N°189 janvier 2016

GNU LINUX MAGAZINE / FRANCE

DÉVELOPPEMENT ET ADMINISTRATION SUR SYSTÈMES OPEN SOURCE

DEV / PHP

11 ans après... PHP 7 ! Trop tard ?

p.06

JAVASCRIPT / FRAMEWORK

TWITTER BOOTSTRAP SERAIT-IL DÉPASSÉ ?

DÉCOUVREZ ET UTILISEZ GOOGLE POLYMER, UN FRAMEWORK WEB VRAIMENT MODULAIRE !

p.76

OPENSTREETMAP / AUTONOMIE

Calculez vous-même vos trajets et géocodez des adresses

p.36

ALGO / GRAPHES

Parcourez vos graphes en largeur... et trouvez le plus court chemin

p.22

DEV / PHARO

Développez rapidement un serveur et un client de chat

p.68

JAVASCRIPT / DATATABLES

Créez simplement des tableaux HTML interactifs

p.46

HOSTAPD / DNSMASQ

Mettez en place votre point d'accès WiFi

p.42

ET AUSSI : Inspeqtor – Le C et les expressions régulières – La norme MISRA C

L 19275 - 189 - F: 7,90 € - RD

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

JUPITER FILMS présente

SANS LE CHOIX
C'EST VOUS LA PROIE !

Les Nouveaux Loups du web

CONDITIONS GÉNÉRALES D'UTILISATION

J'ACCEPTE

DALLAS INTERNATIONAL FILM FESTIVAL — OFFICIAL SELECTION

OFFICIAL SELECTION SLAMDANCE FILM FESTIVAL

OFFICIAL SELECTION hotDOCS — OUTSPOKEN. OUTSTANDING.

Official Selection Seattle International Film Festival

JUPITER FILMS ET HYRAX FILMS PRÉSENTENT "LES NOUVEAUX LOUPS DU WEB"

AVEC MARGARET ATWOOD DANAH BOYD ORSON SCOTT CARD RAY KURZWEIL DOUGLAS RUSHKOFF MOBY ELI PARISER SHERRY TURKLE ET MARK ZUCKERBERG

DIRECTEUR DE LA PHOTOGRAPHIE BEN WOLF — IMAGE CULLEN HOBACK — IMAGE ADDITIONNELLE VINCE SWEENEY — MUSIQUE JOHN M. ASKEW — MONTAGE CULLEN HOBACK — CONSULTANT MONTAGE GEOFF RICHMAN — ANIMATIONS RYAN KRAMER CHRIS ALLISON — PRODUCTEUR ASSOCIÉ BEN WOLF

PRODUIT PAR CULLEN HOBACK JOHN RAMOS NITIN KHANNA — PRODUCTEURS EXÉCUTIFS JAY WALIA KARAN KHANNA NITIN KHANNA JASWINDER GROVER — RÉALISÉ PAR CULLEN HOBACK

AU CINÉMA LE 6 JANVIER

jupiter-films.com

GNU/Linux Magazine France
est édité par Les Éditions Diamond

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

Directeur de publication : Arnaud Metzler
Chef des rédactions : Denis Bodor
Rédacteur en chef : Tristan Colombo
Responsable service Infographie : Kathrin Scali
Exé PAO : Thomas Pichon
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

L'éducation est ce qu'il y a de plus important dans notre société, car c'est ce qui permet de lutter contre toutes les formes d'obscurantisme. Le savoir est une arme de dissuasion massive permettant de prévenir bon nombre de dérives. Il est donc essentiel de le préserver et de le transmettre aux plus jeunes. Dans ce rôle de transmission, on va trouver bien entendu les parents, puis les musées et certains médias (qui ne seront accessibles que par une volonté parentale) et bien sûr les enseignants. Ces derniers agissent en suivant les directives de leur ministre à savoir la ministre de l'Éducation Nationale.

Dans le cadre du Plan Numérique à l'École, la ministre a signé un partenariat entre le Ministère de l'Éducation Nationale et Microsoft. Avant de nous en émouvoir, penchons-nous sur les termes de ce partenariat consultable par tout un chacun [1]. On trouve dans ce document cinq axes :

1. La rédaction sous l'égide des services de l'État d'une « Charte de confiance » visant à assurer la protection des données personnelles des élèves et des enseignants.
2. L'accompagnement et la formation des acteurs du Plan Numérique : il s'agit de former les décideurs des services du Ministère, les chefs d'établissements et les enseignants à l'utilisation des produits Microsoft (dans le document on parle de « technologies », mais le terme « produits » me semble plus approprié puisque l'on a affaire à des logiciels propriétaires qui n'ont rien à voir avec des technologies au sens où nous l'entendons).
3. La mise à disposition de solutions pour l'utilisation des équipements mobiles. On peut traduire cela par une mise à disposition des produits de Cloud Microsoft (Office 365 Education, Microsoft Azure Active Directory, etc.) avec la mise en place d'une plateforme d'analyse des données d'apprentissage. Bien que, là encore, on nous assure une parfaite confidentialité et sécurité des données, nous ne sommes pas dupes : Microsoft se positionne pour implanter ses solutions et ainsi être à même de posséder un maximum de données qu'il pourra rentabiliser par la suite (comme Google).
4. Une expérimentation pour l'apprentissage du code à l'école. « L'apprentissage du code »... cette expression me fait horreur ! On peut apprendre à programmer ou encore s'initier au développement informatique, mais « apprendre le code »... Quel code ? Le code de l'honneur ? Le code de la route ? Cet apprentissage se déroulera sous la forme d'une expérimentation dans une vingtaine de classes dans une ou deux académies. enseignants seront formés à l'animation des cours (là ce n'est

suite en page 4...

ÉDITORIAL

SUITE

pas précisé dans le document donc je l'ajoute : sur des produits Microsoft) et seront donc ensuite à même, après quelques heures de formation, d'enseigner « le code ». En tant que formateur, j'ai été confronté à de nombreux groupes dans lesquels tous les participants possédaient des connaissances de base en informatique. Dire qu'au bout d'une semaine chacun aurait été capable d'enseigner la matière que je venais de leur inculquer me paraît particulièrement irréaliste, car ils n'avaient pas le recul nécessaire. Les enseignants, ne possédant pas forcément le même bagage technique initial, bénéficieront-ils d'une formation d'une durée suffisante leur permettant d'acquérir le savoir-faire, mais surtout la théorie sous-jacente ? Enseigner ce n'est pas recracher bêtement du savoir comme on pourrait ânonner une comptine incomprise, il faut l'avoir fait sien et être en capacité de donner à l'autre la possibilité de progresser.

5. L'apport d'une aide financière ou technique aux acteurs français de l'e-Éducation (constructeurs de terminaux mobiles sous Windows, éditeurs d'applications éducatives Windows App ou éditeurs de ressources numériques et de manuels scolaires souhaitant intégrer leurs manuels numériques à Office 365 Education).

Que déduire de cette analyse ? Chacun des points énoncés ci-dessus peut être réalisé sur la base de logiciels libres. Microsoft a réussi un grand coup de lobbying, car en se positionnant aussi fortement dans notre système éducatif, il s'assure des gains considérables. En effet, les enseignants formés aux produits Microsoft utiliseront ces produits dans leurs enseignements et les enfants les utiliseront également. Par effet de boule de neige, ayant appris à manipuler des produits Microsoft, nous aurons une nouvelle génération d'utilisateurs Microsoft qui arrivera sur le marché. Il y aura bien quelques enseignants « rebelles », mais cela sera insignifiant comparé à la masse qui aura été endoctrinée. Il est bien connu que la résistance aux changements est un frein important quant à la modification des comportements et en s'attaquant aux plus jeunes, Microsoft est certain de remporter la bataille.

Ce qui est très étrange, c'est que cet accord, fortement favorable à Microsoft, a donné lieu à la publication d'un communiqué de presse en date du 30 novembre 2015 et à une communication tous azimuts du ministère de l'Éducation. Un peu comme si quelqu'un criait sur tous les toits sa joie d'avoir été arnaqué. Est-on certain de la présence de conseillers techniques compétents et intègres dans les ministères ?

Autre partenariat, un peu moins médiatisé, le partenariat entre le Ministère de l'Éducation Nationale et Cisco. Il s'agit cette fois essentiellement de formation par e-learning (Cisco Network Academy) aux technologies réseau et internet. Là c'est le coup de grâce... J'ai déjà du mal à comprendre l'accord avec Microsoft mais là, le Ministère de l'Éducation Nationale, qui a donc sous sa tutelle des Universités dans lesquelles on trouve des maîtres de conférence et des professeurs en informatique va demander à une entreprise externe de former des étudiants et des enseignants ? Et le fait d'utiliser les ressources en interne, c'est impossible ? Qui va choisir les thèmes des formations mises à disposition et les technologies employées ?

Nous ne pourrons bientôt plus nous émouvoir de recevoir des fichiers docx illisibles, car ce sera la norme, ce sera ce que le gouvernement a décidé pour nous. Les solutions libres semblaient être de plus en plus prises au sérieux, mais ces partenariats viennent tout remettre en cause en formatant les adultes de demain. La période de l'évangélisation semblait close, il va falloir repartir en campagne...

Référence

[1] Accord de Partenariat entre le Ministère de l'Éducation Nationale et Microsoft France : http://cache.media.education.gouv.fr/file/Partenaires/17/7/convention_signee_506177.pdf

Tristan Colombo

SOMMAIRE

GNU/LINUX MAGAZINE FRANCE N°189

PHP 7 : ENFIN !

Stéphane MOUREY [Mousse sur le Seeraiwer]

Attention ! C'est tout chaud, ça sort du four ! Vous l'attendiez depuis longtemps : voici PHP 7 !

Mots-clés : Performances, Typage, Générateur, Opérateurs, Exceptions, Classes anonymes

Résumé

Le 3 décembre a été annoncée la sortie officielle tant attendue de PHP 7. Beaucoup de nouveautés importantes sont au programme : amélioration des performances, nouveaux opérateurs, généralisation des exceptions, classes anonymes, typage des valeurs de paramètres et de retour pour les fonctions... Après avoir examiné comment tester cette nouvelle version, nous examinerons en détail ses principaux apports.

Il a fallu 8 *Release Candidate* avant la sortie de la version finale de PHP 7.0.0. Si un seul bug avait été trouvé, une autre *RC* aurait pris le relais au 3 décembre, et ainsi de suite jusqu'à ce que plus aucun bug ne soit relevé pendant une période de quinze jours de tests. Le suspens fut presque insoutenable, mais la qualité de PHP, un de ses objectifs prioritaires aujourd'hui, est à ce prix.

Alors qu'il avait fallu à peine plus de neuf ans pour que PHP passe de sa version 1 à sa version 5, plus de onze ans se sont écoulés avant la suivante, la version 7 (pour ceux qui n'auraient pas suivi : non, il n'y a pas eu de version 6, car c'est bien celle-là qui aura longtemps posé problème). Une si longue attente ne pouvait que créer une grande impatience. Et, enfin, nous y sommes : PHP 7 est (presque) prêt ! Voyons donc de quoi il en retourne.

1 Installation

Il faudra un peu de temps pour que PHP 7 soit intégré dans votre distribution préférée. Une installation par les sources est toujours possible, mais assez complexe dans le cas d'un projet aussi lourd que PHP pour justifier un article à part entière. Heureusement, Zend propose des binaires pour au moins deux des plus grandes familles d'architectures Linux : Debian et Redhat. Enfin, pour ceux qui ne souhaiteraient pas l'installer sur leur système, Zend propose un container Docker spécialement conçu pour vous permettre de tester PHP 7.

Nous n'aborderons ici que la configuration du système pour l'utilisation de PHP 7 sur un serveur **Apache**, laissant de côté toute autre problématique, cas le plus répandu. Pour les autres, vous y glanerez assez d'informations pour trouver comment procéder dans votre situation. Dans un premier temps, vous n'appliquerez cette migration que dans un environnement de test pour voir les corrections à effectuer à votre code, attendant la mise à jour des dépôts officiels.

1.1 Debian, Ubuntu et consorts

En **root**, exécutez les commandes suivantes :

```
# echo "deb http://repos.zend.com/zend-server/early-access/php7/repos ubuntu/" >> /etc/apt/
sources.list
# apt-get update && apt-get install php7-nightly
```

PHP7 est maintenant installé dans **/usr/local/php7**.

Reste à voir la configuration de votre serveur Web. Voyons comment procéder avec Apache. Il faut d'abord copier certains fichiers :

```
# cp /usr/local/php7/libphp7.so /usr/lib/apache2/modules/
# cp /usr/local/php7/php7.load /etc/apache2/mods-available/
```

Ajoutez un fichier de configuration **/etc/apache2/mods-available/php7.conf** pour que Apache ne soit pas perdu :

```
<FilesMatch \.php$>
SetHandler application/x-httpd-php
</FilesMatch>
```

Encore trois commandes pour achever la configuration :

```
# a2dismod mpm_event
# a2enmod mpm_prefork
# a2dismod php5
# a2enmod php7
# service apache2 restart
```

Les commandes **a2dismod** et **a2enmod** vous permettront d'activer et de désactiver PHP 5 et PHP 7.

1.2 Centos, Fedora, Redhat...

En root bien sûr, il vous faut commencer par créer un nouveau fichier de dépôt **/etc/yum.repos.d/php7-nightly.repo** avec le contenu suivant :

```
[zend-php7]
name = PHP7 nightly by Zend Technologies
baseurl = http://repos.zend.com/zend-server/early-access/php7/
repos/centos/
gpgcheck=0
```

Puis, exécutez la commande suivante :

```
# yum install php7-nightly
```

Pour la configuration d'Apache, copiez le bon fichier au bon endroit :

```
# cp /usr/local/php7/libphp7.so /etc/httpd/modules/
```

Puis ajoutez la ligne suivante à **/etc/httpd/conf/httpd.conf** :

```
LoadModule php7_module        /usr/lib64/httpd/modules/libphp7.so
<FilesMatch \.php$>
SetHandler application/x-httpd-php
</FilesMatch>
```

Pensez également à retirer la ligne équivalente pour PHP5 et relancez votre serveur.

1.3 Tester avec Docker

Mais vous préférerez peut-être tester PHP 7 avec Docker. Pour cela, Zend met à votre disposition un container avec la toute dernière version de la nuit précédente :

```
$ git clone https://github.com/janatzend/docker-php7-nightly-
build.git
$ cd docker-php7-nightly-build
$ sudo docker build -t zend/php:7.0 .
$ sudo docker run -d -P zend/php:7.0
```

Cette commande expose le port **80** de votre image avec Apache et PHP 7. Elle affichera l'identifiant du container dont vous aurez besoin pour déterminer l'adresse IP à laquelle consulter votre nouveau serveur. Pour cela, lancez la commande suivante en remplaçant **<id>** par l'identifiant affiché :

```
$ sudo docker logs -f <id>
******************************************
PHP 7 is ready for use

Your application is available at http://172.17.0.11

******************************************
```

Vous n'avez plus qu'à ouvrir votre navigateur à cette adresse.

Pour plus de confort, vous trouverez sans doute agréable de monter un dossier local dans votre container à utiliser comme racine de votre serveur. Vous pourrez le faire en lançant votre container de la façon suivante :

```
$ sudo docker run -d -v /votre/chemin/local/:/www zend/php:7.0
```

2 Performances

L'une des critiques les plus constantes faites à PHP concerne les performances : PHP est lent, trop lent. Ce nouveau cru n'y mettra peut-être pas un terme, il s'agit toujours d'un langage interprété, mais beaucoup de progrès ont été réalisés dans le moteur Zend, le cœur de l'interpréteur, et de nouvelles armes, dont nous verrons le détail plus loin, sont apportées aux développeurs pour optimiser leurs applications, charge à eux de les utiliser.

Pour un même code, PHP 7 est en moyenne deux fois plus rapide que PHP 5. Un progrès de 100 %, ce qui le rend généralement plus rapide que son plus proche concurrent, HHVM, l'implémentation de PHP par Facebook qui a fait tant parler d'elle. Il ne s'agit pas de tests de laboratoires sur du code théorique pour examiner combien de fois telle boucle improbable peut être exécutée en une seconde. Non, les tests ont été pratiqués avec des applications réelles telles que Wordpress ou Joomla. Rien que pour cela, vous avez tout intérêt à migrer.

3 Un peu d'abstractions

L'histoire de PHP fait qu'il s'est longtemps construit comme un énorme bricolage : loin d'un projet industriel d'une société établie, PHP est d'abord le résultat du travail d'un étudiant groenlandais, Rasmus Lerdorf, qui désirait faire des statistiques sur les consultations de son CV en ligne... Durant les vingt années qui ont suivi, l'histoire de PHP fut mouvementée, et il fallut du temps pour atteindre l'exigence de qualité qui est aujourd'hui celle de l'équipe au cœur du développement de PHP, dont Rasmus fait toujours partie. Jusqu'ici, le cœur de PHP était principalement composé de deux éléments : le *parseur*, qui vérifie la syntaxe et analyse, et le compilateur qui l'exécute. Avec le temps, d'innombrables *hacks* ont été ajoutés dans la relation entre le parseur et le compilateur pour contourner telle ou telle difficulté, ajouts au code peu structurés qui le rendent moins consistant et efficace, plus difficile à lire et à maintenir. Pour pallier à ce problème, PHP 7 ajoute l'*Abstract Syntax Tree* ou AST. Il s'agit d'une abstraction supplémentaire qui se place en intermédiaire entre le *parseur* et le compilateur. Elle permet de supprimer tous ces *hacks* et de simplifier les règles d'analyse du *parseur* et assainit fondamentalement les relations entre les deux composants.

La syntaxe de PHP n'en est pas affectée à un détail près : **`list()`** fonctionne d'une façon légèrement différente. Tout d'abord, le premier argument de cette fonction devient obligatoire : cela est sans conséquence puisque, même si cette syntaxe était autorisée, elle n'avait tout simplement pas de sens. Le second point est plus important : il semble abstrait et subtil, mais il s'agit typiquement du genre de choses susceptible de provoquer des effets de bord inattendus et être à l'origine de bugs rares mais particulièrement difficiles à démêler. Donc si vous êtes féru de cette fonction, soyez attentif. Le changement de comportement intervient lorsque **`list()`** est utilisée pour affecter des valeurs d'un tableau : l'ordre d'affection est inversé en PHP 7 par rapport à ce qu'il était en PHP 5.

Ainsi avec le code suivant :

```
list($a[], $a[], $a[]) = [1, 2, 3];
var_dump($a);
```

Vous obtiendrez sous PHP 7 :

```
array(3) {
  [0]=>  int(1)
  [1]=>  int(2)
  [2]=>  int(3)
}
```

Mais il produisait sous PHP 5 :

```
array(3) {
  [0]=>  int(3)
  [1]=>  int(2)
  [2]=>  int(1)
}
```

Notez que le comportement de **`list()`** n'est modifié que lors d'affectations de valeurs à un tableau, pas dans le cas de variables simples et ainsi, le comportement de **`list($a, $b)`** n'est pas affecté.

4 Uniformisation de la syntaxe des variables

La syntaxe des variables a été révisée pour lever certaines inconsistances et autoriser des combinaisons plus élaborées encore que celles que PHP 5 permettaient déjà. Par exemple, vous pourrez faire des appels à des fonctions anonymes affectées à des propriétés comme ci-dessous :

```
$a->myClosure = function() {echo
'test';};
($a->myClosure)(); // affichera "test"
```

Ou encore enchaîner des appels statiques à des membres de classes :

```
class foo { static $bar = 'baz'; }
class baz { static $bat = 'Hello World';
}

baz::$bat = function () { echo "Hello
World"; };

$foo = 'foo';
($foo::$bar::$bat)();
```

Cela permet également d'accéder à un index d'un tableau adressé par une propriété variable... Ce n'est pas clair ? Examinez ce code :

```
01: $p = "myArray";
02: $o->myArray = [1,2];
03: var_dump($o->$p[0]);
```

En PHP 5, l'expression **`$p[0]`** renverra la chaîne de caractères **`"m"`**, car **`$p`** est une chaîne de caractères et PHP autorise d'accéder aux caractères d'une chaîne par leur position en utilisant une syntaxe identique à celle d'un tableau... ce qui peut se révéler pratique dans bien des situations, mais on comprend bien qu'ici ce n'est pas le résultat souhaité. En effet, ce code produira comme résultat :

```
PHP Notice:  Undefined property:
stdClass::$m in /var/www/projects/
LinuxMag/php7/test2.php on line 5
NULL
```

Avec PHP 7, l'interprétation devient plus subtile et comprend que, ici, à la ligne 3, il n'est pas souhaité d'accéder au premier caractère de **`$p`**, mais de l'utiliser comme nom d'une propriété, qui se trouve être un tableau dont on recherche le premier index. On obtient donc :

```
int(1)
```

5 Typage explicite des scalaires

Avec PHP 5, il était déjà possible de préciser le type d'un argument de fonction, mais cela excluait les scalaires. Vous pouviez préciser pour une fonction qu'elle attendait un objet de telle classe ou un tableau, mais pas un entier ou une chaîne de caractères. Cela était regrettable, maintenant, c'est corrigé. Il suffit d'indiquer le type de l'argument avant son nom dans la déclaration de la fonction, comme en C :

```
function vardumpString(string $maChaine)
{
```

Par défaut, PHP va transtyper l'argument s'il n'est pas du bon type (ici, l'entier 1 serait converti en la chaîne de caractères "**1**"), mais il est possible de l'empêcher dans un fichier, ce qui me semble une bonne pratique, en indiquant *à la première ligne* :

```
declare(strict_types=1);
```

Dès lors, plus de transtypage, mais une **TypeException**. On ne peut qu'encourager cette pratique, le transtypage automatique étant l'un des responsables de la lenteur de PHP.

6 Type de retour

Autre grande nouveauté très attendue, le typage de retour. Il est maintenant possible de préciser le type de retour d'une fonction dès sa déclaration. Le type est indiqué après la parenthèse de fermeture de la liste des arguments et avant l'accolade ouvrant le bloc de contenu de la fonction, précédé par double point. Voici un exemple pour être plus clair :

```
function test(string $maChaine) : int {
```

Ici, il est précisé que la fonction **test** renverra un entier. Que se passerait-il si tel n'était pas le cas ? Si, par erreur, elle renvoyait une chaîne de caractères telle que "**1 oiseau**" ? Là aussi, le comportement par défaut est d'effectuer un transtypage pour que la valeur de retour se plie au type déclaré. "**1 oiseau**" deviendrait alors l'entier **1**.

Là aussi, il est souhaitable d'éviter ce comportement en utilisant **declare (strict_types=1)** : une violation lèverait alors une **TypeException**.

7 Classes anonymes

Les classes anonymes permettent d'étendre une classe, d'implémenter une interface et/ou d'utiliser un trait au moment de l'instanciation d'un objet. Cela permet par exemple d'ajouter des méthodes à une classe pour un seul objet en faisant l'économie de l'écriture d'une classe dédiée. Cette façon de faire, inédite en PHP, est pourtant courante avec d'autres langages.

La création d'un objet de classe anonyme s'effectue à travers l'instruction **new class**. Voici un exemple d'utilisation complet :

```
class uneClasse {}
interface uneInterface {}
trait unTrait {}

$unObjet = new class(10) extends
uneClasse implements uneInterface {
    private $num;

    public function __construct($num)
    {
        $this->num = $num;
    }

    use unTrait;
});
```

Une utilisation pourra être la mise en œuvre de services, accessibles via un injecteur de dépendances, plus complexes qu'aujourd'hui où ce rôle est souvent tenu par de simples fonctions anonymes, fonctions qui parfois avaient pour seul rôle de renvoyer un objet...

Un exemple de cela avec un service de journal :

```
interface Logger {
    public function log(string $msg);
}
$di->set('logger', new class implements
Logger {
    public function log($msg)
    {
        echo $msg;
    }
});
```

8 De nouveaux opérateurs

8.1 Opérateur de comparaisons combinées

Rejoignez le côté obscur de la Force en utilisant l'opérateur « vaisseau spatial » (*SpaceShip operator*) ! Ainsi appelé à cause de sa forme qui évoquerait, selon son auteur, le vaisseau de combat rapproché de Darth Vader dans *La Guerre des Étoiles*, il permet une comparaison sophistiquée ayant trois valeurs de retour possibles, **1**, **0** et **-1**, selon que les opérandes soient supérieures, égales ou inférieures l'une à l'autre. Il est particulièrement utile associé aux fonctions de tri telles que **usort()**. Cet opérateur se compose de trois signes : un chevron ouvrant, un égal et un chevron fermant. Ce qui nous donne : **<=>** .

On peut l'utiliser ainsi :

```
1<=>1 // renverra 0
1<=>0 // renverra 1
0<=>1 // renverra -1
```

8.2 Opérateur de fusion

Que de fois avez-vous testé si telle variable était définie pour affecter sa valeur à une autre ou utiliser une valeur par défaut à la place ? Sans doute, trop souvent à votre goût.

Jusqu'ici, la syntaxe la plus élégante consistait à utiliser l'opérateur ternaire dans sa version courte :

```
$a = $valeurRecue?:'valeur par défaut';
```

Malheureusement, ce code lève une erreur de niveau **notice**, dans le cas où la variable testée n'existe pas, ce qui n'est pas souhaitable : nous voulons justement définir une valeur par défaut, qui est censée représenter le cas le plus fréquent.

PHP 7 propose l'opérateur de fusion **??** qui remplace l'opérateur ternaire court. Le fonctionnement est exactement le même, mais aucune erreur n'est levée.

9 Innovations sur les générateurs

9.1 Un retour possible

Avec PHP 5, les fonctions utilisées comme générateurs ne pouvaient renvoyer de valeurs autrement que par l'instruction **yield**. Avec PHP 7, Il est possible de retourner une valeur avec **return**, mais une syntaxe particulière sera nécessaire pour y accéder d'une part, et, d'autre part, elle ne pourra être émise qu'après la dernière itération, devenant un indicateur de fin du processus. Sa valeur est accédée grâce à la méthode **getReturn** de l'objet **Generator** :

```
function gen(){
  yield 1;
  yield 2;
  yield 3;
  return true;
}
$gen = gen();
foreach ($gen as $v) {
  echo $v;
}
echo 'Tout s\'est-il bien passé? ';
echo $gen->getReturn()?'Oui':'Non';
```

Une utilisation prématurée de **getReturn** provoquera la levée d'une exception.

9.2 D'un itérateur,à l'autre...

Un générateur peut se prolonger grâce à une autre structure itérable (tableau, générateur, itérateur). Les valeurs renvoyées sembleront ainsi venir de lui. Il suffit pour cela d'ajouter, après **yield**, l'instruction **from** suivie de l'expression de la seconde structure itérable :

```
function gen(){
  yield 1;
  yield 2;
  yield 3;
  yield from range(4, 7);
  yield from array(8, 9, 10);
}

$gen = gen();
foreach ($gen as $v){
  echo $v;
}
```

Ce code affichera les entiers de **1** à **10**.

10 Généralisation des exceptions

De nouvelles exceptions apparaissent avec PHP 7. Certaines sont liées à de nouveaux contextes comme nous l'indiquons en plusieurs endroits de cet article, mais d'autres progrès ont été faits. Si l'on écarte les erreurs de syntaxe qui bloquent l'analyse du code, la nouvelle mouture de PHP systématise l'utilisation des exceptions comme système de gestion des erreurs par défaut. Ainsi, l'appel à une fonction indéfinie, erreur fatale pour PHP 5, provoquera une **EngineException**, que vous pourrez intercepter et traiter comme bon vous semble.

Remarquez que cette nouvelle classe d'exception n'étend pas la classe standard **Exception**. Et cela afin d'obliger le programmeur à leur offrir un traitement spécial et à ne pas se contenter de les intercepter en même temps que toutes les autres. Ainsi, il vous faudra écrire un code ayant une structure du type :

```
try {
  bla(); // fonction indéfinie
} catch (\EngineException $e) {
  var_dump($e);
} catch (\Exception $e) {
  var_dump($e);
}
```

Si vous utilisez des gestionnaires d'erreurs personnalisés, ceux-ci ne fonctionneront plus correctement, et il vous faudra les corriger pour fonctionner avec PHP 7.

