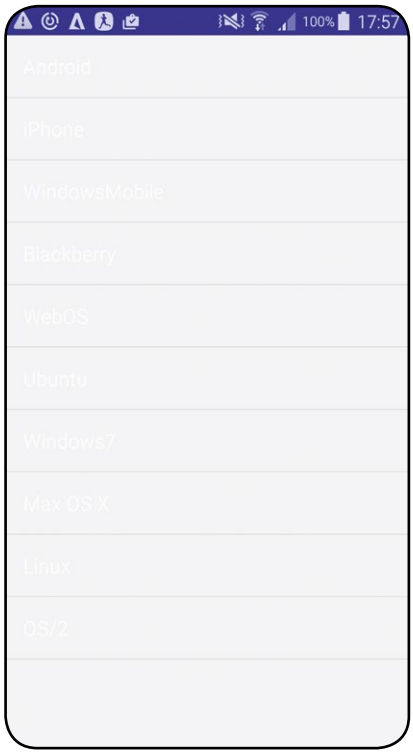


Fig. 2 : Thème avec le Context de l'application

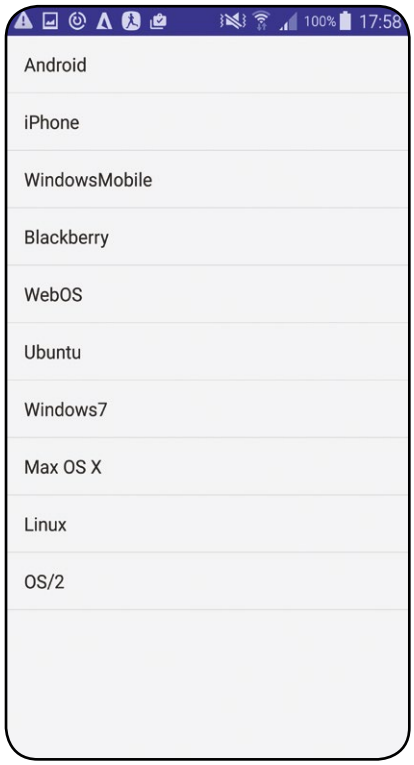


Fig. 3 : Thème avec le ContextThemeWrapper

- Si vous ne voulez surtout pas ignorer le thème de l'application, ne pas utiliser `getApplicationContext()`.
- Quand vous n'êtes pas sûr, utilisez d'abord `getApplicationContext()` et corrigez ensuite.

2.3 getThemeContext()

La méthode `getThemeContext()` est utilisée souvent avec l'`ActionBar`/`ActionBarCompat` lorsque l'on souhaite faire appliquer le thème de l'`ActionBar` aux différentes vues qui le composent.

2.4 getContext()

Le `getContext()` peut être utilisé dans toutes les instances qui ne vivent pas longtemps et n'impliquent pas l'insertion de vues dans `l'ActionBar`. Vous pouvez aussi vous en servir dans des instances qui vivent longtemps, mais il faudrait s'assurer de les passer en `WeakReference` et de tester s'ils sont nuls avant utilisation.

3 Le cas des AlertDialogs

Les `AlertDialogs` peuvent avoir des thèmes différents de l'application, et cela a toujours posé un problème. C'est pour cela que la méthode `getContext()` a été introduite dans le constructeur `AlertDialog.Builder` dans l'API 11 du *framework* Android.

Si vous souhaitez utiliser des thèmes différents pour certaines vues de votre application, il suffirait juste de mettre votre `getContext()` dans un `ContextThemeWrapper` avec le thème voulu.

```
ContextThemeWrapper contextThemeWrapper = new ContextThemeWrapper(this,
              android.R.style.Theme_DeviceDefault_Light_Dialog);
LayoutInflater inflater = getLayoutInflater().cloneInContext(context
ThemeWrapper);
```

La même approche peut être utilisée dans le cas des `AlertDialogs` :

```
AlertDialog.Builder builder = new AlertDialog.Builder(contextThemeWrapper);
```

4 Context en référence

Il peut arriver que l'on ait besoin d'une référence vers un `Context` dans un objet dont le cycle de vie va bien au-delà de celui de l'instance du `Context` ; dans ce cas, on crée

Application	Activity	Service	ContentProvider	BroadcastReceiver
Show a Dialog	NON	OUI	NON	NON
Start an Activity	NON	OUI	NON	NON
Layout Inflation	NON	OUI	NON	NON
Start a Service	OUI	OUI	OUI	OUI
Bind to a Service	OUI	OUI	OUI	OUI
Send a Broadcast	OUI	OUI	OUI	OUI
Register BroadcastReceiver	OUI	OUI	OUI	OUI
Load Resource Values	OUI	OUI	OUI	OUI

Fig. 4 : Utilisation du Context

un singleton qui contiendra une référence vers le **Context** qui servira plus tard à accéder aux ressources dont on aura besoin. Illustrons cela par un exemple de création de singleton :

```
public class ResourcesManager {
    private static ResourcesManager sInstance;
    public static ResourcesManager getInstance(Context context) {
        if (sInstance == null) {
            sInstance = new ResourcesManager(context);
        }
        return sInstance;
    }
    private Context mContext;
    private ResourcesManager(Context context) {
        mContext = context;
    }
}
```

Le problème avec ce singleton est que le **Context** peut venir de n'importe où; cela peut venir d'un **Activity**, d'un **Service** ; etc. Cela nous amènerait alors à garder en référence une instance d'un objet qui a lui aussi en référence d'autres objets comme des vues ou autres objets occupant de l'espace mémoire ; il y a un risque potentiel de fuite de mémoire dans ce cas car le fait de garder ce genre de référence fait que le *garbage collector* ne supprimera pas cet objet qui aura probablement un très long cycle de vie. Pour pallier à ce problème, il faudrait que notre singleton puisse plutôt avoir une référence vers le **Context** de l'application via la méthode **getApplicationContext()**. Illustrons cela par ce bout de code :

```
public class ResourcesManager {
    private static ResourcesManager sInstance;
    public static ResourcesManager getInstance(Context context) {
        if (sInstance == null) {
            sInstance = new ResourcesManager(context.
getApplicationContext());
        }
        return sInstance;
    }
    private Context mContext;
    private ResourcesManager(Context context) {
        mContext = context;
    }
}
```

Dans ce nouveau singleton, quelle que soit l'instance, nous aurons toujours une référence vers le **Context** de l'application, qui est à son tour un singleton ; cela nous éviterait bien des problèmes de fuite de mémoire.

La figure 4 montre un tableau indiquant dans quel cas on peut utiliser tel ou tel **Context**.

Conclusion

Dans la plupart des cas, vous pouvez utiliser le **Context** du composant avec lequel vous travaillez ; vous gardez dans ce cas une référence vers ce **Context** aussi longtemps que votre instance vit. Mais dès que vous ressentez le besoin de devoir utiliser un **Context** bien au delà du cycle de vie de son composant, il est préférable d'utiliser le **Context** de l'application. J'espère avoir été le plus claire possible sur comment choisir le **Context** à utiliser lors de vos développements... maintenant, allez faire de belles applications performantes :-) ■

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

FORK ME ON GITHUB

KEEP CALM AND PULL REQUEST

Contribuez à la plateforme **collaborative** nouvelle génération !

github.com/linagora/openpaas-esn

OpenPaaS

www.open-paas.org