11 Le grand ménage

Une version majeure autorise des changements incompatibles avec les versions précédentes. De nombreuses fonctions présentes dans PHP 5 sont dépréciées depuis plusieurs années, mais conservées, en attente de la prochaine version majeure. Elles sont toutes supprimées avec PHP 7.

L'impact le plus important sera sans doute l'abandon de la vieille bibliothèque MySQL, à laquelle il est recommandé de préférer MySQLi ou PDO depuis bien longtemps maintenant. Mais certains développeurs sont sans doute restés sourds, d'autant plus que quelques débutants risquent fort d'avoir découvert l'utilisation de MySQL depuis PHP au travers de vieux tutoriels surannés, mais toujours en ligne. Bref, malgré la recommandation ancienne, vous n'êtes pas à l'abri de découvrir qu'une de vos applications maison s'appuie encore sur cette bibliothèque. Comme toujours, il convient de migrer avec prudence, après avoir fait une batterie de tests établis à l'avance dans un environnement dédié.

Sachez qu'il était prévu de permettre à cette vieille bibliothèque MySQL de fonctionner avec PHP 7 si nécessaire en l'installant grâce à PECL. Mais à l'heure où j'écris ces lignes, la page [1] dédiée à ce projet semble ne pas laisser beaucoup d'espoir...

Conclusion

PHP 7 s'est fait attendre, mais cela en valait la peine. Les changements apportés ne feront pas taire toutes les critiques, mais nombre d'entre elles n'ont plus lieu d'être. Le type tant des paramètres de fonctions que de leurs valeurs de retour va permettre aux développeurs de produire un code de bien meilleure qualité et plus rapide, surtout si le mode strict se généralise. PHP 7 jette ainsi les bases d'une façon de coder nouvelle pour ses utilisateurs en les mettant sur la voie de développements plus robuste et plus efficace. Longue vie à PHP 7 ! ■

Pour aller plus loin

Vous consulterez utilement la page du site officiel de PHP consacrée à la migration depuis PHP 5.6 vers PHP 7 : http://php.net/manual/fr/migration70.php

Référence

[1] MySQL sur PECL : https://pecl.php.net/package/mysql

Glossaire

▪ **Fonction anonyme** ou **fermeture** ou **clôture :**

Fonction n'ayant pas de nom, qui peut être affectée à une variable pour être manipulée ou utilisée comme retour d'une autre fonction. En anglais, *closure*.

▪ **Scalaire :**

Type de variable simple et non composée tel que les entiers, les décimaux, les chaînes de caractères. Il s'oppose aux types *composés* tels que les objets ou les tableaux.

▪ **Opérateur ternaire :**

Opérateur permettant d'effectuer un test semblable aux instructions `if` et `else` sur une expression de manière synthétique. En PHP, il est composé des signes `?` et `:`. Ainsi, l'expression `$a?1:2` renvoie `1` si **$a** est évalué à vrai, `2` sinon. Il est souvent utilisé pour définir une valeur par défaut d'une variable éventuellement non définie de cette façon : `$a?$a:'valeur par défaut'`, ce qui a mené à la syntaxe équivalente abrégée `$a?:'valeur par defaut'`.

HADOPI : CINQ ANS APRÈS, QUEL BILAN ?

Tris ACATRINEI [Projet Arcadie]

Dans l'avant-dernière chronique, nous avions fait le bilan du secrétariat d'État au numérique. C'est au tour de l'institution la plus détestée de France de passer son examen. Ayant moi-même fait mes classes dans cette institution, il est tout à fait vraisemblable que je sois amenée à manquer de façon flagrante de neutralité.

Mots-clés : Hadopi, Culture, NetFlix, Droit d'auteur, Copie privée

Résumé

Que faire de cette institution devenue l'un des symboles de l'ancienne présidence mais si difficile à supprimer ? Loin de rester confinée dans la sphère qui lui était normalement dévolue – celle de la culture – l'autorité administrative indépendante fait régulièrement parler d'elle, comme pour rappeler à certains qu'elle est encore vivante. Mais pour combien de temps ?

Si vous souhaitez réveiller les convives au cours d'une soirée, dites simplement le mot « Hadopi » : l'effet est instantané, tout le monde ayant une opinion sur cette haute autorité particulièrement décriée, sans pour autant bien en connaître les missions. Petit balisage historico-administratif afin que le lecteur ne soit pas perdu :

- **2001 :** adoption de la directive 2001/29/CE dite EUCD (*European Union Copyright Directive*) au Parlement Européen, qui servira de base à la loi DADVSI et à la loi Hadopi 1 et 2 ;
- **1er Août 2006 :** après des débats houleux, promulgation de la DADVSI qui pose comme principe que toute personne qui téléchargerait de façon illégale des contenus culturels numériques serait passible d'une peine maximum de trois ans d'emprisonnement et de 300 000 euros d'amende. La réponse graduée, qui était déjà évoquée à l'époque, est évacuée.
- **23 Novembre 2007 :** publication du rapport Olivennes et conclusion d'un accord entre l'Élysée, les ayants droit et les fournisseurs d'accès à Internet pour mettre en place de façon effective un système de réponse graduée pour lutter contre le téléchargement illégal ;
- **16 juin 2008 :** présentation du texte Hadopi 1 par Christine Albanel en Conseil des Ministres ;
- **Septembre 2008 :** adoption de la résolution 138 au Parlement Européen, disposant qu' « aucune restriction aux droits et libertés fondamentales des utilisateurs finaux ne doit être prise, sans décision préalable de l'autorité judiciaire ».
- **Mai 2009 :** après des débats très houleux, première censure du Conseil Constitutionnel du texte Hadopi 1 ;
- **Septembre 2009 :** adoption du texte Hadopi 2 ;
- **début 2010 :** création de l'institution Hadopi ;
- **Février 2011 :** création des Labs de l'Hadopi ;
- **Juin 2012 :** création du DREV ;

- **Décembre 2012 :** fermeture officielle des Labs ;
- **printemps 2015 :** restructuration de l'institution ;
- **été 2015 :** licenciement du secrétaire général Éric Walter ;
- **Décembre 2015 :** fin de la présidence de Marie-Françoise Marais et de Mireille Imbert-Quaretta.

Pour que le lecteur comprenne parfaitement les lignes qui vont suivre, nous devons aussi revenir très rapidement sur les trois missions imposées à l'institution par le législateur.

1 Que fait donc l'Hadopi ?

Le législateur a posé trois missions :

- mise en place et application de la réponse/riposte graduée ;
- développement de l'offre légale ;
- labellisation des moyens de sécurisation de l'accès à Internet.

Le volet répressif – qui est également le plus connu – de l'Hadopi est constitué de la réponse graduée et sanctionne – en théorie – non pas le téléchargement illégal, mais le défaut de sécurisation de l'accès à internet d'un abonné.

La réponse graduée consiste à envoyer un premier courriel d'avertissement à un internaute repéré pour avoir fait du téléchargement illégal. S'il récidive, il recevra un nouvel avertissement par courrier recommandé. Enfin, en cas de troisième récidive, en fonction de la délibération de l'Hadopi, son dossier sera ou non transmis au parquet le plus proche du domicile de cet internaute, qui encourt alors une amende de 1 500 euros et la suspension de l'accès à internet pour un mois au maximum. En théorie... car dans la pratique, c'est un peu plus subtil.

Premier problème : l'internaute n'est pas sanctionné pour avoir téléchargé mais parce qu'il n'aurait pas sécurisé son accès à Internet. Transposé dans le monde réel, cela reviendrait à sanctionner la personne qui se fait cambrioler parce qu'elle n'aurait pas investi dans des moyens de sécurité performants.

Deuxième problème : la collecte des informations n'est pas opérée par l'institution elle-même mais par une entreprise privée : TMG. Cette dernière scrute les réseaux peer-to-peer, afin de collecter les adresses IP des personnes qui téléchargeraient des contenus soumis au droit d'auteur figurant sur le catalogue des œuvres des ayants droit. Une fois que TMG a envoyé à l'Hadopi la liste des adresses IP, l'institution prend contact avec les FAI pour obtenir les coordonnées des abonnés détenteurs de ces adresses IP. Les fournisseurs d'accès à Internet (FAI) fournissent les coordonnées de leurs abonnés à l'Hadopi qui peut choisir d'envoyer une première recommandation par courriel.

Troisième problème : l'institution dépense 6 millions d'Euros pour la réponse graduée et y consacre 24 agents. Son budget pour 2016 sera de 8,5 millions d'Euros et il reste une cinquantaine d'agents. Je laisse au lecteur le soin de faire les calculs de répartition.

Quatrième problème : une loi est comme une paire de jambes. Pour être équilibrée, elle doit prévoir un volet répressif et un volet préventif et/ou pédagogiquement dissuasif. Si le volet répressif est constitué avec la réponse graduée, le volet pédagogique est minoré.

Le calcul est simple : si l'on souhaite que les internautes arrêtent de télécharger des œuvres culturelles numériques mises en ligne de façon illicite, il faut leur proposer une alternative séduisante. Dès 2011, la direction de la communication avait planché sur le label PUR (Promotion des Usages Responsables). Au-delà de la maladresse du nom, la communication, créée par l'agence H, était assez désastreuse et a été largement tournée en ridicule.

Sur le plan purement légal (sans mauvais jeu de mots), l'institution n'avait et n'a toujours que très peu de moyens. En effet, il ne fallait pas fausser la libre concurrence, principe constitutionnel – donc il était et est toujours impossible de proposer des conseils techniques, marketing et/ou juridiques pour que les plateformes proposant des contenus culturels numériques de façon licite puissent être mises en avant. En tant que tel, le label PUR qui deviendra le LOL (Label Offre Légal) à la fin de l'année 2013, n'apportait aucune valeur ajoutée aux plateformes.

Même si LOL – au moins sur le plan esthétique – est mieux construit que son ancêtre PUR, il souffre des mêmes lacunes que son prédécesseur. En effet, lors de sa sortie officielle, il référençait 335 sites mais seuls 62 étaient labellisés LOL, les autres étant regardés comme légaux **[1]**. L'institution posait donc que l'internaute pouvait avoir comme postulat que le site semblait en conformité avec la loi sauf qu'une apparence n'est pas une certitude.

Contrairement à l'ARJEL (Autorité de Régulation des Jeux En Ligne), qui impose un label sur les sites de jeux d'argent en ligne, l'absence de label PUR sur un site d'offres culturelles numériques ne le place pas dans l'illégalité. Ce n'est pas parce qu'une plateforme qui s'appellerait jaimelecinema.video (nom de domaine inventé) n'a pas le label PUR qu'elle propose nécessairement des contenus culturels numériques

de façon illicite, alors que si le site joueraupoker.game (nom de domaine inventé) n'est pas labellisé par l'ARJEL, alors il est considéré comme illégal.

Ainsi, du point de vue d'une plateforme de contenus culturels numériques, s'il n'y a ni expertise ni plus-value légale par rapport à des concurrents « illicites », il n'y a pas d'intérêt à être labellisé par l'institution, d'autant que les apports en termes de trafic ne sont pas formidables puisqu'il n'est pas permis aux institutions d'optimiser leurs référencements.

Dernier volet de la loi : la labellisation des moyens de sécurisation de l'accès à Internet. Il ne revenait pas à l'institution de créer elle-même des moyens de sécurisation mais de proposer aux solutions d'être reconnues comme sûres. Que s'est-il passé dans l'esprit du législateur pour arriver à croire qu'il était possible de sécuriser totalement une connexion à Internet ? Le fait que le seul moyen de ne pas se retrouver avec une utilisation malveillante et non consentie de son accès à Internet est de ne pas avoir de connexion Internet.

Cela n'a pas empêché l'institution d'essayer d'y travailler très temporairement.

Il existe une autre mission qui a été déléguée à l'Hadopi : la régulation et l'étude des mesures techniques de protection, que le lecteur connaît certainement sous la dénomination DRM (Digital Right Management). Quand l'Hadopi – institution – a été créée, elle a été fusionnée avec l'ART (Autorité de Régulations des Télécommunications), qui avait notamment pour mission de travailler sur les DRM/MTP.

Au sein de l'Hadopi, il y a toujours eu un débat épistémologique et sémantique sur les termes à utiliser : DRM ou MTP ? En effet, les deux notions ne recouvrent pas la même réalité. Le DRM suppose que vous ayez des droits spécifiques quand vous achetez un support culturel alors que le MTP ne repose que sur le postulat « droit du consommateur ».

2 Que faire avec l'Hadopi ?

En effet, le mandat de la présidente de l'institution, Marie-Françoise Marais, se termine le 23 décembre 2015. Mais quel sera l'avenir de cette institution ?

Que le lecteur soit immédiatement rassuré : l'Hadopi ne sera pas supprimée. Au-delà du fait que cela ne figure absolument pas dans le calendrier législatif déjà fort chargé, il faut savoir que sous la Ve République, on ne supprime pas des autorités administratives indépendantes. On les fusionne éventuellement avec d'autres – cas de l'ART – on en modifie les missions et le spectre de compétence, mais les supprimer purement et simplement, on ne sait pas.

Le budget de l'institution pour l'année 2016 sera de 8,5 millions d'Euros. Or, si la réponse graduée telle qu'elle est configurée actuellement se poursuit, il ne restera que 2 millions pour assurer la promotion de l'offre légale.

Quant au département d'études, à savoir le DDRD, il n'a aucune garantie d'existence après le départ de Marie-Françoise Marais car sa mission n'est pas juridiquement encadrée. Il n'y a rien dans les textes qui obligent l'autorité à garder un département d'études.

Du côté de l'offre légale, en dehors de l'entretien cosmétique et technique de la plateforme Offre Légale, rien n'est acté et aucun moyen supplémentaire ne pourra être alloué. Or, l'arrivée assez fracassante de Netflix sur le marché français a définitivement ringardisé la plateforme de l'Hadopi. Ainsi, j'ai eu la surprise d'entendre des amis me dire qu'ils avaient arrêté de télécharger des contenus culturels numériques sur des plateformes non respectueuses du droit d'auteur, non pas par peur de l'Hadopi – vu que l'institution ne délègue sa surveillance que sur les réseaux peer-to-peer, ni par peur des ayants droit mais parce que le service en question leur offrait un service innovant, rapide et intuitif.

Quid des auteurs justement ? Imaginons que vous soyez un auteur, que vous ayez écrit un roman et que vous constatiez que votre œuvre se retrouve sur des plateformes de téléchargement sans votre accord. Vous ne pouvez pas saisir directement l'Hadopi. Il faudra passer par un processus long et complexe, qui consiste à le signaler à votre maison d'édition, qui va ensuite se retourner vers une société de gestion de droits d'auteur, qui va elle-même se tourner vers TMG pour lui indiquer l'œuvre à surveiller sur les réseaux peer-to-peer et qui transmettra éventuellement des adresses IP collectées sur des réseaux peer-to-peer à l'Hadopi, qui se retournera vers le FAI pour avoir les identités, pour ensuite envoyer aux titulaires des abonnements des emails d'avertissements.

Du point de vue des auteurs de contenus culturels, l'Hadopi est une réponse inadaptée et inefficace et beaucoup trop d'espoirs ont été placés dans une institution au budget ridicule (par rapport à un budget d'Etat), avec des moyens légaux fantaisistes et malheureusement, des agents ayant plus à cœur la promotion de leurs propres intérêts que celle de la chose publique.

Alors, l'Hadopi : zombie ou pas ? Il est certain qu'il ne se passera absolument rien de notable d'ici la fin de l'année 2015 et il faudra certainement quelques mois à la nouvelle présidence de l'institution pour énoncer des lignes directrices claires, sans parler d'un éventuel ménage dans les effectifs.

Du côté du législateur, malgré les différents rapports d'enquêtes, rien ne semble bouger alors qu'il semble urgent de plancher sur une loi Hadopi 3. Cette loi Hadopi devrait – c'est du moins mon souhait personnel – traiter :

- de la rémunération de la copie privée ;
- des relations entre les artistes et les sociétés de gestion de droits afin que certains artistes puissent enfin percevoir les fruits de leurs œuvres ;
- des flux financiers des plateformes mettant à disposition des œuvres culturelles numériques de façon illicite ;
- de la possibilité pour les auteurs de saisir directement l'institution ;
- de la possibilité pour les consommateurs de saisir directement l'institution quand leur exception à la copie privée n'est pas respectée par les éditeurs.

A ce jour, aucun des points listés précédemment n'est traité ou mis en place ou juridiquement faisable par l'institution. Or, ce sont les thématiques qui sont le plus souvent revenues dans les discussions quand le forum des Labs était en activité et cela reste des points litigieux à la fois pour les consommateurs et pour les artistes.

Sur le plan politique, une telle mise à jour de la loi Hadopi et donc de l'institution ne peut pas se faire en fin de mandat présidentiel car dans la mesure où cela touche à l'exception culturelle, c'est politiquement dangereux. En politique, il existe un calendrier implicite des réformes, de celles dont on dit que ce sont des réformes de début de mandat présidentiel car la bataille qui s'en suivra sera tellement longue qu'elle occupera tout l'espace médiatique.

Or, sur le plan culturel, le président François Hollande a totalement délaissé la question de l'exception culturelle et l'ancienne ministre de la Culture n'avait pas les faveurs des ayants droit. Quant à Fleur Pellerin, elle ne fait pas figure d'innovatrice et à un an de la présidentielle, il est évident que le chantier culturel n'est pas une priorité.

Conclusion

La seule éventuelle révolution qui pourrait intervenir concernant l'Hadopi se produira durant la prochaine législature et seulement si la personne qui sera détentrice du portefeuille culture est soit acceptée de façon quasi unanime par les ayants droit, soit issue de la culture.

Enfin, rien ne pourra être réellement innovant si nous ne procédons pas à une remise en question en profondeur de l'exception culturelle. Soit on part du principe que la culture est quelque chose qui sort du champ de la consommation – au sens large du terme – et on lui applique donc des principes dérogatoires du droit commun, y compris en matière de promotion, soit on considère que la culture est un bien marchand comme les autres et on laisse les artistes et leurs intermédiaires se débrouiller entre eux. Aujourd'hui, nous avons un système hybride qui consiste à réguler certaines actions mais qui laissent les acteurs privés imposer leurs règles, sans pour autant les canaliser, système qui revient à pénaliser les consommateurs et les artistes. ■

Référence

[1] Hadopi - j'ai testé leur nouveau service inutile de téléchargement légal : http://rue89.nouvelobs.com/2013/12/13/hadopi-jai-teste-nouveau-service-inutile-telechargement-legal-248325

PRATIQUE DE LA NORME MISRA C

Stéphane LONKENG TOULEPI [Ingénieur en Télécommunication et Entrepreneur en Électronique Médicale]

La plupart d'entre nous (moi le premier) ont écrit leur premier programme en vadrouille ; l'essentiel était de faire que le programme fonctionne. Avec le temps on a appris à écrire des commentaires même si on savait depuis le début que ça existe. Nous allons voir dans les lignes qui suivent qu'il existe plusieurs règles de codage qui permettent de rendre le code plus compréhensible, organisé et sûr; la norme MISRA C s'inscrit dans cette optique et est spécialement dédiée aux systèmes critiques.

Mots-clés : MISRA C, Sûreté de fonctionnement, Programmation C, Règles de codage, Conformité, Test

Résumé

Après Java, le C est le deuxième langage de programmation compilé le plus utilisé au monde [1]. Son succès tient du fait qu'historiquement il fut l'un des premiers langages de haut niveau à la portée du programmeur moyen de l'époque. Par ailleurs, son rapprochement du processeur attire les concepteurs de systèmes critiques. C'est dans l'optique de proposer un ensemble de règles pour les systèmes critiques que le groupe de travail MISRA développa la norme MISRA C. L'application de cet ensemble de règles peut se faire suivant plusieurs méthodologies, chacune adaptée aux domaines comme le ferroviaire, le médical, l'aéronautique, le spatial ou le militaire. Pour couronner le tout, il existe des logiciels permettant de tester et d'effectuer automatiquement une vérification de conformité relativement à MISRA C.

La première chose qu'il faut savoir lorsqu'on démarre un projet ayant des problématiques de sûreté de fonctionnement c'est qu'on sera soumis à des règles strictes lors de la programmation. À tel point qu'il devient parfois plus difficile de respecter les règles que de se concentrer sur les fonctionnalités du code même. La norme MISRA C rassemble des règles bien expliquées et facilement compréhensibles pour les développeurs habitués. La conformité d'un code par rapport à MISRA C est considérée à ce jour comme une caractéristique de sûreté de fonctionnement non négligeable. Nous allons présenter dans ce qui suit l'historique de cette fameuse norme dont vous avez sûrement déjà entendu parler ; nous décrirons également ses objectifs et une méthodologie de mise en pratique.

1 Petit historique

1.1 Naissance du C

Dans les années 60, les programmeurs utilisaient généralement le langage assembleur, très proche de la machine.

Seulement, pour réaliser une tâche en langage assembleur, il fallait écrire plusieurs pages de code. C'est pour corriger cette difficulté que **Ken Thompson** et **Dennis M. Ritchie** ont mis sur pied le *langage B* **[2]** ; par la suite Dennis inventa le *langage C* **[3]**. Ce dernier possédait toutes les caractéristiques nécessaires pour la programmation d'un système d'exploitation ; c'est ainsi que plusieurs composants du système UNIX ont été reprogrammés en C en 1973.

Note

Le langage B a été inventé en 1969 et est aujourd'hui devenu obsolète. Il fut inspiré du BCPL et est considéré comme le langage qui fut la transition du BCPL au C. Ce langage ne disposait que d'un seul type de donnée : le mot, qui pouvait représenter un entier ou une adresse mémoire. Par ailleurs, le B disposait déjà de certaines des fonctions qui par la suite seront utilisées dans la bibliothèque C standard.

1.2 De la popularisation du C…

À partir de cette date, le langage C a été de plus en plus utilisé par les développeurs ; c'est ainsi qu'un comité a été formé en 1983 par **l'ANSI** (*American National Standards Institute*) pour développer une définition moderne et harmonisée du langage C. Par la suite, en 1988, le comité a délivré la version finale du standard **ANSI C**. Le langage C étant très proche du matériel et largement répandu, il a su s'imposer au niveau des systèmes embarqués, dans l'aéronautique, l'automobile et même les dispositifs médicaux. Si vous suivez bien mon regard, vous devez percevoir la problématique de sûreté de fonctionnement des programmes développés en C. En effet, il n'y a aucun langage de programmation qui soit capable de garantir que le programme exécutable se comporte comme le programmeur le souhaite. Plusieurs paramètres sont à l'origine de ce défaut. On peut citer entre autres les erreurs de frappe, les incompréhensions du langage, les erreurs de programmation dudit compilateur ou encore les erreurs durant l'exécution du programme.

Attention !

Il faut souligner ici que l'ANSI C n'est qu'un standard qui permet d'harmoniser les caractéristiques du C. En effet, les compilateurs tels que **gcc** sont développés à partir de ce standard ; si celui-ci n'existait pas, ce serait très difficile de produire un code source qui pourrait être compilé par tous les compilateurs. Notez bien quand même que le fait qu'un code respecte le standard C ne signifie pas qu'il soit adapté à un système critique.

1.3 … à la création de la norme MISRA C

Pendant que le langage C faisait son petit bout de chemin, un projet gouvernemental au Royaume-Uni a vu le jour ; il s'agit du *« SafeIT Program »*. Ce programme incluait le projet **MISRA** (*Motor Industry Software Reliability Association*) **[4]** avec pour objectif de développer des lignes directives pour la création des systèmes embarqués dans les véhicules routiers. En novembre 1994, MISRA a atteint son objectif avec la parution d'un document intitulé *« Development guidelines for vehicle based software »*.

Fig. 1 : Une photo des membres du MISRA C Working Group au lancement officiel de MISRA C:2004 à l'Embedded system Show, le 13 octobre 2004.

Une fois l'objectif atteint, les membres de MISRA ont décidé de travailler sur une base informelle pour développer cette fois des lignes directives pour l'utili-

sation du langage C dans des systèmes critiques divers. Ils ont été rejoints dans cette lancée par FORD et ROVER. C'est ainsi qu'en 1998, une première version de MISRA C puis une version améliorée en 2004 virent le jour (la photo de la figure 1 montre les membres du MISRA C Working Group en 2004).

Avec le temps, MISRA C **[5]** est devenu un standard du langage C reconnu pour l'industrie des systèmes embarqués critiques.

2 Objectifs

L'objectif du MISRA C est de proposer un ensemble de règles qui permettront aux développeurs de mettre sur pied des programmes C adaptés aux systèmes critiques tels qu'on en retrouve dans l'industrie nucléaire, aéronautique, automobile, ferroviaire, spatiale ou médicale. C'est ainsi qu'il met un accent sur les règles de codage, ce qui permet entre autres aux développeurs de s'habituer à ces règles.

Attention !

Il faut noter que MISRA C ne concerne pas le langage C++. En effet, il y a certaines déclarations qui ont différentes interprétations en C et C++. En plus, un code C peut contenir un identifiant qui peut être interprété en C++ comme un mot réservé. Pour ces deux raisons, un compilateur C++ qui compile un code écrit en C et conforme au standard **ISO C** ne produira pas forcément le même résultat avec un compilateur C. Par ailleurs, pour les codeurs C++, il existe un standard MISRA C++ publié en 2008 par MISRA avec les mêmes objectifs pour le langage C++ **[6][7]**.

3 Aperçu des règles MISRA C

Le groupe de travail MISRA a défini un ensemble de règles que doit respecter un programme C pour être conforme MISRA C. Ces règles sont classifiées en deux groupes :

- Le premier groupe concerne les *« Required rules »*. L'ensemble des règles est consigné dans le document de référence de MISRA C **[8]**. Au nombre de 122, ces règles sont obligatoires pour que le programme soit conforme MISRA C.
- Le second groupe concerne les *« Advisory rules »*. Au nombre de 20, ces règles ne sont pas obligatoires pour la conformité mais sont à suivre autant que possible par les programmeurs.

Les règles sont donc présentées comme suit avec leurs significations :

```
Rule <number> (<cathegory>): <requirement text>
                [<source reference>]
<normative text>
-------------------------------------------------------------------------------
----------
<number> : Chaque règle possède un numéro unique qui permet de l'identifier;
<category>: La catégorie peut être "Required" ou "Advisory" comme vu précédemment
<requirement text>: Enoncé de la règle en question
<source reference>: Origines de la règle (Annexe G, ISO/IEC,...). Lorsque ce champ n'est
pas spécifié, cela signifie que la règle a été initiée par une entreprise privée, par un
contributeur privé ou par une bonne pratique reconnue de tous.
<normative text>: Texte décrivant la règle avec des exemples d'applications
éventuellement.
```

Les règles sont subdivisées en suivant plusieurs thèmes dans le document de référence MISRA C 2004 ; par exemple, on a les règles liées à l'environnement, à la documentation du projet, aux types de données, aux conversions de pointeurs, aux conversions arithmétiques, etc. Ceci permet au programmeur de facilement se repérer.

Exemple

Nous allons prendre deux exemples ; le premier concerne la section *« Language extensions »* c'est-à-dire tout ce qui est commentaires et encapsulations de code d'un langage différent (l'Assembleur).

```
Rule 2.4 (advisory):Sections of code should not be "commented out".

Where it is required … change the effect.
```

Il s'agit ici d'une règle *advisory* qui conseille simplement aux programmeurs d'éviter de commenter les codes sources à supprimer comme c'est couramment le cas lorsqu'on veut compiler le code plus d'une fois en fonction d'un paramètre par exemple. À la place, MISRA nous conseille plutôt d'utiliser les directives préprocesseur (*#if … #ifdef*) construites avec des commentaires.

Le deuxième exemple que nous allons prendre est contenu dans la section *« Initialisation »* qui concerne toutes les problématiques liées aux déclarations/initialisations des variables.

```
Rule 9.1 (required): All automatic
variables shall have been assigned a value
before being used.
[Undefined 41]
The intent of this … usually automatically
initialised.
```

Il s'agit ici d'une règle *required* qui oblige le programmeur à toujours initialiser ses variables. En effet, lorsque l'on n'initialise pas les variables, elles ont pour la plupart du temps des valeurs non souhaitées et il y a des risques de les utiliser par erreur. Comme on peut le voir, cette règle est présentée avec le champ **`<source reference> [Undefined 41]`** ; cela vient du fait qu'il s'agit d'une bonne pratique reconnue de tous.

4 Mise en pratique

4.1 Utilisation

Selon MISRA, le niveau de sûreté de fonctionnement d'un programme dépend également du contexte de programmation ; en effet, le groupe de travail suggère six autres bonnes pratiques à mettre en œuvre durant le projet de programmation.

4.1.1 Formation des programmeurs

Avant le démarrage du projet, il est important de former les développeurs sur la programmation en C relativement au domaine du projet (C pour l'embarqué, C pour les systèmes critiques, C pour …). En effet, si votre équipe intègre un très bon programmeur C qui a l'habitude de programmer dans le ferroviaire, il est possible qu'il se plante sur certaines spécificités de programmation propres au médical par exemple. Il est également important de former les développeurs sur les outils qui seront utilisés dans la phase de test. Les outils utilisés dans cette phase permettront de vérifier la conformité des programmes relativement à MISRA C et/ou mesurer la complexité du code. J'ajouterai enfin la formation des programmeurs à MISRA C en elle-même (leur demander de lire cet article par exemple :-)).

4.1.2 Harmonisation du style dans la programmation

Il s'agit simplement ici de faire en sorte que les habitudes de programmation soient harmonisées ; par exemple, comment tous les programmeurs doivent indenter le code, comment rédiger les commentaires, les conventions de nomenclature des variables, inclusion du copyright de la société et de la description générale du contenu d'un fichier dans l'entête, etc. Pour faciliter l'harmonisation du style, il existe des conventions reconnues comme la notation hongroise **[9]** ou le *« GNU Coding Standards »* de GNU ; vous avez le choix en fonction des orientations des développeurs et des caractéristiques du projet.

4.1.3 Choix des outils de développement

Le choix des outils de développement est important pour le bon déroulement d'un projet dans la mesure où il existe aujourd'hui plusieurs solutions ayant chacune leurs particularités qui sont plus ou moins adaptées à chaque domaine. Certains logiciels de développement sont spécialisés dans l'embarqué et possèdent des compilateurs pour plusieurs cibles et vont jusqu'à faire la vérification de conformité MISRA C. Il est évident que ceux-ci coûtent assez cher tandis qu'il y a d'autres logiciels qui n'offrent que la vérification de conformité mais pour lesquels il faut trouver soit même un compilateur avec l'environnement de développement qui va avec. Plusieurs paramètres rentrent donc en jeu parmi lesquels on peut citer le coût, l'efficacité et la difficulté d'utilisation.

4.1.4 Mesure de la complexité du code C

La norme MISRA C recommande de déterminer le niveau de complexité du code source. Ici, les informations qui sont principalement utiles pour déterminer le niveau de complexité sont :

- les zones suspectes en terme de sécurité (*Buffer overflow*, division par zéro, ...) ;
- effort à fournir par un programmeur lambda pour comprendre le code ;
- effort à fournir pour tester le code ;
- niveau d'explication du code par les commentaires.

Il existe par ailleurs des outils tels que **Pmccabe**, **GNU Complexity** et **Frama-C** permettant de déterminer le niveau de complexité du code C relativement aux quatre caractéristiques précédentes. L'un des résultats importants fournis par les outils de mesure de la complexité est le *nombre cyclomatique* **[10-11]**. Plus ce nombre est faible, plus le programme est facile à lire, à tester et entretenir.

Note

Nombre ou compléxité cyclomatique

Le nombre cyclomatique est un outil créé par **Thomas McCabe** en 1976 pour mesurer la complexité d'un programme informatique. Ce nombre représente le nombre de chemins que le programme peut emprunter. Plus simplement, il s'agit du nombre de points de décision sous une condition (`if`, `while`, `case`, …) `+ 1` (le chemin principal). Par ailleurs, le programme peut être modélisé comme étant un graphe dont les nœuds (`N`) sont les conditions et les arêtes (`A`) sont les parcours entre deux nœuds. Ceci permet, grâce à la théorie des graphes, d'obtenir la formule : `Complexité = A - N + 2`.

4.1.5 Test du code

La phase de test du code suit directement le développement ; MISRA C recommande de concevoir le programme de manière à ce qu'on puisse facilement le tester par la suite. Cette pratique s'appelle par ailleurs *« Design For Test »*. Globalement la démarche de test du code doit être planifiée depuis le début du projet. Il faut également noter que c'est cette phase qui intègre la vérification de conformité du code relativement aux règles MISRA C.

4.2 Logiciels

Comme nous l'avons dit précédemment, il existe des logiciels qui permettent d'analyser automatiquement les codes conformément à la norme MISRA C. Ces analyseurs peuvent vérifier les règles de façon statique, en scrutant juste le code source ou de façon dynamique, en l'exécutant. En réalité l'utilisation de ces logiciels n'est pas aussi facile qu'on pourrait l'imaginer. En fait, lorsque le code est inséré dans le logiciel, il détecte un certain nombre de non-conformités ; la plupart des logiciels professionnels détectent et proposent (quand ils le peuvent) des corrections mais dans le cas contraire, c'est aux programmeurs de rentrer dans la documentation MISRA C pour trouver la solution appropriée.

Malheureusement, je n'ai pas pu trouver de logiciels libres dignes de ce nom :-(; j'en profite d'ailleurs pour faire un clin d'œil à la communauté opensource vu que le besoin se fait fortement ressentir. Parmi les logiciels propriétaires, on peut citer entre autres **Astrée**, **Coverity**, **ECLAIR**, **GrammaTech**, **Klockwork**, **LDRA** et **PCLint**.

Conclusion

En pratique, le C est le langage le plus utilisé dans les domaines industriels critiques. La criticité des logiciels a entraîné la mise en œuvre de la norme MISRA C, permettant d'éviter au maximum les erreurs lors de l'exécution de programmes. Cette norme est constituée d'un ensemble de règles à respecter par un programme pour être conforme. Il existe fort heureusement des logiciels permettant d'automatiser les vérifications de conformité bien qu'ils soient à notre connaissance tous propriétaires. Par ailleurs, il faut noter que la conformité doit être associée à une méthodologie globale de gestion de la sûreté de fonctionnement du logiciel. ■

Références

[1] CASS S., « Top 10 Programming Languages », 19 juillet 2014 : http://www.open-source-guide.com/Actualites/Les-langages-de-programmation-les-plus-populaires

[2] Le langage B : https://fr.wikipedia.org/wiki/B_(langage)

[3] L'histoire du langage C : http://www.codingunit.com/the-history-of-the-c-language

[4] Site officiel de MISRA : http://www.misra.org.uk/

[5] Site officiel de MISRA C: http://www.misra-c.com/

[6] MOTTOK J. et SCHILLER F., « MISRA C++:2008 & Safely Embedded Software (SES) », novembre 2008

[7] Site officiel de MISRA C ++ : http://www.misra-cpp.com/

[8] BURNARD A., BURDEN P., WHITING L., TAPP C., McGALL G., HENNELL M.Mike, HILLS C. et MONTGOMERY S., « MISRA C 2004 Guidelines for the use of the C language in critical systems », 2004

[9] Notations hongroises : https://fr.wikipedia.org/wiki/Notation_hongroise

[10] Nombre Cyclomatique : https://fr.wikipedia.org/wiki/Nombre_cyclomatique

[11] Mesure de la Qualité du code source : http://www-igm.univ-mlv.fr/~dr/XPOSE2008/Mesure%20de%20la%20qualite%20du%20code%20source%20-%20Algorithmes%20et%20outils/complexite-cyclomatique.html

DOMAINES | MAIL | HÉBERGEMENT | E-COMMERCE | **SERVEURS**

TESTEZ LE MEILLEUR CLOUD

TESTÉ ET APPROUVÉ PAR CLOUD SPECTATOR

Powered by intel Cloud Technology

1&1 Serveur Cloud : Easy to use – ready to Cloud*

Les performances des nouveaux serveurs Cloud sont imbattables en termes de CPU, RAM et SSD.

Réalisez vos projets dans le Cloud avec la combinaison parfaite entre flexibilité et performances.

- ✓ **Load balancing**
- ✓ **Stockage SSD**
- ✓ **Facturation à la minute**
- ✓ **Intel® Xeon® Processor E5-2660 v2 et E5-2683 v3**

1 mois gratuit !

Puis à partir de 4,99 € HT/mois (5,99 € TTC)*

1 MOIS POUR ESSAYER ★★★★★

1 CLIC POUR CHANGER DE PACK ★★★★★

1 APPEL UN EXPERT VOUS RÉPOND ★★★★★

0970 808 911
(appel non surtaxé)

1&1

1and1.fr

*Facile à utiliser - prêt pour le Cloud. 1&1 Serveur Cloud : 1 mois d'essai gratuit, puis à partir de 4,99 € HT/mois (pour la configuration du serveur Cloud S). Facturation mensuelle en fonction de la configuration choisie. Pas de durée minimum d'engagement, ni de frais de mise en service. Conditions détaillées sur 1and1.fr. Intel et le logo Intel sont des marques commerciales d'Intel Corporation aux États-Unis et/ou dans d'autres pays. 1&1 Internet SARL, RCS Sarreguemines B 431 303 775.

PARCOURIR DES GRAPHES EN LARGEUR

Nicolas PATROIS [Professeur de mathématiques dans l'enseignement secondaire]

Le Docte Roux a besoin de se promener dans le réseau du temps dans sa cabane en bois plus petite dedans que dehors, le TORDUS. Pour anticiper, il a besoin de résoudre, à l'aide du parcours en largeur, quelques problèmes de graphes classiques ou amusants.

Mots-clés : Graphe, Parcours en largeur, BFS, Dijkstra, Python, Algorithme

Résumé

Le parcours en largeur est une technique de parcours de graphe simple à mettre en œuvre mais néanmoins puissante. Cet article propose quelques exemples de problèmes résolus avec le parcours en largeur en Python, dont une variante qu'est l'algorithme de Dijkstra (un algorithme de recherche de plus court chemin).

Il ne s'agit ni de courir en crabe ni de descendre de vélo pour se regarder pédaler comme en pédagogie moderne mais d'utiliser, dans des graphes, le parcours en largeur (*breadth first search* en anglais ou *BFS*) et quelques variantes proches. Le principe est simple, nous le donnerons dans la première partie. La deuxième en fera tourner quelques applications plus ou moins immédiates, la dernière s'intéressera à une variante : l'algorithme de Dijkstra. Tous les codes sont en Python (2 ou 3, aucune importance ici).

1 Le parcours en largeur

Un graphe est, mathématiquement, un ensemble de points, les sommets, reliés par des liens ou arêtes, éventuellement orientés (on parle alors de digraphe et on représente les arêtes par des flèches). Il suffit de regarder un plan des lignes de métro ou de bus d'une ville ou de penser à internet pour comprendre.

1.1 Le principe

Il s'agit de parcourir un graphe de proche en proche, comme une tache d'huile. On commence au sommet initial, puis on va sur les sommets qui lui sont directement reliés. À l'étape suivante, on franchit les sommets non déjà franchis qui sont directement reliés à ceux de l'étape précédente, et ainsi de suite jusqu'à épuisement des sommets disponibles.

Voici ce qui se passe sur un graphe non orienté (je grise les sommets visités petit à petit) (voir tableau suivant) :

Notez que si le graphe est non connexe (en plusieurs morceaux), on ne peut pas franchir tous ses sommets et de même s'il est connexe et orienté, on peut ne pas les rencontrer tous (penser à un arbre orienté ou au graphe du temps de la section 2.2. si on commence au sommet **C**). Les graphes seront dans la suite connexes.

1.2 Modéliser un graphe

Les graphes peuvent, en algorithmique, être modélisés de deux façons principales :

1. une matrice d'adjacence qui est un tableau carré de nombres,
2. une liste d'adjacence où chaque sommet pointe vers les informations qui le concernent.

Étape 1	Étape 2

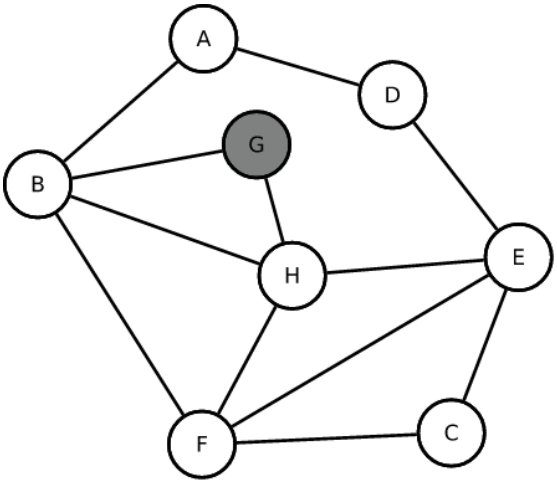
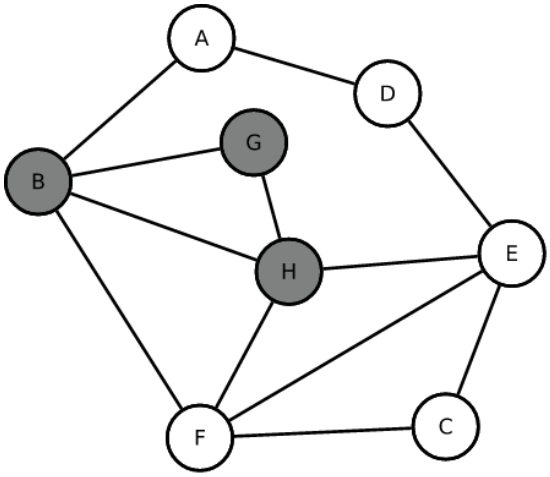

	

Fig. 1 : On part du sommet G, qu'on grise.	*Fig. 2 : Du sommet G, seuls B et H sont reliés, on les grise.*
Étape 3	**Étape 4**

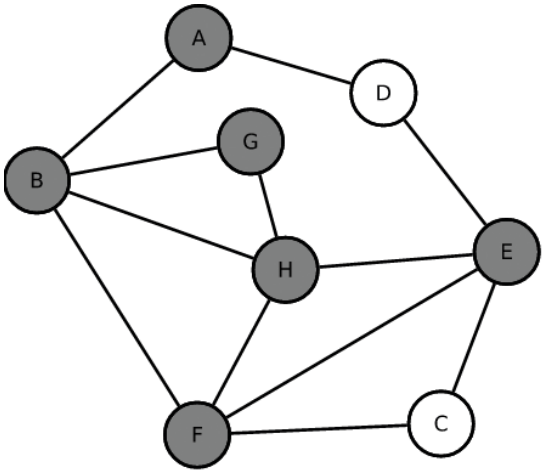
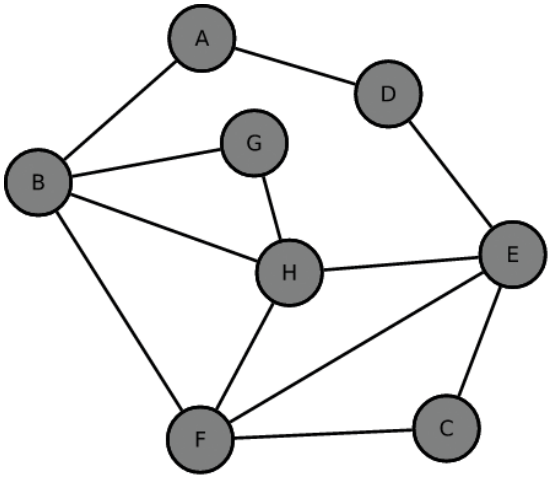
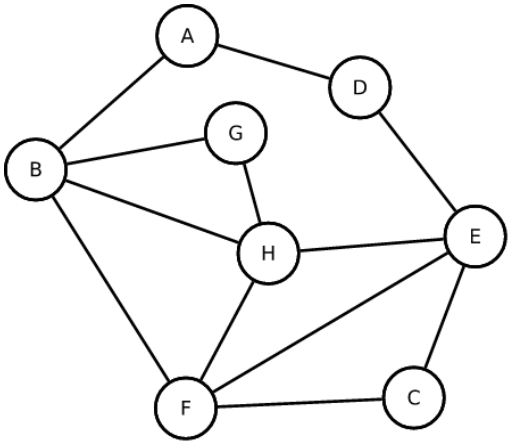

	

Fig. 3 : De B et H, les sommets non déjà parcourus sont A, E et F qu'on grise.	*Fig. 4 : Et de ces trois-là, on peut rejoindre C et D et il ne reste aucun sommet libre.*

On peut, avec une matrice d'adjacence, coder soit la présence d'un lien entre deux sommets avec un **1** (et une absence par **0** ou leur nombre s'il y en a plusieurs) soit coder leur distance (l'absence de lien peut se coder par un symbole spécial, par **0** ou par un nombre assez grand qui assure que le lien ne sera jamais franchi). Dans les deux cas, le graphe peut être orienté, c'est le cas par exemple d'un réseau routier avec des sens interdits ou les durées de marche en montagne, la matrice ne sera pas symétrique.

Pour une liste d'adjacence, on peut utiliser un dictionnaire où la valeur de chaque sommet-clé contient la liste des sommets qui lui sont reliés ou alors une liste de listes plus compactes. Voici un exemple (voir le tableau en page suivante).

Graphe	Matrice d'adjacence	Liste d'adjacence

 *Fig. 5 : Le graphe modélisé ci-contre*	`M1=[[0,1,0,1,0,0,0,0],`
	`[1,0,0,0,0,1,1,1],`
	`[0,0,0,0,1,1,0,0],`
	`[1,0,0,0,1,0,0,0],`
	`[0,0,1,1,0,1,0,1],`
	`[0,1,1,0,1,0,0,1],`
	`[0,1,0,0,0,0,0,1],`
	`[0,1,0,0,1,1,1,0]]`
		Avec un dictionnaire :
		`M2={"A":{"B","D"},"B":{"A","F","G","H"},"C":{"E","F"},"D":{"A","E"},"E":{"C","D","F","H"},"F":{"B","C","E","H"},"G":{"B","H"},"H":{"B","E","F","G"}}`
		ou de manière intermédiaire :
		`M3=[[1,3],[0,5,6,7],[4,5],[0,4],[2,3,5,7],[1,2,4,7],[1,7],[1,4,5,6]]`
		`M4=[{1,3},{0,5,6,7},{4,5},{0,4},{2,3,5,7},{1,2,4,7},{1,7},{1,4,5,6}]`

`M1[2][0]` et `M1[0][2]` valent `0` donc les sommets `A` et `C` ne sont pas reliés, ce qu'on peut lire par `M2["C"]` ne contient pas `A` pas plus que `M2["A"]` ne contient `C` ou de même que ni `M3[2]` ni `M4[2]` ne contiennent `0`.

Une matrice d'adjacence a pour avantage d'être rapide à utiliser et pour inconvénient de consommer beaucoup de mémoire puisque les liens inexistants entre deux sommets non reliés sont tous présents, généralement stockés par un `0`. Par exemple, le labyrinthe des Baleks de la section 2.1. contient `57` cases en comptant l'entrée et le générateur. La matrice d'adjacence contiendra `57²=3249` nombres dont `3135` zéros si j'ai bien compté. La taille explose alors qu'une case a au plus quatre voisins. Une matrice est par ailleurs plus simple pour y stocker des distances.

Une liste d'adjacence a les avantages et inconvénients inverses : si la matrice d'adjacence est creuse (avec beaucoup de zéros), la liste sera courte. En revanche, le temps d'accès est souvent plus long puisque, par exemple, il faut parcourir toute la liste pour se rendre compte qu'un lien est inexistant.

Nous utiliserons les listes dans la deuxième partie et une matrice d'adjacence pour l'algorithme de Dijkstra.

1.3 L'algorithme

Voici un algorithme de parcours en largeur en Python qui fait visiter tous les sommets du graphe, le sommet initial est G :

```
01: #!/usr/bin/python3
02:
03: M={"A":{"B","D"},
04:    "B":{"A","F","G","H"},
05:    "C":{"E","F"},
06:    "D":{"A","E"},
07:    "E":{"C","D","F","H"},
08:    "F":{"B","C","E","H"},
09:    "G":{"B","H"},
10:    "H":{"B","E","F","G"}}
11: afaire={"G"}
12: fait=set()
13: while afaire:
14:   temp=set()
15:   for s in afaire:
16:     temp|=M[s]
17:   temp-=fait
18:   temp-=afaire
19:   fait|=afaire
20:   afaire=set(temp)
21:   print(temp)
```

Libre à vous de modifier l'algorithme selon vos besoins spécifiques et les contraintes et libertés du problème **[1]**, ce dont nous ne nous priverons pas par la suite.

La liste d'adjacence est déclarée de la ligne 03 à la ligne 10. *EXPLIQUEZ, EXPLIQUEZ* - Le sommet `E` est relié aux sommets `C`, `D`, `F` et `H` et il faut les liens réciproques puisque ce graphe n'est pas orienté.

Ligne 12, le sommet initial est `G` et ligne 13, aucun sommet n'est visité au départ.

Ligne 17, on ajoute les sommets voisins des sommets courants à ce qui sera à visiter à l'étape suivante et ligne suivante, on enlève ceux qui ont déjà été visités. Ligne 19, on complète les sommets visités et, ligne suivante, ceux à visiter ; on utilise **`afaire=set(temp)`** et non **`afaire=temp`** parce que dans ce dernier cas, toute modification faite à **`temp`** se répercute immédiatement sur **`afaire`**.

Voici le résultat :

```
> ./pel.py
set(['H', 'B'])
set(['A', 'E', 'F'])
set(['C', 'D'])
set([])
```

Ça marche, le Docte Roux et la fille pas possible Oswim sont prêts à nettoyer l'univers des Baleks avec leur ventouse diabolique.

2 Quelques applications

2.1 Les souterrains des Baleks

Le Docte Roux vient de s'infiltrer dans les souterrains baleks pour les noyer. Il en a eu les plans grâce à Oswim qui s'est faite passer pour un Balek. Il cherche le chemin le plus court vers le générateur qui permet d'écoper l'eau pour le dérégler avec son tournevis conique. On n'a pas besoin pour le moment de l'algorithme de Dijkstra, on le verra en dernière partie.

Note

Les situations de cet article n'ont qu'une réponse, les situations générales peuvent en avoir plusieurs. Les algorithmes proposés ne donnent en fait qu'une solution et non toutes. Par ailleurs, lors d'une passe, si un sommet est rencontré plusieurs fois, seule la dernière est prise en compte.

```
01: #!/usr/bin/python3
02:
03: L=["################",
04:    "E #   #       #",
05:    "# # ### ##### #",
06:    "#       #     #",
07:    "#### #### #####",
08:    "#    #    #   #",
09:    "# # ####### # #",
10:    "G #          # #",
11:    "################"]
12:
13: M=dict()
14: lon=len(L[0])
15: lar=len(L)
16:
17: for j in range(lar):
18:   for i in range(lon):
19:     if L[j][i]=="#":
20:       continue
21:     if L[j][i]=="E":
22:       entree=(j,i)
23:     elif L[j][i]=="G":
24:       generateur=(j,i)
25:     M[(j,i)]=set()
26:     if i>0 and L[j][i-1]!="#":
27:       M[(j,i)].add((j,i-1))
28:     if i+1<lon and L[j][i+1]!="#":
29:       M[(j,i)].add((j,i+1))
30:     if j>0 and L[j-1][i]!="#":
31:       M[(j,i)].add((j-1,i))
32:     if j+1<lar and L[j+1][i]!="#":
33:       M[(j,i)].add((j+1,i))
34:
35: precedent=dict()
36: afaire={entree}
37: fait=set()
38:
39: while generateur not in afaire:
40:   temp=set()
41:   for s in afaire:
42:     tempo=M[s]-fait-afaire
43:     for so in tempo:
44:       precedent[so]=s
45:     temp|=tempo
46:   fait|=afaire
47:   afaire=set(temp)
48:
49: s=generateur
50: pluscourt=[s]
51:
52: while s!=entree:
53:   s=precedent[s]
54:   j,i=s
55:   L[j]=L[j][:i]+"+"+L[j][i+1:]
56:   pluscourt=[s]+pluscourt
57:
58: for l in L:
59:   print(l)
```

La liste d'adjacence est un dictionnaire créé à partir du labyrinthe, lignes 17 à 33, les ordonnées vers le bas et les abscisses vers la droite. Ligne 19, on passe à la case suivante si c'est un mur. Lignes 21 et 23, on gère les cas respectifs de l'entrée et du générateur.

EXPLIQUEZ, EXPLIQUEZ - On teste lignes 27 à 33 les quatre cases autour de la case courante. Chaque sommet est nommé

par le **tuple (j,i)** de ses coordonnées. Pourquoi un tuple ? Parce qu'une liste ne peut pas être une clé d'un dictionnaire ni appartenir à un ensemble au sens de Python.

Ligne 35, dans le dictionnaire **precedent**, si **precedent[so]=s**, cela veut dire que pour aller à **so**, on est passé par **s**, ce qui n'est pas symétrique.

Ligne 39, on n'a pas besoin de parcourir le labyrinthe en entier, seulement d'atteindre le générateur.

Ligne 43, on crée la liste des chemins, ce qu'on retrouvera dans l'algorithme de Dijkstra, pour créer le chemin le plus court des lignes 50 à 53, en remontant du générateur. On en profite pour modifier le labyrinthe et changer chaque case parcourue en un **+**. On affiche le labyrinthe modifié à la fin.

Et la sortie :

```
> ./laby.py
##############
++#   #      #
#+# ### ##### #
#++++  #     #
####+#### #####
#++++#    #   #
#+# ####### # #
G+#         # #
##############
```

Pour savoir le temps nécessaire pour noyer complètement les souterrains si l'eau se déplace d'une case par seconde, il suffit de modifier légèrement le code précédent en remplaçant la gestion de la liste des chemins par un compteur.

```
35: compteur=0
36: afaire={generateur}
37: fait=set()
38:
39: while afaire:
40:   temp=set()
41:   for s in afaire:
42:     temp|=M[s]
43:   temp-=fait
44:   temp-=afaire
45:   fait|=afaire
46:   afaire=set(temp)
47:   compteur+=1
48:
49: print(compteur-1)
```

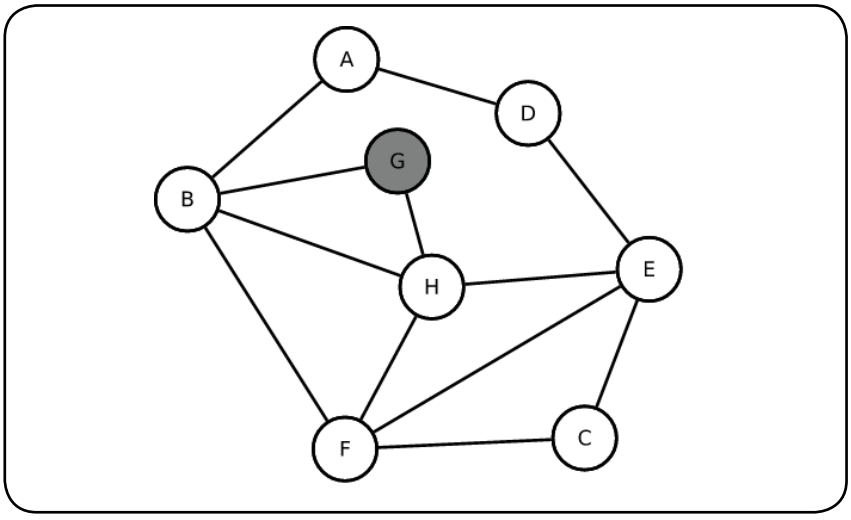


Fig. 6 : Le graphe du temps

Ligne 37, cette fois, on part du générateur qui fuit (comment le rattraper ?).

Lignes 42 à 45, on reprend le code de la section 1.3, plus rapide que le précédent.

Ligne 48, l'eau a franchi une case supplémentaire.

Et ligne 50, en fait, le compteur tourne une fois de trop à l'étape où tout est noyé, c'est-à-dire quand **temp** est vide.

```
Le Docte Roux dispose de 30 secondes pour
s'enfuir à bord du TORDUS :
```

```
> ./labyeau.py
30
```

Je vous laisse modifier le code pour trouver que le temps utile à bord du TORDUS pour toucher à tous les boutons est de 16 secondes.

Notez que ce code sans le compteur peut servir dans un jeu comme **Frozen Bubble [2]** pour rassembler un paquet de bulles de même couleur.

2.2 Le graphe du temps

Les Baleks sont furieux ; ils le sont tout le temps, mais là, encore plus. Le Docte Roux doit donc trouver la ligne du temps la plus longue, la plus éloignée des Baleks pour brouiller sa piste. Il n'est pas question de revenir en arrière, le graphe est donc orienté et n'a pas de cycle. S'il y en a un dans le graphe de la figure 6, par exemple, si on crée un lien de **K** vers **D**, le cycle **D-G-K** crée une boucle et le TORDUS ne s'arrêtera pas.

On doit donc supprimer la gestion des sommets visités de l'algorithme puisqu'on peut passer plusieurs fois par le même, le but étant de trouver le plus long chemin à partir du sommet **A**.

Voici le code :

```
01: #!/usr/bin/python3
02:
03: M={"A":{"B","C"},
04:    "B":{"F","G"},
05:    "C":{"D","E"},
06:    "D":{"F","G","I"},
07:    "E":{"H"},
08:    "F":{"J"},
09:    "G":{"I","J","K","L"},
10:    "H":{"G","K"},
11:    "I":set(),
12:    "J":{"I"},
13:    "K":set(),
14:    "L":set()}
15:
16: afaire={"A"}
17: precedent={c:(None,0) for c in M}
18: compteur=0
19:
20: while afaire:
21:   compteur+=1
22:   temp=set()
```

```
23:   for s in afaire:
24:     for so in M[s]:
25:       precedent[so]=(s,compteur)
26:     temp|=M[s]
27:   afaire=set(temp)
28:
29: compteur-=1
30: for s in precedent:
31:   if precedent[s][1]==compteur:
32:     break
33: print(compteur,s)
34:
35: pluslong=[]
36: while s!=None:
37:   pluslong=[s]+pluslong
38:   s=precedent[s][0]
39: print("-".join(pluslong))
```

Observer ligne 04 — *BFG EXTERMINEZ — Du calme Oswim, tu sais bien que le BFG n'est pas de notre monde et que je n'utilise pas d'arme.* — que du sommet **B** part un lien vers **F** mais pas l'inverse. Certains sommets comme **K** sont des puits, sans sortie.

Ligne 17, cette fois, le dictionnaire **precedent** contient, en plus du sommet précédent, le nombre de sommets parcourus, ainsi, **precedent["C"]=("A",1)**. J'aurais pu utiliser deux listes et jongler avec chr et ord.

Remarquez lignes 26 et 27 qu'on a enlevé la gestion des sommets déjà visités à l'algorithme initial.

Les deux paquets finaux de cinq lignes de code recherchent et affichent respectivement la longueur du plus long chemin et ce plus long chemin.

Et la sortie est :

```
> ./fuite.py
(6, 'I')
A-C-E-H-G-J-I
```

Note

Si **M** était une matrice d'adjacence et **N=M⁶**, le nombre $N_{i,j}$ en notation matricielle ou **N[i][j]** en notation pythonique, est le nombre de chemins qui mènent du sommet **i** au sommet **j** en **6** étapes exactement. Or la matrice d'adjacence **M** du graphe est nilpotente d'indice **7**, ce qui veut dire que **M⁷** est la matrice nulle mais pas **M⁶**. On aurait pu alors calculer les puissances de **M** jusqu'à obtenir la matrice nulle mais on n'aurait pas eu aussi facilement le chemin le plus long pour ce graphe.

2.3 Au poste d'observation

La structure du temps a changé, c'est une espèce de serpillière que le Docte Roux doit maintenant surveiller. Il doit pouvoir intervenir le plus rapidement possible et pour cela, il doit placer le TORDUS en son centre, au point le moins éloigné des extrémités. Il doit pincer la serpillière du temps pour qu'elle soit la plus courte possible quand elle pend : la figure 7 donne la solution, pas la figure 8.

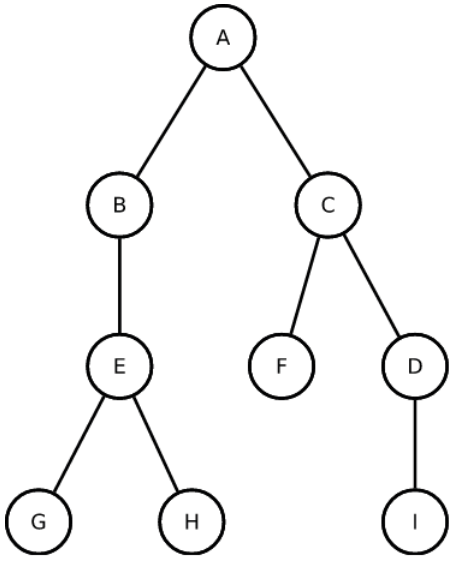


Fig. 7 : Le Docte Roux se place en A car la distance maximale aux extrémités est de 3.

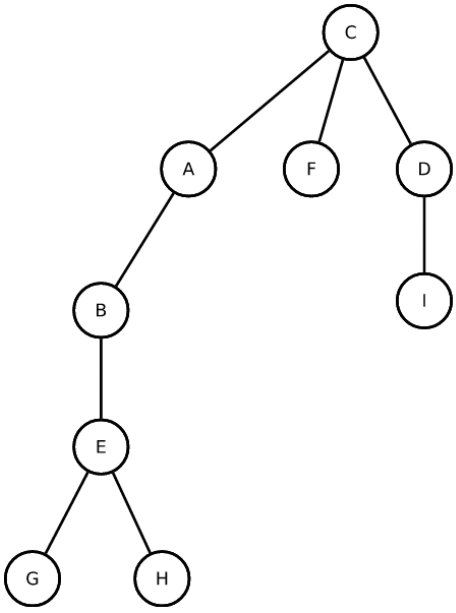


Fig. 8 : Et non en C où la distance maximale est de 4.

Voici le code :

```
01: #!/usr/bin/python3
02:
03: M={"A":{"B","C"},
04:    "B":{"A","E"},
05:    "C":{"A","D","F"},
06:    "D":{"C","I"},
08:    "F":{"C"},
09:    "G":{"E"},
10:    "H":{"E"},
11:    "I":{"D"}}
12:
13: mini=len(M)
14: for s in M:
15:   compteur=-1
16:   afaire={s}
17:   fait=set()
18:   while afaire:
19:     temp=set()
20:     for so in afaire:
21:       temp|=M[so]
22:     temp-=fait
23:     temp-=afaire
24:     fait|=afaire
25:     afaire=set(temp)
26:     compteur+=1
27:   if mini>compteur:
28:     smini=s
29:     mini=compteur
30:
31: print(smini,mini)
```

Ligne 15 *EXPLIQUEZ EXPLIQUEZ* — Au lieu de corriger à la fin comme précédemment, on commence à la place à -1.

Ligne 27, si on trouve un meilleur sommet, on le remplace et on met à jour la plus courte distance.

On obtient :

```
> ./serpillère.py
('A', 3)
```

3 L'algorithme de Dijkstra

Ça y est, les Baleks sont sortis, le Docte Roux sait où et quand. Il peut intervenir en temps et en heure grâce à l'algorithme précédent et avec son ami Staik, le Konkarien au cou court, il lui reste à trouver le plus court chemin dans l'espace. Pour cela, il va utiliser l'algorithme de Dijkstra (prononcer [dɛɪkstra] et non [diʒkstra]), au programme de spécialité mathématique de la série ES du baccalauréat. On veut trouver le chemin le plus court de **A** à **H** dans le graphe de la figure 9.

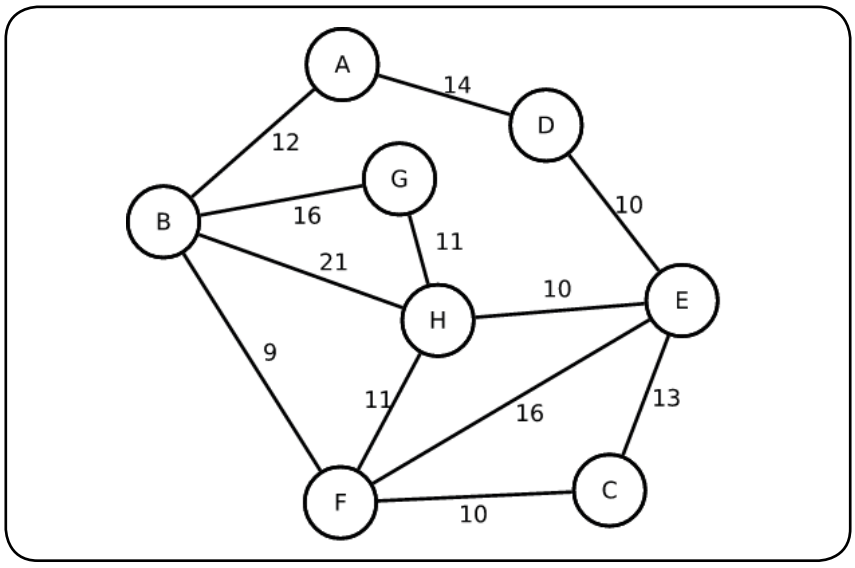


Fig. 9 : Le graphe du sujet de bac ES de métropole de juin 2015 qui servira pour la suite.

L'algorithme est simple :

1. On choisit le sommet disponible le plus proche du départ, qui devient dorénavant indisponible.
2. En partant de ce sommet, on calcule les distances *au départ* des sommets disponibles qui lui sont connectés, en choisissant la plus courte si nécessaire et on garde une trace du sommet choisi.
3. Tant qu'il reste des sommets, on recommence.

Note

Remarques :

1. On ne visite pas tous les sommets disponibles à la fois à chaque passe, seulement un par un : le plus proche du départ ou un seul d'entre eux s'ils sont plusieurs.
2. L'algorithme fonctionne aussi si le graphe est orienté.

L'algorithme donne, dans la liste **distances**, les distances de tous les sommets au sommet initial et dans la liste **precedent**, l'arbre des plus courts chemins, comme d'habitude.

Cette fois, le code présentera la sortie sous la forme du tableau utilisé par les élèves :

NE MANQUEZ PAS
HACKABLE N°10 !

NFC & RFID EN PRATIQUE !
SUR ARDUINO & RASPBERRY PI !

DISPONIBLE
CHEZ VOTRE MARCHAND DE JOURNAUX ET SUR :
www.ed-diamond.com

```
01: #!/usr/bin/python3
02:
03: def form(d,i,fait,inf):
04:   if d=="@":
05:     return " - |"
06:   elif d==inf:
07:     return " ∞ |"
08:   c=str(d)
09:   if i not in fait:
10:     if len(c)==1:
11:       return " "+c+" |"
12:     return " "*(3-len(c))+c+"|"
13:   return " * |"
14:
15: M=[[0,12,0,14,0,0,0,0],
16:    [12,0,0,0,0,9,16,21],
17:    [0,0,0,0,13,10,0,0],
18:    [14,0,0,0,10,0,0,0],
19:    [0,0,13,10,0,16,0,10],
20:    [0,9,10,0,16,0,0,11],
21:    [0,16,0,0,0,0,0,11],
22:    [0,21,0,0,10,11,11,0]]
23:
24: inf=len(M)*(max(max(l) for l in M)+1)
25: precedent=[-1 for s in range(len(M))]
26: distances=[inf for s in range(len(M))]
27: depart=ord("A")-65
28: arrivee=ord("H")-65
29: distances[depart]=0
30: fait=set()
31: afaire=set(range(len(M)))
32:
33: print(" "+" | ".join(chr(i+65) for i in range(len(M)))+" |
choix")
34:
35: while afaire:
36:   dmin=inf
37:   for s in afaire:
38:     if dmin>distances[s]:
39:       dmin=distances[s]
40:       smin=s
41:   print("-"*(4*len(M)+7))
42:   print("".join(form(distances[i],i,fait,inf) for i in
range(len(M)))+" "*3+chr(smin+65))
43:   print("".join(form(chr(precedent[i]+65),i,fait,inf) for i in
range(len(M)))+" préc.")
44:   fait.add(smin)
45:   afaire.remove(smin)
46:   for s in afaire:
47:     if distances[smin]+M[s][smin]<distances[s] and M[s][smin]:
48:       distances[s]=distances[smin]+M[s][smin]
49:       precedent[s]=smin
50:
51: s=arrivee
52: pluscourt=[s]
53: while s!=depart:
54:   s=precedent[s]
55:   pluscourt=[s]+pluscourt
56:
57: print(distances[arrivee])
58: print("-".join([chr(i+65) for i in pluscourt]))
```

La fonction **form**, ligne 03, retourne une cellule du tableau.

La ligne 04 traite le cas où le sommet n'a pas été visité (le **@** a pour valeur ASCII 64=65-1 et **A** 65), la ligne 06 aussi quand on demande d'afficher une distance infinie.

Ligne 09, on tente un affichage centré sur trois caractères… oui, j'aurais pu utiliser la méthode **center** de la classe **str**.

La longueur « infinie » de la ligne 24, **inf**, est supérieure à la longueur maximum d'un chemin hamiltonien (qui passe par tous les sommets et il en existe ici.)

Ligne 33, on affiche l'en-tête du tableau.

Ligne 37, on recherche le sommet disponible le plus proche de **A**.

Ligne 41 à 43, on affiche une ligne du tableau qui affiche le meilleur sommet et sa distance à **A**.

Lignes 46 à 49, on calcule la plus courte distance à **A** des sommets disponibles. Si **M[s][smin]==0**, le lien n'existe pas, ce n'est pas une distance nulle.

Il faudra bien sûr adapter le code précédent si le graphe contient plus de vingt-six sommets ou si les distances sont plus grandes.

Et le beau tableau de sortie :

```
 A | B | C | D | E | F | G | H | choix
-----------------------------------
 0 | ∞ | ∞ | ∞ | ∞ | ∞ | ∞ | ∞ |   A
 - | - | - | - | - | - | - | - | préc.
-----------------------------------
 * | 12| ∞ | 14| ∞ | ∞ | ∞ | ∞ |   B
 - | A | - | A | - | - | - | - | préc.
-----------------------------------
 * | * | ∞ | 14| ∞ | 21| 28| 33|   D
 - | * | - | A | - | B | B | B | préc.
-----------------------------------
 * | * | ∞ | * | 24| 21| 28| 33|   F
 - | * | - | * | D | B | B | B | préc.
-----------------------------------
 * | * | 31| * | 24| * | 28| 32|   E
 - | * | F | * | D | * | B | F | préc.
-----------------------------------
 * | * | 31| * | * | * | 28| 32|   G
 - | * | F | * | * | * | B | F | préc.
-----------------------------------
 * | * | 31| * | * | * | * | 32|   C
 - | * | F | * | * | * | * | F | préc.
-----------------------------------
 * | * | * | * | * | * | * | 32|   H
 - | * | * | * | * | * | * | F | préc.
32
```

Il permet de reconstruire facilement le chemin le plus court :

1. De **H**, on vient de **F**.
2. De la ligne où **F** a été choisi, on regarde pourquoi, c'est parce qu'on vient de **B** à la colonne **F**, parce que le plus court chemin vers **F** vient de **B**, de longueur **21**.
3. Et de **B**, on vient de **A**.

D'où le meilleur chemin **A-B-F-H** de longueur **32**.

Note

J'ai codé cet algorithme (sans le sucre syntaxique pythonique ni le tableau) sur ma calculette de poche Ti86 en simili-BASIC **[3]**. Il faudra adapter le code pour d'autres modèles ou d'autres marques.

Conclusion

Les graphes sont une source inépuisable de problèmes, comme trouver une chaîne hamiltonienne (qui passe par tous les sommets une seule fois chacun) ou eulérienne (qui passe par toutes les arêtes une seule fois chacune), trouver une chaîne hamiltonienne qui maximise une grandeur (problème du voyageur de commerce), déterminer si un graphe possède des cycles ou ses composantes connexes et leur nombre… Certains de ces problèmes se résolvent à l'aide du parcours en largeur, d'autres avec le parcours en profondeur **[4]** (comme l'ordre de succession au trône britannique). Amusez-vous bien ! ■

Références

[1] Certains problèmes sont inspirés par le site https://www.codingame.com/, je vous laisse trouver lesquels. Ne rêvez pas, les solutions ne sont pas optimales : vous n'aurez pas 100% en recopiant mes codes, il faudra travailler encore un peu.

[2] http://www.frozen-bubble.org/ si vous n'avez pas lu le GLMF de l'été.

[3] Code sur ideone.com : http://ideone.com/Iz8cal (i capitale au début et L minuscule à la fin). Il faut supprimer les commentaires, le langage ne sait pas ce que c'est. Par ailleurs, les indices des listes et des matrices commencent à 1.

[4] Colombo T., « De Königsberg à Kalingrad, une petite histoire de graphes », *GNU/Linux Magazine* n°173, juillet/août 2014, p. 24 à 37.

INSPEQTOR : SURVEILLEZ VOS PROCESSUS

Benoît BENEDETTI [Administrateur Systèmes Linux]

Ils demandent une attention de tous les instants, nous réveillent parfois au beau milieu de la nuit pour que l'on s'occupe d'eux, après avoir passé des heures à leur trouver un petit nom (voir http://xkcd.com/910/). « Ils », ce sont nos serveurs et leurs petits processus , que nous allons voir comment monitorer.

Services Monitoring SysAdmin CLI SystemD Google Go

L'OBJECTIF

Inspeqtor est un outil de monitoring en ligne de commandes développé en Go par Mike Perham sous Licence GPL (une version professionnelle payante sous licence commerciale est également disponible), à qui l'on doit aussi la librairie **Sidekiq** pour exécuter des processus Ruby de manière asynchrone. Inspeqtor est très proche de **Monit**, par lequel il a été influencé, ce qui lui a d'ailleurs valu une notice DMCA de la part des responsables de Monit et une fermeture temporaire de son dépôt sur Github: https://news.ycombinator.com/item?id=8416773. Rien de grave car une fois la notice retirée, le dépôt a été rouvert, et Barbara Streisand a fait son effet : cela a fait de la publicité pour Inspeqtor au détriment de Monit qui a perdu des utilisateurs.

Dans cet article, nous allons voir comment installer, configurer et utiliser Inspeqtor, sorti récemment en version stable 1.0. En particulier, nous verrons comment installer Inspeqtor sous Debian, installation qui présente quelques particularités par rapport aux distributions officiellement supportées (Ubuntu Precise et Trusty, et RedHat CentOS 6 et 7). Nous verrons aussi comment configurer Inspeqtor pour un monitoring simple de services.

LES OUTILS

Vous aurez besoin des outils et connaissances suivantes pour suivre cet article :

- GNU/Linux Debian 8.2,
- La ligne de commande,
- **apt-get** ou **aptitude**, le système de gestion de paquets Debian en général.

PHASE 1

Préparation du système de paquets et installation sous Debian

Une simple ligne de commandes suffit pour installer Inspeqtor sur les distributions officiellement supportées. Sous Debian il va par contre falloir procéder manuellement. On commence par créer un fichier de configuration de sources de apt **/etc/apt/sources.list.d/contribsys_inspeqtor.list** :

```
deb https://packagecloud.io/contribsys/inspeqtor/ubuntu/ trusty main
deb-src https://packagecloud.io/contribsys/inspeqtor/ubuntu/ trusty main
```

Ce fichier utilise des dépôts en https, il nous faut donc installer le paquet **apt-transport-https** :

```
$ sudo apt-get install apt-transport-https
```

Puis installer la clé du dépôt :

```
$ curl -s https://packagecloud.io/gpg.key | sudo apt-key add -
```

Avant de pouvoir mettre à jour la base de liste des paquets du système :

```
$ sudo apt-get update
```

On peut ensuite installer Inspeqtor :

```
$ sudo apt-get install inspeqtor
```

PHASE 2 Configuration globale

Le fichier **/etc/inspeqtor/inspeqtor.conf** est le fichier de configuration global de Inspeqtor. Il possède beaucoup d'options, commentées pour la plupart. Ce qu'il vous intéressera de configurer en priorité dans ce fichier ce sont les paramètres de contact par *email* pour que Inspeqtor puisse vous contacter en cas d'alerte. Un fichier **/etc/inspeqtor/inspeqtor.conf** minimal utilisant un compte et un serveur de messagerie externe avec authentification ressemblera à cela :

```
send alerts via email with
  username admin
  password $ecret
  smtp_server smtp.acme.com
  to_email admin@acme.com
```

PHASE 3 Configuration initiale Host

Pour pouvoir démarrer et être utilisé, Inspeqtor doit au minimum avoir une configuration de type **host** dans un fichier **/etc/inspeqtor/host.inq**, contenant le monitoring de l'hôte sur lequel s'exécute Inspeqtor : utilisation cpu, disques et swap. Voici la configuration de base livrée par défaut :

```
check host
  if swap > 20% then alert
  if load:5 > 10 then alert
  if disk:/ > 90% then alert
```

Ce fichier de configuration est clair et compréhensible : chaque ligne représente une condition (mot clé **if**) sur une ressource (la swap, la charge des 5 dernières minutes, l'usage disque de la partition **/**), suivie de la condition à vérifier (supérieure à un pourcentage ou à une valeur d'utilisation), et si la condition est vérifiée, une action à effectuer, après le **then**. Un *check* de type **host** ne permet qu'une action, **alert**, pour envoyer un *email* si la condition est vérifiée.

Notre configuration minimale en place, il nous reste à faire une dernière manipulation avant de pouvoir démarrer Inspeqtor.

PHASE 4 Activation et démarrage du service

Il nous faut à nouveau manuellement configurer le service dans Systemd sous Debian. On commence par copier le fichier de configuration de Systemd installé avec le paquet (on ne fait pas un lien symbolique) :

```
$ sudo cp -a /usr/share/inspeqtor/systemd/inspeqtor.service /lib/
systemd/system/inspeqtor.service
```

On peut ensuite activer le service Inspeqtor au démarrage (seulement si vous avez fait une copie à l'étape précédente, et non un lien symbolique, j'insiste) :

```
$ sudo systemctl enable inspeqtor.service
```

On peut enfin démarrer Inspeqtor :

```
$ sudo systemctl start inspeqtor
```

PHASE 5 Inspeqtorctl

Inspeqtor est fourni avec un script, **inspeqtorctl**, qui permet de dialoguer avec le service **Inspeqtor**, via le socket **/var/run/inspeqtor.sock**. Pour pouvoir passer des commandes au service via le socket, le script **inspectorctl** utilise **netcat**, qu'il faut installer. Mais il faut installer une version de **netcat** qui possède l'option **-U**, pour supporter les sockets Unix. Sous Debian, le paquet traditionnel **netcat** ne propose pas cette option. Il faut installer la variante OpenBSD, **netcat-openbsd** :

```
$ sudo apt-get install netcat-openbsd
```

Pour vous éviter d'utiliser **inspectorctl** en tant que **root**, ajoutez votre utilisateur au groupe **adm**, qui lui peut l'exécuter (vous aurez sûrement besoin de relancer votre session) :

```
$ sudo usermod -a -G adm votre_login_utilisateur
```

Si tout fonctionne comme prévu, vous pourrez utiliser **inspectorctl** pour interagir avec Inspeqtor, comme pour

afficher le statut du monitoring effectué par Inspeqtor par défaut suivant notre fichier **/etc/inspeqtor/host.inq** :

```
$ inspeqtorctl status
Inspeqtor 1.0.0, uptime: 2m16.865240554s, pid: 448

Host: DEBIANVM
  cpu                                0.1%
  cpu:iowait                         0.0%
  cpu:steal                          0.0%
  cpu:system                         0.0%
  cpu:user                           0.1%
  disk:/                             10.0%           90%
  load:1                             0.00
  load:15                            0.05
  load:5                             0.02            10
  swap                               0.0%            20%
```

On retrouve bien dans la première colonne toutes les métriques supportées pour le type **host**, au milieu la valeur en cours pour cette métrique, et une troisième colonne indique ou non qu'un *check* a été défini, avec sa valeur.

La commande **export** d'**inspectorctl** vous permet d'exporter sur la sortie standard la valeur courante des métriques, si vous avez besoin par exemple de les tuber vers un autre processus :

```
$ inspeqtorctl export | python -mjson.tool
{
    "exported_at": 1443530776,
    "host": {
        "metrics": {
            "cpu": {
                "": 0.13333333333333333,
                "iowait": 0,
                "steal": 0,
                "system": 0,
                "user": 0.13333333333333333
            },
…
```

La commande **show** prend en paramètre un type de *check* et la métrique à afficher pour ce type de *check*, sous forme de graphique en mode console. Par exemple, pour afficher la métrique **cpu:system** de l'hôte :

```
$ inspeqtorctl show host cpu:system
DEBIANVM cpu:system min 0.1% max 20.4% avg 1.4%
ÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐ
ÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐ
```

Attention, ces métriques et graphiques sont remis à zéro à chaque redémarrage de Inspeqtor.

PHASE 6 Service générique

En plus de l'hôte, Inspeqtor peut monitorer tout service, à partir du moment où celui-ci est géré par un gestionnaire de services récent. **Init** n'est donc pas supporté. Sous Debian 8, le gestionnaire officiel est devenu SystemD, nous avons d'ailleurs créé précédemment le fichier **/lib/systemd/system/inspeqtor.service** pour gérer Inspeqtor comme service SystemD, nous pouvons donc monitorer Inspeqtor par Inspeqtor ! Le monitoring d'un service inclut seulement deux familles de métriques, **cpu** et **memory**, et des métriques différentes que celles pour **host**. On commence par créer un fichier pour ce service dans **/etc/inspeqtor/services.d/**. Un tel fichier **/etc/inspeqtor/services.d/inspeqtor.inq** pourra ressembler à :

```
check service inspeqtor
  if memory:rss > 100m then alert
  if cpu:user > 80% then restart
```

Pour monitorer un service, dans ce fichier, il faut, en première ligne, déclarer un *check* de type **service**, avec pour argument le nom du service à vérifier, nom qui doit correspondre à celui sous lequel votre service est connu de votre gestionnaire de services (ici **inspeqtor**, comme définit dans le fichier de configuration SystemD). Viennent ensuite les conditions sur les métriques à vérifier, similaires au fichier *host.inq*. La différence pour un *check* de type *service*, c'est que vous avez comme action à disposition, en plus de **alert**, les actions **reload** et **restart**.

Nous avons modifié la configuration de Inspeqtor, on n'oublie pas de recharger le service pour qu'elle soit prise en compte :

```
$ sudo systemctl reload inspeqtor.service
```

La commande **status** de **inspeqtorctl**, en plus des métriques **host**, vous affiche désormais les métriques et conditions propres au service défini :

```
$ inspeqtorctl status
...
Service: inspeqtor [Unknown/0]
  cpu:system                       0.0%
  cpu:total_system                 0.0%
  cpu:total_user                   0.0%
  cpu:user                         0.0%            80%
  memory:rss                       -0.00m          100m
```

Note

Inspeqtor peut redémarrer un service en cas de condition de dépassement d'un seuil sur une métrique, comme dans l'exemple précédent. Par contre, si le service est indisponible, Inspeqtor, à la différence d'un outil comme Monit, ne redémarrera pas ce service. C'est un choix de conception : Inspeqtor juge que c'est votre gestionnaire de services le mieux placé pour ce travail. Vous recevrez néanmoins un *email* de la part de Inspeqtor dès qu'il s'apercevra qu'un service monitoré ne répond plus.

PHASE 7 Services officiellement supportés

Inspeqtor est livré en standard avec le support pour différents services : autrement dit, en plus des familles de métriques **cpu** et **memory** disponibles par défaut pour tout service monitoré, vous pouvez en plus utiliser des métriques propres à ces services officiellement supportés. Par exemple, installons un serveur MySQL à monitorer :

```
$ sudo apt-get install -y mysql-server
```

Il nous faut également au préalable modifier la configuration de MySQL, car Inspeqtor s'attend à trouver son fichier de pid dans **/var/run/mysql.pid** ou **/var/run/mysql/mysql.pid**, ce qui n'est pas le chemin par défaut sous Debian. On commence par arrêter MySQL :

```
$ sudo systemctl stop mysql.service
```

On modifie la configuration du chemin du fichier pid, dans le fichier **/etc/mysql/my.cnf**, ligne 36 :

```
pid-file        = /var/run/mysql/mysql.pid
```

On renomme le dossier original vers ce nouveau dossier :

```
$ sudo mv /var/run/mysqld/ /var/run/mysql/
```

On redémarre MySQL :

```
$ sudo systemctl start mysql.service
```

Puis on peut définir un fichier **mysql.inq** pour notre service, toujours dans le dossier **/etc/inspeqtor/services.d** :

```
check service mysql
  with username root, socket /var/run/mysqld/mysqld.sock
  if memory:rss > 2g then alert
  if mysql:Queries > 100/sec for 2 cycles then alert
  if mysql:Connections > 1/sec for 2 cycles then alert
```

Cette fois-ci, nous avons utilisé la famille de métriques **mysql**, et des métriques associées, disponibles pour ce service. La commande **inspeqtorctl** devrait vous les afficher :

```
$ sudo systemctl reload inspeqtor.service
$ inspeqtorctl status
...
Service: mysql [Up/12780]
  cpu:system                    0.0%
  cpu:total_system              0.0%
  cpu:total_user                0.0%
  cpu:user                      0.0%
memory:rss                     45.79m          2g
  mysql:Connections             3.0             1/sec
    mysql:Queries               6.0             100/sec
```

Cette famille **mysql** possède de nombreuses autres métriques. Les services officiellement supportés sont **memcached**, **mysql**, **redis**, **nginx** et **postgresql**. Pour connaître les différentes métriques des différents services officiellement supportés, visitez le wiki officiel (https://github.com/mperham/inspeqtor/wiki/Daemon-Specific-Metrics), voire allez jeter un coup d'œil dans le code source du service en question (https://github.com/mperham/inspeqtor/blob/master/metrics/daemon/mysql.go#L202), certaines métriques ne sont pas documentées...

PHASE 8 Opérations de Maintenance

Inspeqtor mis en place, celui-ci peut s'avérer gênant si vous avez une tâche de maintenance planifiée : mise à jour, déploiement d'une nouvelle version d'une application, sauvegarde, etc. Dans ces cas, un service peut avoir besoin d'être arrêté, ou bien il va être gourmand en ressources, et Inspeqtor va vous submerger *d'emails*. Plutôt que d'utiliser la méthode *bourrine* d'arrêter le service Inspeqtor, la commande **inspeqtorctl** vous permet de mettre les notifications d'Inspeqtor en pause :

```
$ inspeqtorctl start deploy
```

Inspeqtor est désormais en pause (cinq minutes par défaut, durée que vous pouvez configurer dans **/etc/inspeqtor/inspeqtor.conf** avec le paramètre **deploy_length**) :

```
$ inspeqtorctl status

Inspeqtor 1.0.0, uptime: 6.47015966s, pid: 13746
Silenced until: 2015-10-03T15:06:37+02:00
...
```

Une fois votre tâche de maintenance terminée, réactivez les notifications Inspeqtor :

```
$ inspeqtorctl finish deploy
```

LE RÉSULTAT

Nous avons maintenant un monitoring de base mis en place grâce à Inspeqtor. Inspeqtor est dogmatique : si ses choix de conception vous semblent trop limités, un système de monitoring plus complet comme **Nagios**, **Shinken** ou **Zabbix** sera plus conseillé. Si votre parc est limité, et que vous n'avez pas le temps d'investir dans l'apprentissage et la maintenance d'une solution de monitoring avancée, Inspeqtor est une très bonne alternative, simple et efficace. ■

SERVICE DE CALCUL D'ITINÉRAIRE

Daniel JAKOTS [Étudiant et contributeur au libre]

Il peut être très utile de savoir comment se rendre d'un point A à un point B. C'est encore plus utile quand on connaît le temps de trajet estimé. Installons ça chez nous.

Mots-clés : Openstreetmap, Ubuntu, Adminsys, Calcul d'itinéraire, Backend, Système d'information géographique

Résumé

Il existe plusieurs logiciels qui permettent de prévoir un itinéraire utilisant les données issues d'Openstreetmap. Les deux plus connus, car libres, sont GraphHopper et OSRM. C'est ce dernier que nous allons installer car il est très rapide. Pour ceux auxquels cela parle, il utilise une méthode des mathématiques appliquées nommée en anglais *Contraction hiérarchies* [1]. Ce logiciel est écrit en C++ mais utilise des fichiers de configuration en Lua (mais n'ayez pas peur, vous n'aurez pas à en écrire un, au pire à le personnaliser un petit peu).

Dans cet article, nous allons installer (en compilant) **OSRM** qui va nous permettre de calculer des itinéraires. Ensuite, nous verrons comment le configurer, où trouver les données et comment les traiter avec OSRM. Enfin, nous aurons un service Web qui répondra aux requêtes d'itinéraire.

1 Installation

Comme pour le précédent article **[2]**, cet article se base sur l'utilisation d'Ubuntu 14.04.

Pour l'instant il n'y a pas de paquets à ma connaissance pour OSRM donc nous allons le compiler ; c'est un logiciel léger donc ça ne va pas prendre trop de temps, rassurez-vous.

Nous allons commencer par installer les logiciels nécessaires à son installation :

```
$ sudo apt-get install build-essential git cmake pkg-config libprotoc-dev libprotobuf8
protobuf-compiler libprotobuf-dev libosmpbf-dev libpng12-dev libbz2-dev libstxxl-dev
libstxxl-doc libstxxl1 libxml2-dev libzip-dev libboost-all-dev lua5.1 liblua5.1-0-dev
libluabind-dev libluajit-5.1-dev libtbb-dev
```

Le développement a lieu sur GitHub et nous allons télécharger les sources sur ce site. Même si les développeurs ont l'air de faire en sorte que la branche master fonctionne correctement, je trouve plus sûr d'utiliser une *release*.

Il faut donc aller sur https://github.com/Project-OSRM/osrm-backend/releases afin de trouver la dernière *release* puis on récupère l'archive **tar.gz**. À l'heure où ces lignes sont écrites, la dernière version est la **v4.7.1**. Je vous laisse soit utiliser cette version soit modifier les commandes données pour qu'elles s'appliquent à la version que vous téléchargez. Nous allons créer un répertoire pour mettre les sources. Je vous laisse choisir celui que vous voulez, libre à vous d'utiliser celui que je crée ou non.

```
$ mkdir ~/osrm-src && cd ~/osrm-src
$ wget https://github.com/Project-OSRM/osrm-backend/archive/v4.7.1.tar.gz
$ tar xvzf v4.7.1.tar.gz
$ cd osrm-backend-4.7.1
```

Maintenant, nous allons compiler OSRM. Les commandes sont normalement valables, quelle que soit la version téléchargée. On va créer un répertoire pour compiler OSRM puis générer un *makefile* à l'aide de **cmake**. Enfin, on va installer OSRM afin de pouvoir l'utiliser :

```
$ mkdir build && cd build
$ cmake ..
$ sudo make install
```

Maintenant, OSRM devrait être installé si tout s'est déroulé correctement.

2 Configuration

Si vous avez suivi l'installation du serveur de tuiles dans l'article précédent **[2]**, vous devriez déjà avoir les données Openstreetmap dans un fichier **pbf**. Sinon il faut aller télécharger sur http://download.geofabrik.de/ pour la région que vous souhaitez. Pour changer, je vais utiliser la Bretagne comme exemple pour cet article.

Nous allons créer un répertoire de travail. Encore une fois, vous pouvez utiliser celui que vous voulez.

```
$ mkdir ~/osrm && cd ~/osrm
```

OSRM a besoin d'un fichier de profil en Lua pour fonctionner. Rassurez-vous, il y a plusieurs fichiers de profil fournis avec. Ils sont disponibles dans le dossier **profiles** présent dans l'archive d'OSRM qu'on a téléchargé. Supposons que vous souhaitiez faire du calcul d'itinéraire pour les voitures. On prendra alors le fichier nommé **car.lua** puis nous créerons un lien symbolique en l'appelant **profile.lua** :

```
$ ln -s ~/osrm-src/osrm-backend-4.7.1/
profiles/car.lua profile.lua
```

OSRM est très rapide car il fait une grande partie des calculs avant, ce qu'on appelle du *pre-processing*. Ce *pre-processing* utilise une bibliothèque pour gérer l'espace disque **[3]**. On va donc créer un dossier de travail puis créer un fichier pour dire combien on alloue sous la forme **disk=path,capacity,access**. J'ai choisi arbitrairement **1G** et j'ai laissé les autres options aux valeurs conseillées sur la page précédemment indiquée :

```
$ echo "disk=/tmp/stxxl,1G,syscall" > .stxxl
```

Nous allons exécuter deux commandes afin de lancer les calculs du *pre-processing*. La première commande va préparer les données pour OSRM et la deuxième va calculer les hiérarchies qui permettent au moteur de routage de trouver le chemin le plus court en un temps très rapide.

Puisque ces deux commandes consistent à faire beaucoup de calculs, la rapidité d'exécution de ces commandes dépend de la taille des données en entrée (si vous calculez pour une région de la France, ça ne prendra évidemment pas le même temps que pour toute la planète) et, comme toujours, de la puissance de la machine.

OSRM met les fichiers résultants de son traitement dans le dossier où on lui a donné le fichier source, donc le mieux est de le mettre dans le dossier de travail, qui est dans mon cas **~/osrm** :

```
$ osrm-extract bretagne-latest.osm.pbf
```

Cette commande a généré un fichier portant le même nom que celui que vous aviez donné en entrée mais qui a comme extension **.osrm**. C'est ce fichier que nous allons donner à la commande suivante :

```
$ osrm-prepare bretagne-latest.osrm
```

Maintenant, OSRM a tout ce dont il a besoin. On va donc utiliser la commande pour lancer le service qui va répondre aux requêtes :

```
$ osrm-routed bretagne-latest.osrm
```

Cette commande va alors écouter le port **5000** afin de répondre aux requêtes. Si tout s'est déroulé correctement, la dernière ligne devrait être :

```
[info] running and waiting for requests
```

Il suffit alors de prendre son navigateur préféré (donc **Firefox** :)) pour aller demander un itinéraire afin de vérifier son bon fonctionnement. Je vous invite à vous munir des coordonnées géographiques (en degrés décimaux) de deux points situés dans la zone que vous avez choisie.

On va donc à l'url http://adresse-ip:5000/viaroute?loc=lat1,long1&loc=lat2,long2 en remplaçant les mots **lat1** et **long1** par la latitude et la longitude du premier point et **lat2** et **long2** par la latitude et la longitude du deuxième point. Dans mon cas, pour la Bretagne ça donne http://xxx.xx.xx.x:5000/viaroute?loc=48.4574,-4.3650&loc=48.4009,-4.1188 (n'essayez pas d'y voir un quelconque signe des Illuminatis, ce sont deux points au hasard pour vérifier qu'OSRM me donne bien une route).

Conclusion

On a donc installé le service qui permet de faire des calculs d'itinéraire. Il ne reste plus qu'à installer un *frontend* (tel que **osrm-frontend [4]**) pour afficher la route directement sur une carte. ■

Références

[1] Méthode des contractions hiérarchies : https://en.wikipedia.org/wiki/Contraction_hierarchies

[2] JACKOTS D., « Installer son propre serveur de tuiles », *GNU/Linux Magazine* n°188, novembre 2015.

[3] STXXL : http://stxxl.sourceforge.net/tags/master/install_config.html

[4] Frontend pour OSRM : https://github.com/Project-OSRM/osrm-frontend

INSTALLATION D'UN MOTEUR DE RECHERCHE POUR DU GÉOCODAGE

Daniel JAKOTS [Étudiant et contributeur au libre]

Le géocodage c'est associer des coordonnées géographiques à une adresse. Cela permet de savoir avec précision où se situe l'adresse. Il faut donc avoir les adresses et un moyen de chercher dans cette énorme base de données.

Mots-clés : Openstreetmap, Redis, Géocodage, Adminsys, Adresses

Résumé

Pendant longtemps, les bases d'adresses ont été la chasse gardée de certaines entreprises (comme La Poste) ou institutions (IGN). Début 2014, Openstreetmap a initié le projet BANO (Base d'Adresse Nationale Ouverte) afin de fournir ces données sous la licence utilisée par le projet, la licence *Open Database License* (ODbl). Suite à ce projet est née la Base d'Adresse Nationale (BAN) qui est un projet collaboratif entre Openstreetmap, IGN, La Poste et Etalab. Il y a donc deux jeux de données : la BAN et la BANO qui est un peu moins complète que la BAN mais plus ouverte (d'un point de vue de la licence) et plus facilement accessible. Nous allons utiliser les données de la BANO sous licence ODbl.Avoir les données c'est bien. Pouvoir faire des recherches, c'est mieux ! Nous allons donc voir comment installer addok [1] qui est un moteur de recherche développé par l'Etalab.

Dans ce court article, nous allons voir comment installer addok, un moteur de recherche d'adresses. Nous verrons ensuite où trouver les données et comment les mettre dans ce moteur de recherche. Il n'y aura ensuite plus qu'à lancer le service Web pour répondre aux requêtes.

1 Installation d'Addok

Il y a plus de 20 millions d'adresses dans la BANO, donc pour avoir des performances acceptables l'auteur a choisi d'utiliser **Redis [2]** qui est une base NoSQL qui stocke ses données en RAM. Cela implique que :

1. c'est rapide ;
2. ça consomme beaucoup de RAM.

Le reste du logiciel est développé en Python3 (le futur est arrivé \o/). Nous allons donc travailler dans un **virtualenv**.

On va installer les logiciels sur notre ubuntu 14.04 :

```
$ sudo apt-get install redis-server
python3.4 python3.4-dev python-pip
python-virtualenv virtualenvwrapper
```

On va se créer un dossier dans lequel on va faire notre tambouille :

```
$ mkdir ~/ban && cd ~/ban
```

On crée ensuite notre **virtualenv** qui va accueillir addok :

```
$ virtualenv addok --python=/usr/bin/
python3.4
```

On active le **virtualenv** puis on installe addok :

```
$ source addok/bin/activate
$ pip install addok
```

Maintenant, addok devrait être installé.

NE MANQUEZ PAS

LINUX PRATIQUE HORS-SÉRIE N°35 !

CRÉEZ UN BLOG WORDPRESS QUI VOUS RESSEMBLE !

DISPONIBLE DÈS LE 29 JANVIER

CHEZ VOTRE MARCHAND DE JOURNAUX ET SUR :

www.ed-diamond.com

2 Téléchargement des données

On va donc télécharger les données de la BANO. Je vous laisse choisir quelles données vous souhaitez avoir dans votre moteur de recherche, mais faites attention, plus on met de données, plus Redis va consommer de la mémoire vive. Pour toute la France, Redis a besoin de plus de 20 Gio de RAM.

Pour télécharger les données du département 14, on utilise la commande suivante :

```
$ wget http://bano.openstreetmap.fr/data/bano-14.json.gz
```

Notez que si vous voulez télécharger plusieurs départements, vous n'êtes pas obligé de taper plusieurs fois la commande, vous pouvez mettre les numéros entre accolades séparés par des virgules. Par exemple, pour télécharger les données de la Basse-Normandie, soit les départements **14**, **50**, et **61**, on peut le faire comme ça :

```
$ wget http://bano.openstreetmap.fr/data/bano-{14,50,61}.json.gz
```

On décompresse le(s) fichier(s) téléchargé(s) et on met tout dans un fichier qu'on va importer :

```
$ gzip -d *gz
$ cat *json > data.json
```

2.1 Code INSEE

Il se peut que vous ayez besoin des codes INSEE des communes. Pour cela, il suffit de rajouter un fichier de configuration qu'on va appeler **local.py**. Dedans il suffit de mettre :

```
EXTRA_FIELDS = [
            {'key': 'citycode'},
            ]
FILTERS = ["type", "postcode", "citycode"]
```

Addok appelle le code INSEE « citycode » (c'est ce que vous devrez chercher dans le JSON contenant les réponses). Ensuite, on indique ce fichier à addok avec :

```
$ export ADDOK_CONFIG_MODULE=chemin/vers/le/fichier/local.py
```

Donc si **local.py** est dans le répertoire courant, il suffit de faire :

```
$ export ADDOK_CONFIG_MODULE=local.py
```

Lorsque vous lancerez les deux commandes suivantes, vous devriez avoir le message qui vous indique qu'il l'a bien pris en compte :

```
Loaded local config from local.py
```

3 Import des données

On va pouvoir importer les données maintenant que tout est en place :

```
$ addok batch data.json
```

Cela va mettre un petit moment, plus ou moins long suivant la quantité de données que vous lui faites ingérer. Vous allez aussi voir la quantité de RAM utilisée exploser. Pire que quand on lance du Java... enfin presque.

Pour les départements du **14**, **50** et **61**, la vm ubuntu qui me sert à vérifier l'intégralité des commandes que je donne consomme 994Mio de RAM.

Maintenant que les données sont dans Redis, on va calculer les n-gram **[3]**. Un n-gram est une partie de **n** éléments d'une suite de mots. Pour faire simple, c'est l'autocomplétion, c'est-à-dire que quand on lui donne « infant » il va nous proposer les adresses ayant le mot « infanterie ». Cette étape va prendre un peu de temps. :

```
$ addok ngrams
```

4 Lancement du service Web

On va lancer **gunicorn** qui va gérer les requêtes. Par défaut il écoute sur **localhost** ce qui n'est pas très pratique pour un serveur ; on surcharge donc cette adresse. Par défaut il écoute sur le port **8000** :

```
$ gunicorn addok.server:app --bind 0.0.0.0
```

On teste avec un navigateur en allant sur http://adresse-ip:8000/search/?q=Ville en remplaçant **Ville** par une ville présente dans les données que vous lui avez fournies. Dans mon cas, http://adresse-ip:8000/search/?q=Caen renvoie un fichier json avec toutes les informations qu'il trouve liées au mot **Caen**.

Maintenant, pour lancer **gunicorn**, le plus simple est d'écrire un script d'init pour que **systemd** (ou un autre init si vous êtes réticent au progrès :)) le lance au démarrage.

Conclusion

On vient donc d'installer un moteur permettant d'effectuer des recherches dans toutes les adresses que l'on souhaite, et ce très rapidement. ■

Références

[1] Dépôt Github d'addok : https://github.com/etalab/addok

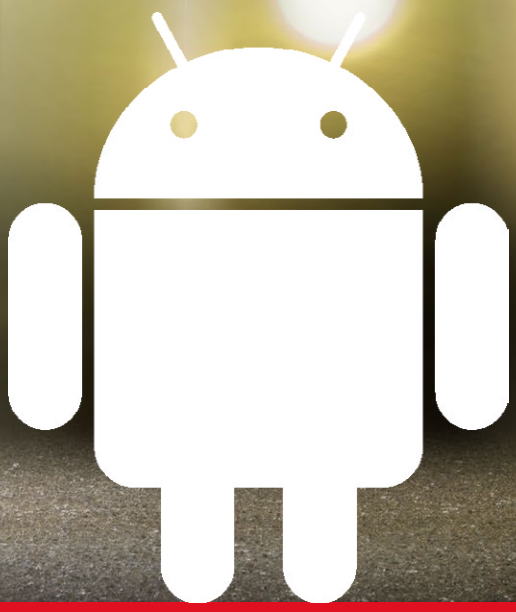
[2] Page Wikipédia anglophone de Redis : https://en.wikipedia.org/wiki/Redis

[3] Page Wikipédia anglophone des n-gram : https://en.wikipedia.org/wiki/N-gram

NE MANQUEZ PAS

GNU/LINUX MAGAZINE HORS-SÉRIE N°82 !

LE GUIDE POUR
APPRENDRE À DÉVELOPPER DES APPLICATIONS ANDROID



DISPONIBLE DÈS LE 18 JANVIER

CHEZ VOTRE MARCHAND DE JOURNAUX ET SUR :

www.ed-diamond.com

POINT D'ACCÈS WIFI

Arnaud FÉVRIER [Maître de conférences en Informatique, Aix Marseille Université, laboratoire I2M, équipe eRISCS]

Nous présentons comment mettre en place un point d'accès WiFi à partir d'un serveur GNU/Linux et d'une clef WiFi. Le coût du matériel nécessaire pour héberger un serveur domestique est inférieur à la centaine d'euros et certains ont une consommation électrique inférieure à 5 Watts. Il est donc envisageable d'utiliser ces équipements pour sécuriser son réseau domestique.

Mots-clés : Wifi, Access point, Routeur, Firewall, hostapd, dnsmasq

Résumé

Nous présentons la mise en place d'une borne d'accès WiFi sur un système GNU/Linux muni d'une interface WiFi. Cette installation peut se faire sur un ordinateur basse consommation, comme un Raspberry pi, par exemple. Il s'agit de mettre en place la liaison de données et l'authentification wpa2 avec hostapd. La liaison IP et dns sera établie avec un serveur dnsmasq. Enfin, nous mettrons en place le routage et la translation d'adresses.

Les équipements WiFi sont de plus en plus nombreux. À la maison, nous allons trouver les smartphones et tablettes, mais aussi des consoles de jeux et de plus en plus d'équipements « connectés ». Ces équipements sont vendus avec des fonctionnalités, certes attractives, mais avec quelques défauts majeurs : les fonctionnalités sont peu, voire pas, documentées et la sécurité n'est prise en compte que dans le discours commercial. Certains équipements sont même conçus avec des vulnérabilités abracadabrantesques **[1]** pour fournir des renseignements aux services secrets de tous les pays et éventuellement établir des VPN non officiels **[2]**.

Bref, il est temps de reprendre le contrôle sur les flux d'informations domestiques avant que votre réfrigérateur ne se fasse inculper pour avoir été source d'une attaque **[3]**.

Nous allons montrer comment mettre en place un point d'accès WiFi. Cet accès pourra être utilisé pour laisser un accès WiFi à tous les équipements non sécurisés. Il pourra aussi permettre, mais c'est un travail ardu de vérifier le comportement de ces appareils en validant le trafic réseau.

Nous supposons donc que nous avons un serveur Debian stable (Jessie) équipé d'une interface WiFi fonctionnelle. Cette méthode s'applique, bien entendu, à toute distribution GNU/Linux suffisamment récente. Il n'y a pas d'interface graphique sur cette machine qui sera installée loin d'un écran. Nous présentons la configuration dans la figure 1.

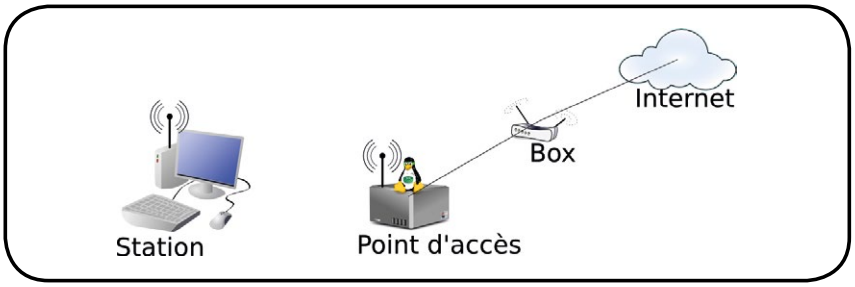


Fig. 1 : Plan de la maquette.

Nous allons installer en premier hostapd qui permettra d'établir un lien entre notre serveur et un équipement WiFi (une tablette ou un portable). Quand hostapd sera validé nous installerons dnsmasq qui nous fournira un serveur dhcp permettant

de fournir des adresses aux équipements distants. Dnsmasq fournit aussi un serveur dns qui permet d'utiliser les noms des machines du réseau local. Enfin, nous présenterons les règles élémentaires de routage pour mettre en place un réseau *natté*.

1 Hostapd : la liaison WiFi

Hostapd est le programme qui permet de transformer un système GNU/Linux en point d'accès WiFi. Il permet d'établir la liaison de données entre le serveur et le, voire les, mobiles. Pour la mise en place du réseau WiFi, il faudra fournir une adresse IP, par exemple avec un serveur DHCP puis mettre en place le routage.

L'installation d'hostapd est simple. Il faut auparavant avoir une interface réseau WiFi utilisable. Nous supposerons que son nom est **wlan0**. Cette interface doit avoir une adresse IP fixe définie par la configuration du serveur, par exemple dans le fichier **/etc/network/interfaces** :

```
allow-hotplug wlan0
iface wlan0 inet static
  address 10.3.91.1
  netmask 255.255.255.0
  broadcast 10.3.91.255
```

La commande **ip addr show wlan0** doit montrer les paramètres et indiquer que la carte est active. Nous pouvons maintenant installer hostapd :

```
# apt-get install hostapd
```

Le fichier **/etc/default/hostapd** définit le nom du fichier de configuration qui sera utilisé. Il est possible d'utiliser des fichiers d'exemples trouvés sur Internet ou celui livré directement avec hostapd (sous Debian : **/usr/share/doc/hostapd/examples/hostapd.conf.gz**). C'est ce dernier que nous allons utiliser.

Ayant nommé mon serveur Hermès, le gardien des routes et des carrefours, je vais nommer le fichier de configuration : **/etc/hostapd/hermesAP.conf** (voir sur le Github de GNU/Linux Magazine).

Je modifie le fichier **/etc/default/hostapd** :

```
DAEMON_CONF="/etc/hostapd/hermesAP.conf"
```

Ce fichier contient beaucoup de commentaires permettant de configurer en détail hostapd. Nous allons faire l'impasse sur beaucoup de possibilités pour nous concentrer sur une mise en place relativement simple. L'impact de tous ces éléments sur la sécurité devrait être évalué.

Si, par mégarde, juste après avoir copié le fichier, le serveur redémarre, alors un réseau nommé **test** apparaît. Pour l'instant le réseau IP n'est pas configuré, donc la plupart des appareils n'accepteront pas de se connecter. Il doit être néanmoins possible d'utiliser cette connexion pour tenter de se connecter sur le serveur. Néanmoins avec une Debian Jessie à jour et sans erreurs de configuration, il ne devrait pas y avoir d'intrusion.

1.1 Le fichier de configuration

Nous allons lister ici les éléments que nous modifions ou que nous vérifions dans le fichier de configuration :

- **interface** : il faut vérifier si le nom d'interface est le bon (ici **wlan0**) ;
- **ssid** (*Service Set Identification*) : l'identifiant de notre réseau (ici **HermesAP**) ;
- **canal** : le canal (**channel**) utilisé. La valeur **0** permet de laisser hostapd choisir en fonction de l'état courant lors du lancement du service. Certaines valeurs peuvent ne pas être compatibles avec certains équipements ;
- **MIMO** : si votre interface le permet, vous pouvez activer le *Multiple In Multiple Out* (**ieee80211n=1**) ;
- **WPA** : si vous utilisez le wpa2, alors **wpa=2** ; indiquer ensuite la phrase de passe et le gestionnaire de clef utilisé.

S'il n'y a pas d'erreur, alors vous devriez voir le réseau apparaître sur votre station WiFi après un **service hostapd restart**. Sur un téléphone Android, vous entrez la phrase de passe et il devrait bloquer sur la récupération de l'adresse IP.

Il peut y avoir quelques soucis lors de la mise au point. J'ai parfois dû désactiver la carte WiFi et la réactiver pour accepter les nouveaux paramétrages. Dans le fichier de log (**tail -f /var/log/syslog**), il y a des messages très utiles. Quand il faut couper et relancer l'interface WiFi, il y a des messages du genre :

```
wlan0 : link is not ready
```

Sinon, si la connexion avance, il est possible de voir le succès de wpa :

```
WPA : pairwise key handshake completed
```

Nous allons maintenant voir ce qui se passe sur l'interface WiFi.

1.2 Que se passe-t-il sur le réseau ?

J'installe **tshark** et le lance sur l'interface **wlan0** : **tshark -n -i wlan0**. Voici un extrait du trafic initial :

```
 1  LgElectr -> Broadcast XID 20 Basic Format; Type 1 LLC (Class
I LLC);
 2  Tp-LinkT -> LgElectr EAPOL 113 Key (Message 1 of 4)
 3  LgElectr -> Tp-LinkT EAPOL 135 Key (Message 2 of 4)
 4  Tp-LinkT -> LgElectr EAPOL 193 Key (Message 3 of 4)
 5  LgElectr -> Tp-LinkT EAPOL 113 Key (Message 4 of 4)
 6 :: -> ff02::XXX ICMPv6 78 Neighbor Solicitation for fe80::XXX
12 0.0.0.0 -> 255.255.255.255 DHCP 351 DHCP Discover
13 0.0.0.0 -> 255.255.255.255 DHCP 351 DHCP Discover
```

Ce trafic montre le début de la tentative de connexion. Cela commence par un message de contrôle Ethernet, puis quatre paquets pour l'authentification wpa, des requêtes pour la configuration IPv6 et enfin des requêtes ICMP. Bien entendu, pas de réponse. Nous avons prouvé que la connexion au niveau liaison de données entre les deux équipements se passe bien, mais qu'il n'y a pas de réseau IP. Nous allons donc configurer IP dans la prochaine section.

Note

Ce qui a été laissé de côté

Nous avons privilégié l'utilisation en mode routeur, mais il est possible d'utiliser un mode *bridge*. Cela peut faciliter la mise en réseau entre une console de jeu branchée sur Ethernet et une utilisant le WiFi.

Radius permet de mettre en place des points d'accès avec une authentification par *login* et mot de passe dans une interface Web. C'est ce qui est fait dans certains hôtels.

Hostapd peut respecter les limites légales concernant l'émission d'ondes dans l'espace public. Ainsi, si un équipement originaire d'un pays permissif est installé dans un pays restrictif, alors la réglementation sera respectée.

Il est possible de limiter le nombre de stations qui peuvent se connecter simultanément ou de filtrer les adresses MAC des stations autorisées (ou non) à se connecter.

Et quelques autres éléments que nous ne commenterons pas ici...

2 Dnsmasq : les paramètres réseau

Si la liaison se passe bien, il faut établir les paramètres réseau. Nous gardons IPv4 pour le moment. Il faut donc un serveur dhcp. Le serveur de l'ISC **[4]** (*Internet Systems Consortium*) est très bien, mais dnsmasq fournit simultanément les services dhcp (pour fournir les paramètres IP) et dns (pour convertir les noms locaux en adresses IP).

Nous allons donc installer et configurer dnsmasq :

```
# apt-get install dnsmasq
```

L'installation lance le démon, mais celui-ci n'est pas configuré pour notre réseau. Le fichier **/etc/default/dnsmasq** permet de préconfigurer quelques paramètres. Comme Dnsmasq est déjà activé, nous ne modifierons pas ce fichier.

2.1 Notre réseau

Comme la configuration de la carte WiFi le montre, notre réseau a les paramètres suivants :

- **Adresses IP** : **10.3.91.X**
- **Masque** : **255.255.255.0**
- **Routeur** : **10.3.91.1** (l'adresse de l'interface WiFi)
- **DNS** : **10.3.91.1** (mais, ce n'est pas obligatoire)

Nous allons réserver les adresses entre **10.3.91.10** et **10.3.91.50** pour les IP fixes (les machines connues qui doivent conserver la même adresse IP) et les adresses **10.3.91.100** à **10.3.91.199** pour les stations non référencées (elles auront donc des adresses variables).

2.2 Configuration

Pour la mise au point, il est possible soit de lancer **dnsmasq** depuis un terminal, soit de suivre le fichier de log (**tail -f /var/log/syslog**). Nous allons modifier le fichier de configuration **/etc/dnsmasq.conf**.

Il est possible de positionner son nom de domaine en insérant la directive **domain=mon.domaine.tld**. Pour valider la connexion, nous définissons l'ensemble des adresses :

```
dhcp-range=10.3.91.100,10.3.91.199,255.255.255.0,12H
```

Il y a quatre paramètres : la première et dernière adresses de la plage, le masque réseau et la durée de conservation de la même adresse.

Relancez le serveur : **service dnsmasq restart**, puis tentative de connexion. Le syslog doit montrer les requêtes dns et l'envoi d'une adresse pour la station. En vérifiant les paramètres IP sur la station, il est facile de valider la connexion. Comme le routage n'est pas activé, les connexions vers l'Internet vont toutes échouer.

Nous allons maintenant spécifier une adresse fixe pour une station. Pour cela, il faut ajouter dans le fichier de configuration :

```
dhcp-host=e0:c8:f2:45:24:45,10.3.91.10
```

L'adresse mac peut être récupérée en consultant la table arp du serveur **/usr/sbin/arp -na**.

En rajoutant des entrées dans le fichier **/etc/hosts**, il est possible d'associer un nom, éventuellement avec son domaine à une adresse IP. On peut alors se connecter à **monmediacenter.mamaison.amoi** (même si **mamaison.amoi** n'est défini que localement).

Nous avons donc vu comment déclarer un ensemble d'adresses, spécifier des adresses fixes pour les stations déclarées et ajouter des enregistrements DNS. Pour une TPE ou une utilisation domestique, c'est suffisant. Il reste à fournir un accès complet au réseau.

3 Routage et accès réseau

Dans le cas général, il faut autoriser le routage et activer les translations d'adresses. Pour activer le routage une fois, il suffit d'indiquer au noyau que le routage doit être activé :

```
# echo 1 > /proc/sys/net/ipv4/ip_forward
```

Ceci ne fonctionnera que jusqu'au prochain reboot. Pour le rendre permanent, il faut changer le fichier de configuration **/etc/sysctl.conf** pour dé-commenter la ligne autorisant le routage :

```
# Uncomment the next line to enable packet forwarding for IPv4
net.ipv4.ip_forward=1
```

Maintenant le routage fonctionne, mais notre box ne sait probablement pas que le réseau 10.3.91 se trouve derrière notre routeur. Il faut donc activer la translation d'adresses. Si notre routeur est connecté à la box par l'interface eth0, alors il faut entrer la commande suivante :

```
# /sbin/iptables -t nat -A POSTROUTING -o eth0 -j MASQUERADE
```

À ce moment, la station peut accéder à Internet. Sinon, il faut lancer tshark sur toutes les interfaces de notre routeur pour comprendre ce qui se passe. Bien sûr, pour rendre permanente la règle iptables, on peut modifier **/etc/network/interfaces** pour ajouter une commande quand la carte WiFi s'active :

```
allow-hotplug wlan0
iface wlan0 inet static
  address 10.3.91.1
  netmask 255.255.255.0
  broadcast 10.3.91.255

  post-up /sbin/iptables -t nat -A POSTROUTING -o eth0 -j
MASQUERADE
```

Un petit reboot pour vérifier que tout se passe bien en cas de coupure électrique, et c'est fini !

Conclusion

Nous avons montré comment facilement mettre en place un point d'accès WiFi pour un usage analogue aux box. Cette utilisation peut permettre de rajouter un point d'accès WiFi dans une autre partie de la maison si les murs ou la distance perturbent trop le signal.

Cela fournit aussi la possibilité de voir quelle machine souhaite communiquer avec quel serveur et identifier des connexions non souhaitées. **tshark** peut tourner pendant des semaines pour capturer tout le trafic réseau et donner la possibilité de découvrir des connexions rares.

Cela permet aussi d'isoler mon réseau WiFi avec mes machines GNU/Linux sécurisées et laisser les enfants jouer avec leurs terminaux. Je peux aussi établir sur mon routeur un contrôle parental avec extinction des feux à l'heure prévue, ne leur laisser un accès que quelques heures par jour, bloquer le trafic vers les sites d'espionnages...

Un outil complémentaire est **arpwatch** qui me permet de lister les machines qui se sont connectées dans le passé. C'est particulièrement utile si toutes les stations sont référencées. Sinon, à chaque fois qu'une station change d'adresse IP arpwatch génèrera une erreur.

Par contre, certains jeux risquent de ne plus fonctionner dans ce cadre. Le protocole upnp **[5]** permet à une station de modifier le comportement de la box pour autoriser des connexions entrantes vers cette station. La sécurité de ce genre de comportement est pour le moins suspecte ! ■

Références

[1] BODOR D., « Transcend WIFI SD... », *Open Silicium* n°10, mai 2014, p. 56 à 63.

[2] KERMA G., « Comment perdre la garantie de son NAS», *GNU/Linux Magazine* n°156, janvier 2013, p. 70

[3] Hackers use refrigerator in cyber attack : http://www.foxnews.com/tech/2014/01/20/hackers-use-refrigerator-in-cyber-attack/

[4] Le site de l'Internet Systems Consortium (serveurs dhcp et dns) : https ://www.isc.org/

[5] Le protocole upnp : https://fr.wikipedia.org/wiki/Universal_Plug_an_Play

DATATABLES : INTERAGIR AVEC LES TABLEAUX HTML

Mariana ANDUJAR et Magali CONTENSIN [Ingénieurs en informatique au CNRS]

Le plugin DataTables de jQuery permet d'interagir facilement avec un tableau HTML. Il gère la personnalisation du style, la pagination, le filtrage des données et le tri sur une ou plusieurs colonnes.

Tableau HTML Tri Pagination Query Plugin Datatables Style

L'OBJECTIF

Cet article apprend à installer et utiliser le plugin **DataTables**, qui permet à l'internaute d'interagir avec un tableau HTML. L'objectif est d'afficher un tableau paginé, trié sur plusieurs colonnes, au style personnalisé, avec des éléments de contrôle en français.

LES OUTILS

Deux systèmes :

→ Un navigateur web ;

→ Un éditeur de texte.

PHASE 1

Installation

Pour tester le *plugin*, il n'est pas nécessaire d'utiliser un serveur Web. Créez un répertoire **projet** et téléchargez la dernière version du *plugin* DataTables sur https://www.datatables.net/download/packages. La version utilisée dans cet article est la 1.10.9. Décompressez l'archive téléchargée, renommez-la en **DataTables**, et placez-la dans le répertoire **projet**. L'archive contient le *plugin* ainsi que **jQuery** dans le sous-répertoire **media** (voir figure 1). Créez ensuite un fichier **tableau.html** qui contiendra le tableau sur lequel le *plugin* DataTables sera appliqué, ainsi qu'un fichier **tableau.js** qui contiendra le code JavaScript.

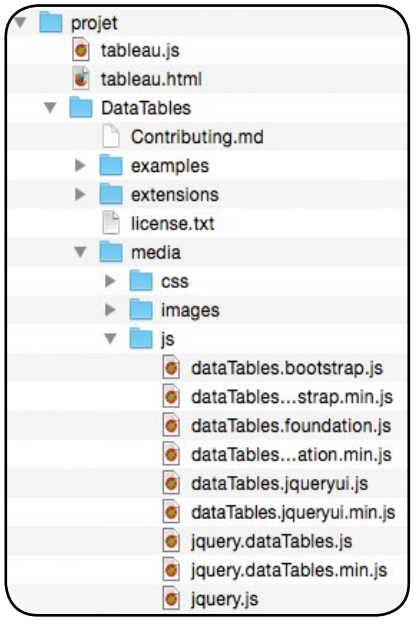


Fig. 1 : Arborescence du projet.

Pour utiliser DataTables, vous devez tout d'abord inclure, dans le fichier **tableau.html**, les bibliothèques jQuery et DataTables dans l'en-tête de la page html (élément **head**), ainsi que le fichier de style CSS de DataTables. Le code ci-après montre les inclusions :

```
<!doctype html>
<html>
<!-- fichier tableau.html -->
<head>
    <meta charset="utf-8">
    <title>Liste fromages</title>
    <script type="text/javascript"
src='DataTables/media/js/jquery.js'></
script>
    <script type="text/javascript"
src="DataTables/media/js/jquery.
dataTables.min.js"></script>
    <script type="text/javascript"
src="tableau.js"></script>
    <link rel="stylesheet" type="text/
css" href="DataTables/media/css/jquery.
dataTables.min.css">
</head>
```

Le tableau html doit contenir l'élément **thead,** qui définit les titres des colonnes, et l'élément **tbody,** comportant les données. Le code ci-après montre le corps du fichier HTML :

```
<body>
    <table id='tab' class='display'>
        <caption>Fromages</caption>
        <thead>
            <tr><th>Nom</th><th>Lait</th><th>Prix</th></tr>
        </thead>
        <tbody>
            <tr><td>Roquefort</td><td>brebis</td><td>4</td></tr>
            <tr><td>Morbier</td><td>vache</td><td>1</td></tr>
            <tr><td>Raclette</td><td>vache</td><td>3</td></tr>
            ...
            <tr><td>St Nectaire</td><td>vache</td><td>2</td></tr>
        </tbody>
    </table>
</body>
</html>
```

Dans le fichier JavaScript **tableau.js**, associez DataTables au tableau dont l'attribut **id** a la valeur **tab** :

```
// fichier tableau.js
$(document).ready(function () {
    $('#tab').DataTable();
});
```

La figure 2 montre le résultat obtenu lorsque le fichier **tableau.html** est chargé dans le navigateur.

Par défaut, DataTables trie le tableau par ordre croissant sur la première colonne (comme sur la figure 2). L'utilisateur peut ensuite trier le tableau, par ordre croissant ou décroissant, sur n'importe quelle colonne. Le tableau de la figure 3 est par exemple trié par ordre de nom décroissant.

Show 10 entries Search:

Fromages

Nom	Lait	Prix
Raclette	vache	3
Reblochon	vache	7
Robiola	vache	6
Roquefort	brebis	4
St Nectaire	vache	2

Showing 21 to 25 of 25 entries Previous 1 2 3 Next

Fig. 2 : Tableau avec les éléments de contrôle par défaut.

Afficher 10 éléments Rechercher :

Fromages

Nom	Lait	Prix
Brillat Savarin	vache	7
Brie	vache	4
Brebiou	brebis	3
Boursin	vache	6
Banon	chèvre	5

Affichage de l'élément 21 à 25 sur 25 éléments Précédent 1 2 3 Suivant

Fig. 3 : Tableau trié par ordre décroissant sur la 1ère colonne, avec les éléments de contrôle en français.

PHASE 2

Langue

DataTables affiche le texte des éléments de contrôle en anglais (figure 2), mais il est possible de le configurer pour utiliser une autre langue, par exemple le français (figure 3).

Pour cela, téléchargez le fichier de la langue souhaitée : https://www.datatables.net/plug-ins/i18n.

Dans cet article, le fichier téléchargé est celui de la langue française, obtenu grâce au CDN de DataTables : http://cdn.datatables.net/plug-ins/1.10.9/i18n/French.json.

Placez le fichier **French.json** dans **DataTables/media**, et affectez le chemin de ce fichier à la propriété **url** de la propriété **language** de DataTables dans le fichier JavaScript :

```
$(document).ready(function () {
    $('#tab').DataTable({
        language: {
            url: "DataTables/media/French.
json"
        }
    });
});
```

PHASE 3

Éléments de contrôle

L'option **dom** de DataTables permet de définir les éléments de contrôle du tableau qui seront affichés ainsi que leur ordre dans la page. Le tableau ci-après présente les différents contrôles disponibles pour cette option.

Par défaut la valeur de **dom** est **lftrip** (voir figure 4). Il est possible d'utiliser plusieurs fois la même lettre pour dupliquer un contrôle.

Le code ci-après affiche la pagination au-dessus du tableau (voir figure 5) :

Contrôle	Description
l	Affiche le *select* indiquant le nombre de lignes à afficher par page
f	Affiche le champ permettant de filtrer les données du tableau
i	Affiche des informations
p	Affiche le menu de pagination
t	Indique l'emplacement du tableau par rapport aux contrôles
r	Affiche un message de traitement

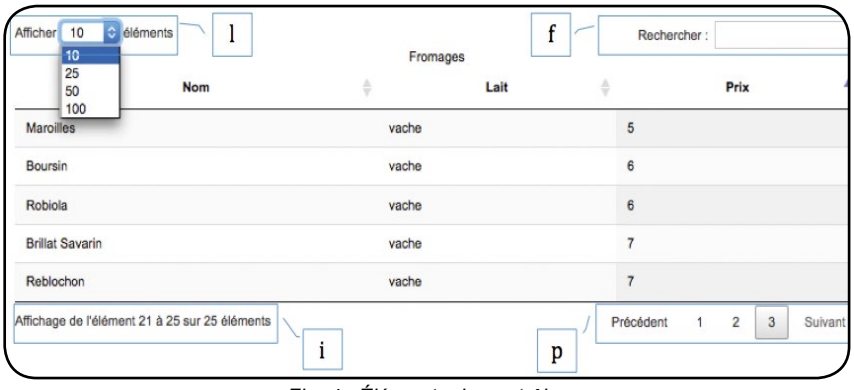


Fig. 4 : Éléments de contrôle.

```
$(document).ready(function () {
    $('#tab').DataTable({
        dom: 'ptlf',
        language: {
            url: "DataTables/media/French.json"
        }
    });
});
```

Le champ ***rechercher*** restreint l'affichage des lignes du tableau à celles qui contiennent la chaîne saisie dans ce champ. Le filtrage n'est pas sensible à la différence majuscules/minuscules.

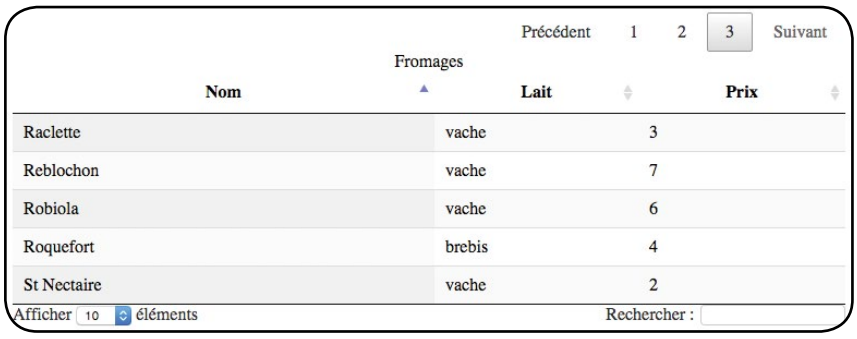


Fig. 5 : Modification de l'emplacement des éléments de contrôle.

PHASE 4

Pagination

Il est possible de personnaliser la pagination avec l'option **`pagingType`**. Quatre types de pagination sont disponibles : **`simple`**, **`simple_numbers`**, **`full`**, **`full_numbers`** (voir figure 6).

Le menu qui définit le nombre de lignes affichées par page est configuré avec l'option **`lengthMenu`**. Par défaut, DataTables propose un menu comportant : **`10`**, **`25`**, **`50`**, **`100`** (voir figure 4). Le nombre de lignes à afficher au chargement du tableau est contrôlé par l'option **`pageLength`**.

Le code ci-après, affiche 3 lignes par page, et permet de basculer l'affichage à **`5`**, **`10`**, **`15`**, **`20`** ou **`25`** lignes par page (voir figure 7).

```
pagingType: "simple_numbers",
lengthMenu:[5,10,15,20,25],
pageLength: 3
```

PHASE 5

Tri

Par défaut, le tri est réalisé sur la première colonne, par ordre croissant (voir figure 2). La propriété **`order`** indique les colonnes à utiliser pour le tri et l'ordre à leur appliquer : croissant

Fig. 6 : Différents types de pagination.

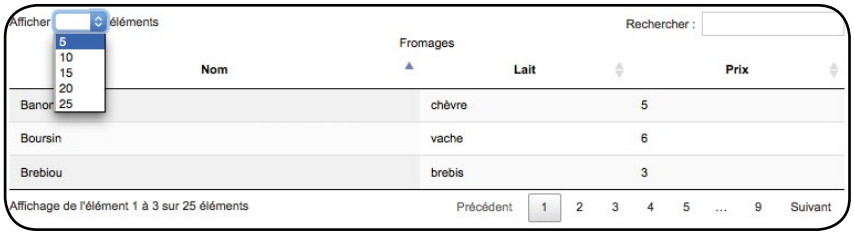


Fig. 7 : Personnalisation de la pagination.

(**asc**) ou décroissant (**desc**). Dans le code ci-après, le tri est réalisé sur la deuxième colonne par ordre décroissant, puis sur la première colonne par ordre croissant (voir figure 8a).

```
order:[[1,'desc'], [0, 'asc']]
```

L'option **columns**, permet de définir le type de la colonne (alphabétique, numérique, …). L'exemple ci-après, indique que les deux premières colonnes sont de type texte et que la troisième est de type numérique. Le type de la colonne sera pris en compte pour le tri.

```
columns: [
        {type:"text"},
        {type:"text"},
        {type:"num"}
]
```

Une cellule peut contenir des éléments html comme dans l'exemple de code ci-après :

```
<td><em>buffle</em></td>
```

Alors, au moment du tri, les balises **** seront considérées comme du texte, le tri sera différent de celui attendu par l'utilisateur. Dans la figure 8b, page suivante, **buffle** est situé sur la dernière ligne du tableau, lorsque le tri est réalisé par ordre décroissant, car le caractère **<** a une valeur inférieure aux caractères de l'alphabet.

Pour ignorer les éléments HTML et obtenir le tri souhaité lors de la comparaison (voir figure 8a, page suivante), il faut donner à la colonne le type **html** :

```
columns: [
        {type:"text"},
        {type:"html"},
        {type:"num"}
]
```

Il est possible d'exclure une ou plusieurs colonnes du tri avec la propriété **orderable :**

```
columns: [
        {type:"text"},
        {type:"html"},
        {orderable:false}
]
```

PHASE 6
Style

Le style du tableau est contrôlé par une ou plusieurs classes de style prédéfinies (voir tableau ci-après), placées dans l'attribut **class** de l'élément **table** :

Classe de style	Description
cell-border	Encadre les cellules
row-border	Encadre les lignes
stripe	Alterne les couleurs de fond des lignes
hover	Change la couleur de fond de la ligne survolée par la souris
order-column	Change la couleur de fond des colonnes utilisées pour trier le tableau
display	Raccourci pour **stripe**, **hover**, **row-border**, **order-column**
nowrap	Désactive le passage à la ligne dans une cellule, le texte de la cellule est placé sur une seule ligne
compact	Diminue les espaces dans les cellules

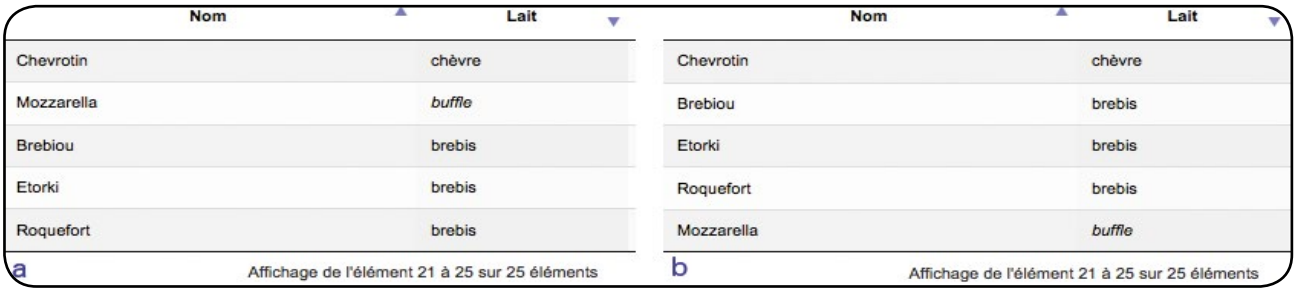


Fig. 8 : Option de tri

```
<table id='tab' class='display compact
cell-border'>
```

DataTables supporte les thèmes **bootstrap** et **jQuery UI**, et il est aussi possible de personnaliser le thème par défaut avec l'interface de création de thème (voir figure 9) : https://www.datatables.net/manual/styling/theme-creator. Une fois le style défini, cliquez sur le bouton ***create stylesheet***, puis copiez le code qui apparaît dans l'onglet CSS, et placez-le dans un fichier **DataTables/media/css/datatables.css**. Remplacez le chemin de la feuille de style dans l'élément **link** du fichier **tableau.html** :

```
<link rel="stylesheet" type="text/css"
href="DataTables/media/css/datatables.css">
```

LE RÉSULTAT

La figure 10 montre le résultat obtenu une fois toutes les étapes réalisées.

```
$(document).ready(function () {
    $('#tab').DataTable({
        language: {
            url: "DataTables/media/French.
json"
        },
        dom: "tip",
        pagingType: "simple",
        pageLength: 8,
        order: [[1, 'desc'], [0, 'asc']],
        columns: [
            {type: "text"},
            {type: "html"},
            {orderable: false}
        ]
    });
});
```

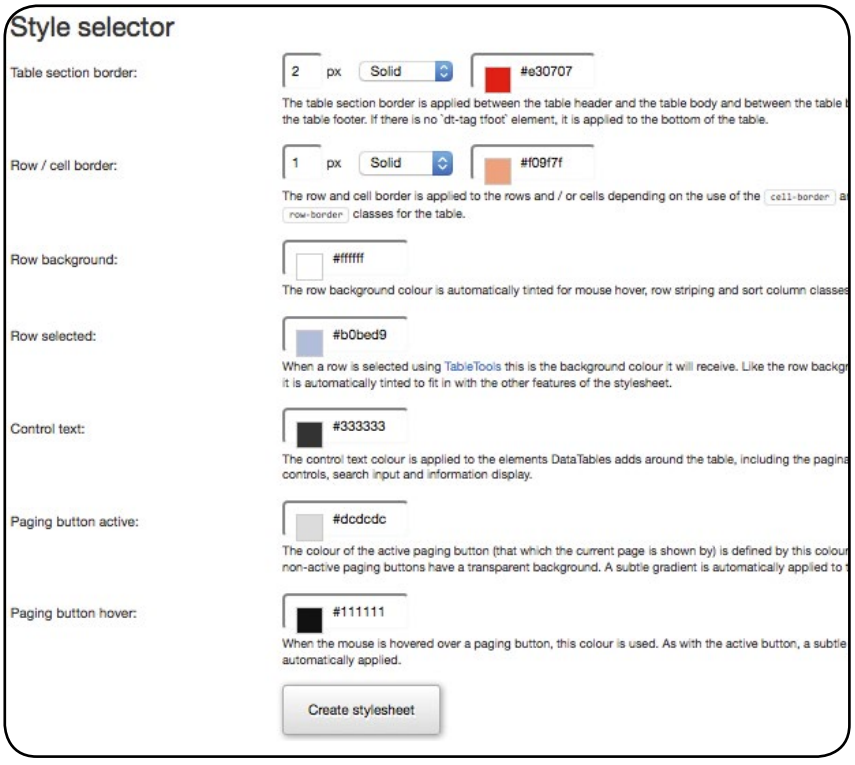


Fig. 9 : Interface de création de thème.

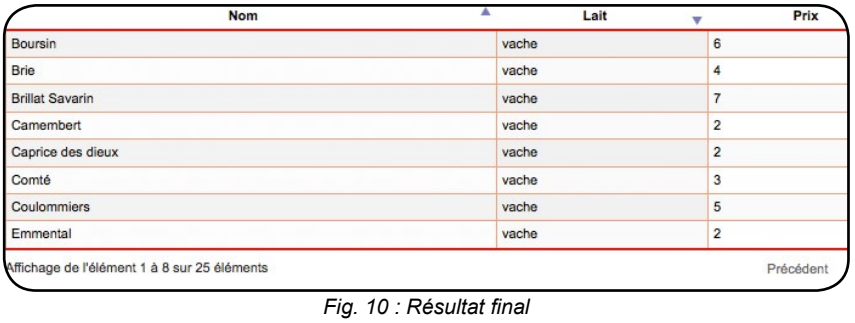


Fig. 10 : Résultat final

Le tableau a un style **display**, **compact** et **cell-border.** Les couleurs ont été personnalisées avec le créateur de thème. Les éléments de contrôle présents sont les informations et la pagination de type **simple**, et ils sont traduits en français. La dernière colonne ne peut pas être triée. ■

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

LINUX MAGAZINE / FRANCE

PROFESSIONNELS !

DÉCOUVREZ NOS OFFRES D'ABONNEMENTS ...

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

www.ed-diamond.com

PDF COLLECTIFS PRO

Prix TTC en Euros / France Métropolitaine

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

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

ACCÈS COLLECTIFS BASE DOCUMENTAIRE PRO

Prix TTC en Euros / France Métropolitaine

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

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

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

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

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

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

LINUX MAGAZINE / FRANCE

Édité par 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 :

.....

.....

GÉRER LES EXPRESSIONS RÉGULIÈRES EN LANGAGE C

Martial BORNET [Consultant système, auteur du livre « Expressions Régulières, syntaxe et mise en œuvre » chez ENI, créateur de la commande de colorisation « hl »]

Cet article va vous présenter la gestion des expressions régulières depuis un programme C. À partir d'exemples simples, vous découvrirez les mécanismes qui vous permettront d'écrire vos propres programmes C utilisant des expressions régulières.

Langage C
Expressions Régulières
regexec()
regcomp()
regerror()

L'OBJECTIF

L'objectif de cet article est de présenter la manipulation des expressions régulières à partir d'un programme écrit en langage C. Les expressions régulières, *basiques* et *étendues*, sont supposées connues. Nous ne verrons dans cet article que les mécanismes permettant de les manipuler à partir d'un programme C. Si vous souhaitez des informations sur leurs syntaxes, veuillez vous reporter à la page de manuel **regex(7)**, accessible à l'aide de la commande :

```
$ man 7 regex
```

LES OUTILS

L'outil principal dont nous allons avoir besoin est un compilateur C pouvant être exécuté sur un système d'exploitation GNU/Linux fonctionnel, et bien sûr de quoi saisir un programme source C, soit :

- un système GNU/Linux (Devuan, Debian, Red Hat, ou un dérivé) ;
- **gcc** (le compilateur C : paquet **gcc**) ;
- **vim** (paquet **vim**) ou tout autre éditeur de codes (**vi**, **ex**, **ed**, ou même **cat** pour les gurus ;-).

PHASE 1 Test de correspondance entre une chaîne de caractères et une ER basique

Nous allons commencer par un programme simple, capable d'effectuer un test de correspondance entre une chaîne de caractères et une *expression régulière basique* (nous verrons dans la suite comment manipuler les *expressions régulières étendues*).

Dans un premier temps, nous allons écrire un programme C uniquement capable de récupérer à partir de la ligne de commande les deux arguments cités (la chaîne de caractères et l'expression régulière). L'expression régulière sera le premier argument, et la chaîne à tester sera le deuxième argument. Nous allons afficher les arguments tels qu'ils auront

été reçus par le programme, ceci dans le but de valider les paramètres et nous éviter d'éventuelles pertes de temps en mise au point et en recherche de bug.

Le squelette du programme (re_disp.c) sera donc le suivant :

```
#include  <stdio.h>
#include  <stdlib.h>

int main(int argc, char *argv[])
{
     char          *_regexp, *_string;

     if (argc != 3) {
          fprintf(stderr, "Usage: %s expression_reguliere
chaine\n",
                  argv[0]);
          exit(1);
     }
     _regexp       = argv[1];
     _string       = argv[2];

     printf("Expression Reguliere = [%s]\n", _regexp);
     printf("Chaine               = [%s]\n", _string);

     return 0;
}
```

Le programme exécutable est obtenu à l'aide de la commande suivante faisant appel au compilateur **gcc** :

```
$ gcc -o re_disp re_disp.c
```

Le binaire généré peut être appelé de la façon suivante :

```
$ ./re_disp '[a-z][a-z0-9]*' 'azerty98'
```

Le résultat sera le suivant :

```
Expression Reguliere = [[a-z][a-z0-9]*]
Chaine               = [azerty98]
```

Bien, maintenant que le squelette est opérationnel, passons à l'étape suivante : la recherche de correspondance.

PHASE 2 Recherche de correspondance

Il s'agit maintenant d'exploiter les fonctions **regcomp()** et **regexec()** disponibles en bibliothèque pour pouvoir effectuer ce qui nous intéresse, à savoir le test de correspondance entre la chaîne de caractères et l'expression régulière.

Ce test nécessite deux étapes :

1. La compilation de l'expression régulière par la fonction **regcomp()**, permettant d'obtenir une expression régulière compilée, directement exploitable par la fonction **regexec()**.
2. La comparaison au moyen de la fonction **regexec()** de la chaîne et de l'expression régulière compilée précédemment. Si la fonction **regexec()** trouve une correspondance entre la chaîne et l'expression régulière, son code retour sera **0**. Dans le cas contraire, son code retour sera **REG_NOMATCH**.

Comme c'est souvent le cas, l'utilisation de fonctions en bibliothèque nécessite l'inclusion, dans le programme source, de fichiers *headers* spécifiques contenant des définitions de macros, de structures et de prototypes. Ici, il s'agit de **<sys/types.h>** et **<regex.h>**.

Le programme ressemblera à ceci (**re_basic.c**) :

```
#include  <stdio.h>
#include  <stdlib.h>
#include  <sys/types.h>
#include  <regex.h>

int main(int argc, char *argv[])
{
     char          *_regexp, *_string;
     regex_t        _reg;
     int            _cflags, _eflags;

     if (argc != 3) {
          fprintf(stderr, "Usage: %s expression_reguliere
chaine\n",
                  argv[0]);
          exit(1);
     }
     _regexp       = argv[1];
     _string       = argv[2];

     printf("Expression Reguliere = [%s]\n", _regexp);
     printf("Chaine               = [%s]\n", _string);

     _cflags       = 0;
     if (regcomp(&_reg, _regexp, _cflags) != 0) {
          fprintf(stderr, "%s: erreur de compilation de
l'expression reguliere !\n",
                  argv[0]);
          exit(2);
     }

     _eflags       = 0;
     if (regexec(&_reg, _string, 0, 0, _eflags) == 0) {
          printf("Correspondance entre \"%s\" et \"%s\"\n",
_string, _regexp);
     }
     else {
          printf("Pas de correspondance entre \"%s\" et \"%s\"
!\n", _string, _regexp);
     }

     return 0;
}
```

Ce programme affiche les paramètres passés en ligne de commande, puis effectue la compilation de l'expression régulière et la recherche de correspondance entre la chaîne et l'expression régulière.

L'instruction **regcomp(&_reg, _regexp, _cflags)** compile l'expression régulière, reçue par le programme sous forme de chaîne de caractères ASCII (texte), et en produit une version directement exploitable par la fonction **regexec()**. Cette dernière effectue ensuite une recherche de correspondance entre la chaîne de caractères et l'expression régulière (dans sa version compilée).

L'initialisation de la variable **_cflags** à la valeur **0** (c'est-à-dire sans le flag **REG_EXTENDED**) a pour effet d'indiquer à la fonction **regcomp()** que l'expression régulière est une *expression régulière basique*.

L'exécutable est généré avec la commande suivante :

```
$ gcc -o re_basic re_basic.c
```

Testons par exemple avec les paramètres de la phase précédente :

```
$ ./re_basic '[a-z][a-z0-9]*' 'azerty98'
```

Le résultat sera :

```
Expression Reguliere = [[a-z][a-z0-9]*]
Chaine               = [azerty98]
Correspondance entre "azerty98" et "[a-z][a-z0-9]*"
```

Le test de correspondance est positif car la chaîne **azerty98** contient bien une chaîne commençant par un caractère alphabétique (**[a-z]**) suivi d'une suite éventuellement vide de caractères alphanumériques (**[a-z0-9]***).

On peut maintenant souhaiter savoir quelle correspondance exacte le programme a trouvée dans la chaîne passée. C'est ce que nous allons voir dans la phase suivante.

PHASE 3 Recherche de la position d'une sous-chaîne correspondant à une ER

Nous sommes donc capables de déterminer s'il existe une correspondance entre une chaîne de caractères et une expression régulière : c'est bien, mais il pourrait également être utile de déterminer quelle est la position de la chaîne qui correspond à l'expression régulière spécifiée. C'est heureusement possible grâce à la puissance des mêmes fonctions bibliothèque **regcomp()** et **regexec()**. Il suffit de demander lors de la compilation de pouvoir récupérer les informations relatives aux sous-chaînes.

Cela est fait par défaut si l'on ne passe pas la valeur **REG_NOSUB** dans le paramètre d'appel **cflags** de la fonction **regcomp()**. Les valeurs relatives aux sous-chaînes trouvées sont stockées dans le tableau de structures **pmatch** sous forme d'indices de la chaîne de caractères principale, chaque sous-chaîne étant repérée par un indice de début et un indice de fin, stockés respectivement dans les membres **rm_so** et **rm_eo** de la structure **pmatch**.

Remarque : afin de ne pas surcharger le code des programmes qui suivent, on sortira de la boucle d'affichage des correspondances avec les sous-expressions dès que l'une d'entre elles n'a pas de correspondance. Il ne faut cependant pas considérer que la rencontre d'un membre **rm_so** valant **-1** signifie la fin des correspondances.

Voici le programme affichant les sous-chaînes (**re_basic_sub.c**) :

```
#include  <stdio.h>
#include  <stdlib.h>
#include  <sys/types.h>
#include  <regex.h>

#define   SIZE          (1024)

int main(int argc, char *argv[])
{
     char          *_regexp, *_string;
     regex_t        _reg;
     int            _cflags, _eflags, _i, _s, _e;
     size_t         _nmatch;
     regmatch_t     _pmatch[SIZE];

     if (argc != 3) {
          fprintf(stderr, "Usage: %s expression_reguliere
chaine\n",
                  argv[0]);
          exit(1);
     }
     _regexp       = argv[1];
     _string       = argv[2];

     printf("Expression Reguliere = [%s]\n", _regexp);
     printf("Chaine               = [%s]\n", _string);

     _cflags       = 0;
     if (regcomp(&_reg, _regexp, _cflags) != 0) {
          fprintf(stderr, "%s: erreur de compilation de
l'expression reguliere !\n",
                  argv[0]);
          exit(2);
     }
```

```
    _eflags       = 0;
    _nmatch       = sizeof(_pmatch) / sizeof(_pmatch[0]);
    if (regexec(&_reg, _string, _nmatch, _pmatch, _eflags) ==
0) {
         printf("Correspondance entre \"%s\" et \"%s\"\n",
_string, _regexp);

         for (_i = 0; _pmatch[_i].rm_so != -1; _i++) {
              _s   = _pmatch[_i].rm_so;
              _e   = _pmatch[_i].rm_eo - 1;

              printf("Correspondance %2d : debut = %2d, fin =
%2d, valeur = [%.*s]\n",
                     _i, _s, _e, _e - _s + 1, _string + _s);
         }
    }
    else {
         printf("Pas de correspondance entre \"%s\" et \"%s\"
!\n", _string, _regexp);
    }

    return 0;
}
```

Voici un exemple de récupération de sous-chaînes avec le programme précédent :

```
$ ./re_basic_sub '\([a-z]\)\([a-zA-Z]*\)\([0-9]*\)' 'azerty98'
```

Le résultat est le suivant :

```
Expression Reguliere = [\([a-z]\)\([a-zA-Z]*\)\([0-9]*\)]
Chaine               = [azerty98]
Correspondance entre "azerty98" et "\([a-z]\)\([a-zA-Z]*\)\([0-
9]*\)"
Correspondance  0 : debut =  0, fin =  7, valeur = [azerty98]
Correspondance  1 : debut =  0, fin =  0, valeur = [a]
Correspondance  2 : debut =  1, fin =  5, valeur = [zerty]
Correspondance  3 : debut =  6, fin =  7, valeur = [98]
```

PHASE 4 Et les ER étendues ?

Les *expressions régulières basiques* sont puissantes, mais les *expressions régulières étendues* le sont plus encore car elles permettent l'utilisation d'un OU logique et l'utilisation de facteurs de répétition autorisant une syntaxe condensée.

Elles permettent de spécifier des alternatives (avec **|**), des expressions optionnelles (avec **?**), des facteurs de répétition (avec **{m,n}**), et le groupement d'expressions avec des parenthèses, comme dans :

```
([a-zA-Z_][a-zA-Z_0-9]+|[-+]?[0-9]+)
```

Pour les utiliser, nous devons le spécifier lors de la compilation de l'expression régulière en positionnant le *flag* **REG_EXTENDED** dans le paramètre d'appel **cflags** de la fonction **regcomp()**, ce qui peut être fait de la façon suivante :

```
_cflags = REG_EXTENDED;
 if (regexec(&_reg, _string, _nmatch, _pmatch, _eflags) == 0) {
```

Voici le programme gérant les expressions régulières étendues (**re_extended_sub.c**) :

```
#include  <stdio.h>
#include  <stdlib.h>
#include  <sys/types.h>
#include  <regex.h>

#define   SIZE          (1024)

int main(int argc, char *argv[])
{
     char           *_regexp, *_string;
     regex_t         _reg;
     int             _cflags, _eflags, _i, _s, _e;
     size_t          _nmatch;
     regmatch_t      _pmatch[SIZE];

     if (argc != 3) {
          fprintf(stderr, "Usage: %s expression_reguliere
chaine\n",
                  argv[0]);
          exit(1);
     }
     _regexp        = argv[1];
     _string        = argv[2];

     printf("Expression Reguliere = [%s]\n", _regexp);
     printf("Chaine               = [%s]\n", _string);

     _cflags        = REG_EXTENDED;
     if (regcomp(&_reg, _regexp, _cflags) != 0) {
          fprintf(stderr, "%s: erreur de compilation de
l'expression reguliere !\n",
                  argv[0]);
          exit(2);
     }

     _eflags        = 0;
     _nmatch        = sizeof(_pmatch) / sizeof(_pmatch[0]);
     if (regexec(&_reg, _string, _nmatch, _pmatch, _eflags) ==
0) {
          printf("Correspondance entre \"%s\" et \"%s\"\n",
_string, _regexp);

          for (_i = 0; _pmatch[_i].rm_so != -1; _i++) {
               _s   = _pmatch[_i].rm_so;
               _e   = _pmatch[_i].rm_eo - 1;

               printf("Correspondance %2d : debut = %2d, fin =
%2d, valeur = [%.*s]\n",
                      _i, _s, _e, _e - _s + 1, _string + _s);
          }
```

```
    }
    else {
        printf("Pas de correspondance entre \"%s\" et \"%s\"
!\n", _string, _regexp);
    }

    return 0;
}
```

PHASE 5 Les flags modifiant le comportement de l'interprétation de l'ER

En plus du choix du type d'expression régulière (*basique* ou *étendue*), il est possible d'utiliser d'autres *flags* pour modifier le comportement de l'interprétation de l'expression régulière. Ces *flags* sont les suivants:

- sont traités lors de la compilation par la fonction **regcomp()** (ces *flags* influent sur la compilation de l'expression régulière, et ne sont pas désactivables une fois l'expression compilée) :
- **REG_ICASE** : (*Ignore case*) demande de ne pas distinguer les majuscules des minuscules ;
- **REG_NEWLINE** : demande que le caractère spécifiant un début de ligne (**^**) corresponde à la chaîne vide suivant immédiatement un caractère *newline*, et que le caractère spécifiant une fin de ligne (**$**) corresponde à la chaîne vide précédant immédiatement un caractère *newline*. Une expression désignant un caractère quelconque ne correspond pas au caractère *newline*. Une expression désignant tous les caractères ne correspondant pas à une liste qui ne contient pas un *newline* ne correspond pas au caractère *newline* ;
- **REG_NOSUB** : (*No sub-string*) demande de ne pas faire de traitement spécifique pour mémoriser les informations relatives aux sous-chaînes.
- sont traités lors du test de comparaison par la fonction **regexec()** (ces *flags* peuvent ou non être passés à **regexec()** pour une même expression régulière compilée) :
- **REG_NOTBOL** : indique que le caractère désignant un début de ligne (**^**) ne correspondra jamais dans la chaîne passée, ce qui permet de faire des recherches de correspondances répétées entre l'expression régulière et des fragments de la chaîne initiale, en considérant que le début de la chaîne ne doit pas être interprété comme un début de ligne. Voir par exemple les sources de la commande de colorisation **hl** disponible sur github : **https://github.com/mbornet-hl/hl/blob/master/src/cr_main.c**.
- **REG_NOTEOL** : indique que le caractère désignant une fin de ligne (**$**) ne correspondra jamais dans la chaîne passée.

PHASE 6 Traiter les erreurs et libérer la mémoire

Les étapes précédentes nous ont permis d'apprendre à manipuler les expressions régulières depuis un programme C, mais pour être complet il nous reste deux choses à voir : comment traiter les erreurs éventuelles, et comment libérer la mémoire devenue inutile.

Nous disposons pour cela des deux fonctions suivantes :

- **regerror()** ;
- **regfree()**.

Les prototypes de ces deux fonctions sont spécifiés dans la page de manuel **regex(3)**, accessible à l'aide de la commande :

Lors de la compilation d'une expression régulière, la fonction **regcomp()** peut retourner un code d'erreur numérique indiquant le type d'erreur rencontrée. Cet indicateur d'erreur doit être converti en message sous forme de texte, bien plus clair et facile à comprendre : c'est le rôle de la fonction **regerror()**. Celle-ci doit être appelée de la façon suivante :

```
regerror(_error, &_reg, _errbuf, sizeof(_errbuf));
```

Voici le programme **re_extended_sub_nl_err.c** intégrant l'appel à la fonction d'affichage en clair du code d'erreur de la fonction **regcomp()** :

```
#include  <stdio.h>
#include  <stdlib.h>
#include  <sys/types.h>
#include  <regex.h>

#define   SIZE          (1024)

int main(int argc, char *argv[])
{
    char          *_regexp, *_string, _errbuf[256];
    regex_t        _reg;
```

```
    int             _cflags, _eflags, _i, _s, _e, _error;
    size_t          _nmatch;
    regmatch_t      _pmatch[SIZE];

    if (argc != 3) {
        fprintf(stderr, "Usage: %s expression_reguliere chaine\n",
                argv[0]);
        exit(1);
    }
    _regexp         = argv[1];
    _string         = argv[2];

    printf("Expression Reguliere = [%s]\n", _regexp);
    printf("Chaine               = [%s]\n", _string);

    _cflags         = REG_EXTENDED | REG_NEWLINE;
    if ((_error = regcomp(&_reg, _regexp, _cflags)) != 0) {
        fprintf(stderr, "%s: erreur de compilation de l'expression reguliere !\n",
                argv[0]);
        (void) regerror(_error, &_reg, _errbuf, sizeof(_errbuf));
        fprintf(stderr, "%s: regcomp error for \"%s\" : %s\"\n",
                argv[0], _regexp, _errbuf);
        exit(2);
    }

    _eflags         = 0;
    _nmatch         = sizeof(_pmatch) / sizeof(_pmatch[0]);
    if (regexec(&_reg, _string, _nmatch, _pmatch, _eflags) == 0) {
        printf("Correspondance entre \"%s\" et \"%s\"\n", _string, _regexp);

        for (_i = 0; _pmatch[_i].rm_so != -1; _i++) {
            _s   = _pmatch[_i].rm_so;
            _e   = _pmatch[_i].rm_eo - 1;

            printf("Correspondance %2d : debut = %2d, fin = %2d, valeur = [%.*s]\n",
                  _i, _s, _e, _e - _s + 1, _string + _s);
        }
    }
    else {
        printf("Pas de correspondance entre \"%s\" et \"%s\" !\n", _string, _regexp);
    }

    return 0;
}
```

Voici un exemple d'erreur de syntaxe :

```
$ ./re_extended_sub_nl_err '*ab' 'abcd'
```

Le résultat d'exécution est le suivant :

```
Expression Reguliere = [*ab]
Chaine               = [abcd]
./re_extended_sub_nl_err: erreur de compilation de l'expression reguliere !
./re_extended_sub_nl_err: regcomp error for "*ab" : Invalid preceding regular expression"
```

Voici un autre exemple d'erreur de syntaxe :

```
$ ./re_extended_sub_nl_err 'a(b|c' 'abcd'
```

Le résultat est le suivant :

```
Expression Reguliere = [a(b|c]
Chaine               = [abcd]
./re_extended_sub_nl_err: erreur de
compilation de l'expression reguliere !
./re_extended_sub_nl_err: regcomp error
for "a(b|c" : Unmatched ( or \("
```

Et finalement, il ne nous reste plus qu'à libérer, avec **regfree()**, la mémoire allouée par **regcomp()** lorsque nous n'en avons plus besoin, de façon à ne pas consommer de mémoire inutilement. La zone mémoire à libérer est la version compilée de l'expression régulière, et sa libération est effectuée de la façon suivante :

```
regfree(&_reg);
```

LE RÉSULTAT

Vous disposez maintenant des connaissances vous permettant d'écrire des programmes C capables de manipuler des expressions régulières, ainsi que d'une petite collection de programmes de base vous permettant de faire des tests élémentaires. Laissez maintenant votre imagination faire le reste, et écrivez les programmes qui vous seront utiles, ou qui seront utiles aux autres.

Si vous souhaitez continuer un peu, et examiner un exemple concret d'utilisation de ces fonctions, vous pouvez vous reporter aux sources de la commande de colorisation **hl** disponible sur GitHub à l'URL suivante : **https://github.com/mbornet-hl/hl/tree/master/src**.

Et maintenant que vous savez gérer les expressions régulières en C, amusez-vous et soyez créatifs ! ■

LES ADDONS, C'EST ENCORE SA FORGE

Jérôme BATON [Développeur, arrière-*-arrière-petit-fils de maréchal ferrant]

Dans ce nouvel article, nous allons finir d'apprendre comment créer des addons pour Forge, le framework de création rapide d'applications Java EE édité par JBoss/Red Hat, et ainsi, l'étendre selon nos envies.Vous avez aimé la création de projet Java EE du premier article [1], alors vous allez aimer créer un addon pour manipuler des projets Android !

Mots-clés : Java, JBoss, Forge, Addon, Génération de code, Template, Android

Résumé

JBoss Forge est un générateur de code, initialement orienté Java EE. Dans cet article, nous allons voir comment l'étendre pour générer du code Android !

Pour suivre cet article, il est préférable de connaître l'utilisation de JBoss Forge, de **JBoss Developper Studio** (JDS) bref, d'avoir lu les articles précédents consacrés à Forge **[1][2]**.

Le but de cet article est de vous permettre de créer VOS propres outils pour VOS projets en abordant des points essentiels, mis en œuvre dans un contexte Android. Ces outils vous permettront d'automatiser des tâches répétitives dans vos propres projets, quels qu'ils soient... donc moins d'ennuis au quotidien. Ô joie !

1 Le mode « moins simple mais mieux »

De façon « inception-nelle », le point de départ est d'utiliser Forge pour générer l'*addon*. Nous allons donc créer une nouvelle classe.

Ouvrez la fenêtre Forge pour lancer la commande `Addon : New UI Command` (voir figure 1).

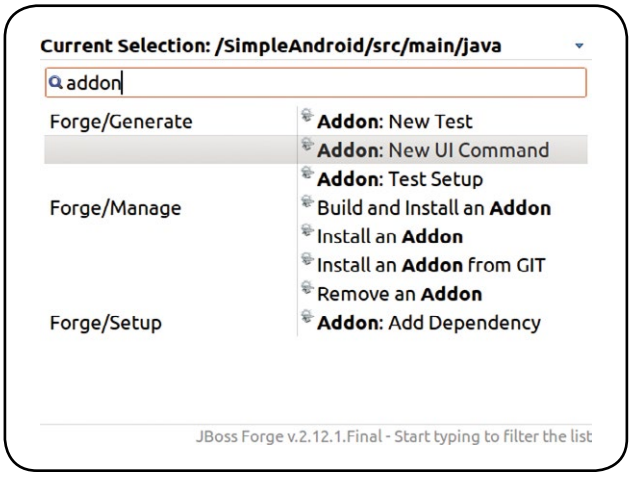


Fig. 1 : Fenêtre de commande Forge.

Une fenêtre telle que celle présentée en figure 2 est affichée

Le champ ***Type Name*** correspond au nom de la classe qui contiendra la commande, dont le nom est saisi dans le champ ***Command Name***.

À l'identique de la façon précédente, il est possible de saisir plusieurs catégories. Ici, la catégorie reste ***Android***.

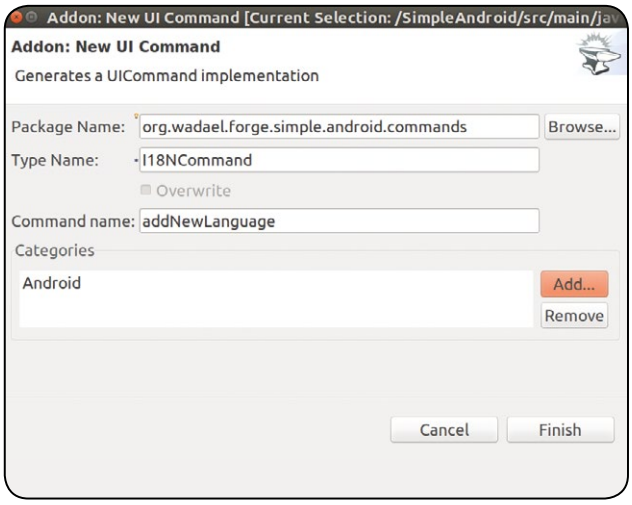


Fig. 2 : Interface de génération d'un addon.

À la validation, le code généré est le suivant :

```
package org.wadael.forge.simple.android.commands;

public class I18NCommand extends AbstractUICommand{

   @Override
   public UICommandMetadata getMetadata(UIContext context){
      return Metadata.forCommand(I18NCommand.class).
name("addNewLanguage")
            .category(Categories.create("Android"));
   }

   @Override
   public void initializeUI(UIBuilder builder) throws Exception{
   }

   @Override
   public Result execute(UIExecutionContext context) throws
Exception{
      return Results
            .success("Command 'addNewLanguage' successfully
executed!");
   }
}
```

Cela reste simple, il n'y a qu'une classe qui possède trois méthodes.

Note

On ne peut donc avoir qu'une seule commande par classe.

La méthode **`getMetaData`** indique au moteur de Forge (nommé **Furnace**) le nom de la commande et sa catégorie. Le corps de la méthode indique ce qui a été saisi au préalable. On peut y ajouter une description pour être plus complet (voir code ci-dessous).

La seconde méthode, **`initializeUI`**, a pour rôle de mettre en place les composants qui constitueront l'interface utilisateur de cet « écran ».

La troisième méthode correspond aux instructions qui seront exécutées lorsque les saisies seront effectuées et que l'utilisateur cliquera sur ***Finish***.

Une quatrième méthode, **`validate`**, est à ajouter afin de vérifier les saisies de l'utilisateur et renvoyer des messages d'erreur significatifs. Elle sera exécutée à chaque changement de valeur d'un champ.

Pour créer une commande qui dupliquera le fichier **`strings.xml`** pour créer un fichier pour une langue donnée (comme **`strings_fr`** pour le français), le code sera :

```
package org.wadael.forge.simple.android.commands;

public class I19NCommand extends AbstractUICommand {
  public static List<String> PAYS ;

  @Inject
  private UIInput<String> folder;

  @Inject
  @WithAttributes(label="language",required=true, description="New
language to support" )
  private UISelectOne<String> language ;

  public I19NCommand() {
    PAYS = new ArrayList<String>();
    PAYS.add("fr");
    PAYS.add("de");
    PAYS.add("ar");
    PAYS.add("dg");
  }

  @Override
  public UICommandMetadata getMetadata(UIContext context) {
    return Metadata.forCommand(I19NCommand.class).
name("addNewLanguage").category(Categories.create("Android")).
description("Adds a new supported language to the application.");
  }

  @Override
  public void initializeUI(UIBuilder builder) throws Exception {
    language.setValueChoices(PAYS);
    // Ci-dessous, ces lignes pourraient remplacer l'annotation @
WithAttributes
    // language.setRequired(true);
    // language.setDescription("New language to support");
    folder.getFacet(HintsFacet.class).setInputType(InputType.
DIRECTORY_PICKER);
    folder.setRequired(true);
    builder.add(folder);
    builder.add(language);
  }
```

```
  @Override
  public void validate(UIValidationContext validator) {
    super.validate(validator);
    // Rien d'autre à vérifier avec ces composants.
  }

  @Override
  public Result execute(UIExecutionContext context) throws
Exception {
    File master = new File(folder.getValue() + File.separator +
"res" + File.separator + "values" + File.separator + "strings.
xml");
    File dest = new File(folder.getValue() + File.separator +
"res" + File.separator + "values" + File.separator + "strings_" +
language.getValue() + ".xml");

    FileUtils.copyFile(master, dest);
    return Results.success("COPIE faite : fichier " + dest.
getAbsolutePath() + " créé !");
  }
}
```

À l'exécution, le résultat sera celui présenté en figure 3.

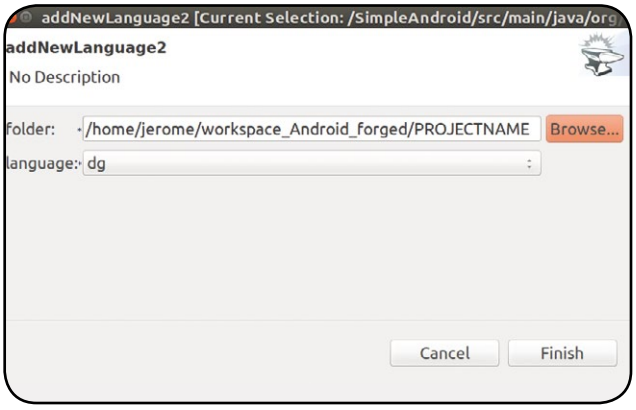


Fig. 3 : Sélection d'une variante de breton comme nouvelle langue pour une application

Et cela marche, le fichier **strings.xml** est dupliqué avec le suffixe choisi. Regardons le code de plus près. Les nouveautés par rapport au code précédent sont les attributs, composants graphiques, annotés avec **@Inject**. Il s'agit d'une annotation de la norme CDI. Le conteneur gère l'instanciation (selon le principe d'injection de dépendance).

Note

Notons le préfixe **UI** de ces classes, pratique pour rechercher avec **<Ctrl>** + **<Shift>** + **<T>**.

La ligne suivante est non-triviale (et pire, introuvable avec seulement l'autocomplétion) :

```
folder.getFacet(HintsFacet.class).setInputType(InputType.
DIRECTORY_PICKER);
```

Elle permet d'indiquer que le composant doit permettre de choisir un répertoire alors qu'il est déclaré comme stockant une chaîne :

```
private UIInput<String> folder;
```

Je vous invite à regarder les autres constantes définies dans **InputType** pour avoir la liste des types de données supportés par Forge.

Le caractère obligatoire d'une information est ajouté via la méthode **setRequired**.

Cette approche est déjà plus professionnelle, et grâce au choix des types de saisies et aux validations l'on peut commencer à partager ses *addons* avec des utilisateurs qui ne sont pas au fait de son implémentation.

L'inconvénient de cette approche est aussi son caractère mono-écran. On est toujours dans une vision simpliste : on saisit des valeurs, on valide et *l'addon* s'exécute. Impossible de faire faire une saisie à l'utilisateur sous condition de la valeur d'une autre saisie.

Un autre inconvénient, qui n'aura pas d'influence pour tous vos *addons* mais seulement pour ceux nécessitant le plus d'entrées par l'utilisateur, est que le nombre de *widgets* de saisie est limité par l'espace vertical de votre écran. Pour le test, j'ai créé une commande avec 10 paires de sélecteurs de répertoires et de listes déroulantes. Sur un écran 16/10 en 1080p, seules huit paires sont affichées et les deux dernières sont perdues car inaccessibles. Évidemment, faire saisir 20 informations à l'utilisateur ressemble (très) fortement à un problème de *design* mais, à mon humble avis, la limite est basse notamment pour les cas où l'interface serait créée à la volée à partir de métadonnées JDBC et où il faudrait proposer de modifier plusieurs informations (prérenseignées) par colonne (comme le type Java, le nom du champ, un commentaire, etc...). Une demande d'amélioration est ouverte sur ce point (pour que le conteneur soit *scrollable*) **[3]**.

Comme pour chacun de ses outils, il est bon d'en connaître les limites. En voici une.

2 Collaboration des addons

L'illustration la plus simple de la collaboration entre *addons* est l'utilisation des composants d'interface utilisateur telle que **UISelectOne** qui appartient à l'*addon* **UI**.

LINUX MAGAZINE / FRANCE

DÉCOUVREZ NOS OFFRES D'ABONNEMENTS !

PRO OU PARTICULIER = CONNECTEZ-VOUS SUR :

www.ed-diamond.com

LES COUPLAGES PAR SUPPORT :

VERSION PAPIER

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

VERSION PDF

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

ACCÈS À LA BASE DOCUMENTAIRE

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

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

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

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

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

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

LINUX MAGAZINE / FRANCE

Édité par 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 :

VOICI TOUTES LES OFFRES COUPLÉES AVEC GNU/LINUX MAGAZINE !

POUR LE PARTICULIER ET LE PROFESSIONNEL ...

Prix TTC en Euros / France Métropolitaine

CHOISISSEZ VOTRE OFFRE !

Prix en Euros / France Métropolitaine

Offre	SUPPORT / ABONNEMENT	PAPIER Réf	PAPIER Tarif TTC	PAPIER + PDF (PDF 1 lecteur) Réf	PAPIER + PDF Tarif TTC	PAPIER + BASE DOCUMENTAIRE (1 connexion BD) Réf	PAPIER + BASE DOCUMENTAIRE Tarif TTC	PAPIER + PDF + BASE DOCUMENTAIRE (PDF 1 lecteur + 1 connexion BD) Réf	PAPIER + PDF + BASE DOCUMENTAIRE Tarif TTC
LM	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 !

Tout comme l'utilisation de composants est possible, il est possible de créer un *addon* qui partagera ses fonctionnalités avec d'autres *addons*. Pour cela, il faut lors de la création du projet, cocher la case ***Create API, Impl, SPI, Tests and Addons modules*** (voir figure 4).

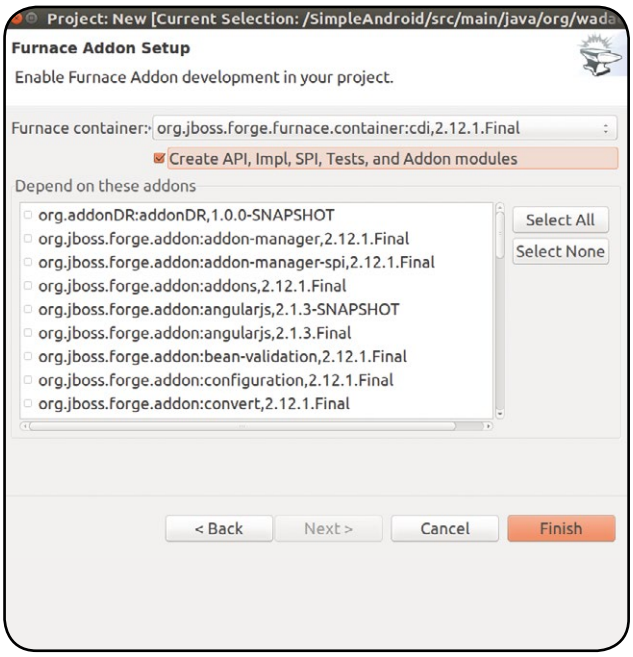


Fig. 4 : La terrible case à cocher qui peut remplir votre workspace

La première conséquence sera la création de multiples projets dans votre *workspace*. Ce cas de figure est donc à réserver aux *addons* auxquels vous souhaiterez faire proposer des fonctionnalités réutilisables par d'autres *addons*.

Note

Cela va sans dire mais cela va mieux en le disant, un *addon* restant un projet de type **Maven**, on pourra y ajouter des dépendances qui ne sont pas des *addons*.

3 Autres addons

Pour utiliser d'autres *addons*, il faut soit les choisir à la création du projet soit les ajouter ultérieurement dans son fichier de configuration Maven (**pom.xml**), sans oublier la balise suivante au sein de la déclaration de dépendance :

```
<classifier>forge-addon</classifier>
```

Note

Mon astuce est, à la création du projet, de sélectionner tous les *addons* disponibles pour qu'ils soient référencés dans le fichier **pom.xml**, où je mets en commentaire ceux que je n'utilise pas (encore).

3.1 Alliance ou alliage ? Mixer les addons

Je reviens non pas sur ce groupe des années 90 mais sur *l'addon* **Configuration**, qui permet de mémoriser des informations de façon persistante, qui seront disponibles *ad vitam æternam* et pour tous les *addons* puisque stockées sur disque. Il permet donc de passer des informations d'une commande à l'autre (sur un même poste).

D'une certaine manière, il fond les *addons* entre eux, crée un lien entre eux.

Dans le cadre de mon job de jour, j'ai dû écrire plusieurs commandes qui utilisent les mêmes informations JDBC (url, *username* et mot de passe), aussi, plutôt que de les faire saisir en paramètre de chaque commande, j'ai créé des commandes pour faire saisir ces informations et les partager avec les véritables *addons* utilisateurs qui s'arrêtent en échec (avec **return Results.fail("message ")**) si une des informations n'est pas saisie.

Le code ci-dessous décrit plusieurs commandes qui utilisent l'*addon* **Configuration** selon ce principe. Ainsi, cette première classe l'utilise pour mémoriser une valeur métier :

```
public class MyConfiguration {
  public static final String CLEF = "maClef_Valeur1";

  @Inject
  private Configuration conf;

  @Command(value="set-valeur1",categories="metier")
  public Result setLaValeurMetier(
    @Option(value = "valeur", label = "Quelle valeur donner ?",
required = true, type = "String", defaultValue = "8000", enabled
= true)
    String valeur){
    conf.setProperty(CLEF, valeur);
        return Results.success(CLEF +" vaut désormais " + conf.
getString(CLEF));
  }

  public String getLaValeurMetier(){
    return conf.getString(CLEF);
  }
}
```

On pourrait multiplier les valeurs à renseigner dans cette classe et pour chacune, créer une méthode de valorisation (**set**) et de consultation (**get**). La deuxième classe est la classe utilisatrice dans laquelle on injecte la première (merci Furnace version CDI).

```
public class MyConfigurationUser {
  @Inject
  MyConfiguration conf ;

  @Command(value="affiche-valeur1",categories="metier")
  public Result afficheLaValeurMetier(){
    String leopard = conf.getLaValeurMetier();
    if(leopard==null) return Results.fail("Pas de valeur pour " +
MyConfiguration.CLEF);
    return Results.success(MyConfiguration.CLEF + " vaut " +
leopard );
  }
}
```

L'utilisation de ces commandes dans la console donne :

```
[SimplistAddon]$ set-valeur1 --valeur 2222
***SUCCESS*** maClef_Valeur1 vaut désormais 2222
[SimplistAddon]$ affiche-valeur1
***SUCCESS*** maClef_Valeur1 vaut 2222
```

Attention !

À savoir : les informations ainsi stockées le sont en clair dans le fichier **~/.forge/config.xml**. Tout *addon* a accès à l'ensemble de ces informations. Oui, même ceux des autres !

3.2 Chers patrons

Parce que l'on ne peut pas générer tous les fichiers d'un projet avec des concaténations de *String*, il y a la possibilité d'utiliser des patrons (ou modèles, ou *templates*).

Basé sur le bien connu projet **FreeMarker [4]**, dont l'utilisation est des plus simples, le principe est de créer un fichier modèle contenant des références puis que ces références soient remplacées par les valeurs fournies. La formule magique est :

fichier modèle + données = fichier personnalisé

avec les données pouvant provenir d'objets personnalisés, de **Maps** ou de types simples.

Le code suivant lit un fichier contenu dans le jar de votre *addon* et l'affiche après avoir utilisé une **Map** comme source de données. Ce *template* Freemarker, nommé **voila.txt** a le contenu suivant :

```
Dennis dit :
${slogan}
```

Il est utilisé par le code suivant :

```
public class FileAddon extends AbstractUICommand {
  @Inject
  private TemplateFactory tfactory;
  @Inject
  private ResourceFactory rfactory;

  @Override
  public UICommandMetadata getMetadata(UIContext context) {
    return Metadata.forCommand(FileAddon.class).name("readVoila")
        .category(Categories.create("test"));
  }

  @Override
  public void initializeUI(UIBuilder builder) throws Exception {
  }

  @Override
  public Result execute(UIExecutionContext context) throws
Exception {
    URL voilaURL = getClass().getClassLoader().getResource("voila.
txt");
    Resource<URL> templateResource = rfactory.create(voilaURL);
    Template template = tfactory.create(templateResource,
FreemarkerTemplate.class);
    Map<String, Object> params = new HashMap<String, Object>();
    params.put("slogan", "JBoss Forge : c'est bon, mangez-en");
    String output = template.process(params) ;
    return Results.success(output);
  }
}
```

Il s'agit de récupérer une référence au fichier (URL) puis de la transformer en *template*. Les paramètres donnés serviront à des remplacements de chaînes.

Pour insérer des listes, utiliser les balises *#list*, comme ceci :

```
<#list entries as entry>
 ${entry.name}
</#list>
```

Ici la liste a été ajoutée sous le nom **entries**.

3.3 Classes à connaître

Puisque que l'orientation principale des *addons* est de générer du code source, autant savoir se repérer dans le projet.

Pour cela, la classe **org.jboss.forge.addon.projects.ProjectFactory** est essentielle. Obtenue par injection, elle permet d'obtenir le projet courant à une condition, qui est d'étendre non pas **AbstractUICommand** mais **AbstractProjectCommand** (quelques implémentations de méthodes à prévoir). Ensuite, un simple appel tel que **DirectoryResource dr = (DirectoryResource) proj.getRoot();** avec **Project proj = getSelectedProject(context);** vous renvoie un objet de type **DirectoryResource** qui correspond au répertoire racine du projet. À vous de voir si vous devez laisser choisir le répertoire ou non.

La classe **org.jboss.forge.addon.resource.ResourceFactory** a pour intérêt de vous permettre d'accéder à des ressources (**DirectoryResource**, **FileResource**, **JavaResource**, …).

La classe **org.jboss.forge.addon.templates.TemplateFactory** vue dans le code précédent permet de générer dynamiquement du contenu, que l'on insère par exemple dans une **FileResource** avec **setContents(String)**.

Avec ces seules trois classes, vous voici prêt à générer du code source ! Si vous devez étudier un code source, intéressez-vous à l'*addon* **parser-java**.

4 Des astuces !

Par défaut, le contenu du répertoire **src/main/resources** n'est pas mis dans le jar de l'*addon* :(Or, je préfère ne pas mettre les *templates* dans l'arborescence des sources (**src/main/java**) et les mettre dans **ressources**. Une petite modification du fichier **pom.xml** s'impose. Au niveau de la balise **<build>**, ajouter la partie en rouge suivante :

```
<build>
  <resources>
    <resource >
       <directory>${basedir}/src/main/
resources</directory>
    </resource>
  </resources>
…
```

Ayant eu besoin d'un driver JDBC, j'ai utilisé le traditionnel **Class.forName()**. Par réflexe acquis… sans résultats ! Imaginez mon désarroi ! C'est un peu plus compliqué mais la réponse se trouvait sur internet. Pour charger ce jar, j'ai utilisé cette méthode qui renvoie une connexion :

```
private Connection getConnection(Configuration configuration) {
   if (!driverSetupDone) {
     FileResource<?> resource = factory.create(FileResource.
class, new File(JDBCConfig.getJDBCDriversFolder(configuration)));
  if (resource == null) {
       return null;
     }
     File file = (File) resource.getUnderlyingResourceObject();
     if (resource != null && resource.exists()) {
       try (JarFile jarFile = new JarFile(file);) {
         URL[] urls = new URL[] { file.toURI().toURL() };
         URLClassLoader classLoader = URLClassLoader.
newInstance(urls);
         Class<?> driverClass = classLoader.loadClass(Driver.
class.getName());
         Enumeration<JarEntry> iter = jarFile.entries();
         while (iter.hasMoreElements()) {
           JarEntry entry = iter.nextElement();
           if (entry.getName().endsWith(".class")) {
             String name = entry.getName();
             name = name.substring(0, name.length() - 6);
             name = name.replace('/', '.');
             try {
               Class<?> clazz = classLoader.loadClass(name);
```

```
            } catch (ClassNotFoundException | NoClassDefFoundError
err) {
              // ignorons
            }
          }
        }
        driverClass = classLoader.loadClass(JDBCConfig.
getJDBCDriver(configuration));
        Driver d = (Driver) driverClass.newInstance();
        DriverManager.registerDriver(d);
        DriverManager.registerDriver(new DelegatingDriver(d));
      } catch (Exception e) {
        e.printStackTrace();
      }
    }
  }

  if (con == null)
    try {
      con = DriverManager.getConnection(JDBCConfig.
getJDBCUrl(configuration), JDBCConfig.
getJDBCUsername(configuration), JDBCConfig.getJDBCPassword(configu
ration));
    } catch (SQLException e) {
      e.printStackTrace();
    }
  return con;
 }
```

Au secours, n'est-il pas ? Je vous conseille de vous créer un package d'utilitaires avec des classes pour rassembler des appels tels que cette méthode.

5 Des buts dingues !

Petite homophonie pour introduire cette activité passionnante de notre quotidien de programmeur : la chasse aux erreurs i.e. débugger toute la journée (et le lendemain aussi). Parce qu'il est fort possible que votre code ne soit pas parfait à la première écriture, Dennis Ritchie a inventé le *debugging* (et si pour une fois ce n'est pas lui, il en était capable). Pour cela, la configuration la plus pratique à mon avis est d'écrire son code dans JBoss Developper Studio et d'avoir un shell où vous lancez Forge avec l'option de **`debug`** :

```
$ forge --debug
```

Ensuite, en complément, dans JBDS, sélectionner votre projet puis, par un clic droit, sélectionner ***Debug As > Remote Java Application***.

Ce sera dans le shell que vous devrez lancer l'installation de votre *addon* (**`addon-build-and-install`**) et son exécution.

Si, en modifiant vos sources, le *debugger* perd la synchronisation (*out of synch*), alors il est plus simple de quitter et relancer Forge dans un shell que de relancer Eclipse.

Autre problème que je ne suis pas le seul à avoir rencontré, c'est la non-mise à jour de l'*addon* en phase de développement. Après un *build*, la commande indiquée comme réussie, la/les commandes correspondantes ne sont pas accessibles pour autant alors que tout semble s'être déroulé sans accrocs. Il se trouve que le fichier jar est *locké* par Forge, ce qui doit être la source de son non-remplacement. Fâcheux mais signalé, ce souci est voué à disparaître rapidement. Pour le moment, le contournement que j'utilise est de désinstaller son *addon* à la main, avec ces quelques étapes :

- quitter Forge ;
- éditer **`~/.forge/addons/installed.xml`** pour y enlever la ligne correspondant à son *addon* ;
- supprimer le répertoire de son *addon*, dans **`~/.forge/addons/`** ;
- relancer Forge.

Note

Il est possible que ce problème soit réglé à l'heure où vous lirez ces lignes.

6 Nous avons vu...

Il y a énormément de raisons mais seulement trois manières de faire un *addon* Forge :

- la première est mono-écran et simpliste. C'est celle avec les annotations **`@Command`**. C'est la plus rapide, que l'on qualifierait de *« quick and dirty »*.
- la deuxième manière ajoute du choix dans les types de saisies proposées à l'utilisateur, ainsi que leur validation. Elle reste mono-écran, les types de saisies sont plus variés, des validations sont possibles.
- il en existe une troisième, non élaborée dans cet article, qui est multi-écran. Elle permet de créer des assistants avec un *workflow*. Il s'agit de la classe **`org.jboss.forge.addon.ui.wizard.UIWizard`**, non abordée ici mais vous avez maintenant les connaissances pour vous y atteler par vous-même.

Nous avons aussi vu qu'il est possible de réutiliser d'autres *addons*, notamment ceux du noyau (core, UI), dans ses propres *addons*.

En fonction de votre besoin, à vous de décider quelle approche employer pour vos développements, et créer les outils qui vous simplifieront le travail.

Conclusion

J'ai pour opinion que la génération de code sera dans un avenir proche aussi naturelle que l'utilisation de librairies tierces de nos jours. N'utilisons-nous pas déjà de nombreux assistants également ? Des IDEs sophistiqués ? Malgré tout, les outils disponibles sur le marché ne peuvent pas couvrir tous nos besoins, aussi avons-nous besoin d'une boîte à outils pour créer nos outils.

Issu de Red Hat, un grand contributeur de l'écosystème Open Source Java et géré par une équipe de qualité, JBoss Forge est un outil sur lequel il est raisonnable d'investir du temps et sur lequel baser vos sources.

À vous de battre le fer pendant qu'il est chaud et d'écrire vos *addons* dès aujourd'hui. Je vous laisse *fer*, souffler sur les braises de votre créativité et forger vos *addons*, vous avez les outils pour le faire. Ensuite, ayez la gentillesse de répondre à mon sondage à l'adresse suivante http://goo.gl/uRUQmf.

Puis, rejoignez-moi dans la création d'un *addon* plus ambitieux dédié à Android **[5]** ! ■

Références

[1] BATON J., « Les addons, c'est sa forge (1re partie)», *GNU/Linux Magazine* n°186, octobre 2015, p.72 à 77.

[2] BATON J., « JBoss Forge2, Java EE facile, très facile », *GNU/Linux Magazine* n°180, mars 2015, p. 72 à 82.

[3] Bug report pour un container scrollable : https://issues.jboss.org/browse/FORGE-2175

[4] FreeMarker : http://fr.wikipedia.org/wiki/Freemarker

[5] Mon Github : http://www.github.com/wadael

Remerciements

Je remercie Fabien Dubail, Koen Aers, Horacio Gonzalez et Philippe Charrière pour leur aimable travail de relecture de cet article, ainsi qu'Antonio Goncalves pour m'avoir fait découvrir l'extensibilité de cet outil, George Gastaldi pour son aide, ma société : DRiMS, qui m'a permis de travailler avec cet outil (8 h/j >> 3 h/nuit) et bien sûr, Sylvie qui s'occupe de nos enfants quand je travaille sur un article !

UN CHAT EN PHARO : LE SERVEUR

Stéphane DUCASSE [Responsable de l'équipe INRIA RmoD, laboratoire CRIStAL de l'Université Lille 1]
Olivier AUVERLOT [Membre de l'équipe RmoD, laboratoire CRIStAL de l'Université Lille 1]

À l'aide de Teapot, Pharo permet la définition d'un serveur REST en quelques dizaines de lignes. Grâce à lui, vous allez développer rapidement un serveur de chat basé sur HTTP et permettant l'échange de messages entre plusieurs interlocuteurs.

Mots-clés : Pharo, REST, HTTP, Serveur, Teapot, Zinc

Résumé

Cet article va décrire l'implémentation d'un serveur de chat basé sur HTTP. Écrit en Pharo, il utilise les *frameworks* Zinc et Teapot qui sont respectivement la couche HTTP de Pharo et un *framework* permettant la conception rapide de services basés sur la spécification REST.

L'objectif de cet article est de vous faire développer en trois classes, un serveur de chat. Cette petite aventure vous permettra de vous familiariser avec Pharo et de voir l'aisance avec laquelle un serveur REST peut être défini. Développée en quelques heures, **TinyChat** a été conçue comme une application pédagogique.

Avant toute chose, téléchargez Pharo à partir du site officiel du projet (www.pharo.org). Pharo est principalement disponible pour Linux, Mac et Windows. L'installation se résume à la décompression d'un fichier zip. Une fois Pharo lancé, vous devez lui ajouter le *framework* Teapot.

Vous pouvez charger Teapot en utilisant le *Configuration Browser* qui se trouve dans le menu ***Tools*** du menu principal. Sélectionnez ***Teapot*** et utilisez ***Install Stable***. Une autre solution consiste à utiliser le script suivant :

```
Gofer it
  smalltalkhubUser: 'zeroflag'
  project: 'Teapot';
  configuration;
  loadStable.
```

Votre environnement de développement est maintenant prêt. La première étape va consister à mettre en place l'élément de base d'un chat : les messages.

1 Architecture et représentation des messages

Nous allons donc construire un serveur de discussion. La communication entre le serveur et son client sera basée sur HTTP et REST. La classe principale sera `TCServer`. Nous définirons également deux autres classes : la classe `TCMessage` qui représente les messages échangés (dans le futur, vous pourrez étendre TinyChat pour échanger des éléments plus structurés comme JSON ou STON qui est le format textuel Pharo) et la classe `TCMessageQueue` qui stocke les messages.

1.1 La classe TCMessage

Un message est un objet très simple avec un texte et un identifiant pour l'émetteur.

Nous définissons la classe `TCMessage` dans le package `TinyChat` :

```
Object subclass: #TCMessage
  instanceVariableNames: 'sender text
separator'
  ClassVariableNames: ''
  category: 'TinyChat'
```

Les variables d'instances sont les suivantes :

- **sender** : le *login* de l'expéditeur ;
- **text** : le texte du message ;
- **separator** : un caractère séparateur pour l'affichage.

1.2 Gestion des variables d'instance sender et text

Nous créons les accesseurs suivants :

```
TCMessage >> sender
  ^sender

TCMessage >> sender: anObject
  ^sender := anObject

TCMessage >> text
  ^ text

TCMessage >> text: anObject
  text := anObject
```

1.3 Initialisation de la classe

La méthode **initialize** définit la valeur du caractère séparateur :

```
TCMessage >> initialize
  super initialize.
  separator := '>'.
```

La méthode de classe **TCMessage class>>from:text:** permet d'instancier un message :

```
TCMessage class >> from: aSender text:
aText
  ^ self new
    sender: aSender;
    text: aText;
    yourself
```

Le message **yourself** rend le receveur du message : c'est une manière de s'assurer que le nouvel objet créé sera bien retourné par le message **from:text:** et non le résultat du message **text:**.

1.4 Convertir un message en chaîne de caractères

Nous ajoutons une méthode **printOn:** pour transformer le message en une chaîne de caractères. Celle-ci est constituée des éléments suivants :

- l'identifiant de l'expéditeur ;
- le caractère séparateur ;
- le texte envoyé ;
- un retour à la ligne.

La méthode **printOn:** est invoquée par la méthode **printString**. Il est important de comprendre que la méthode **printOn:** est invoquée par les outils tels que le débogueur ou l'inspecteur d'objets. L'utilisation de **printOn:** dans vos projets Pharo constitue une aide précieuse pour la mise au point de votre code . En adoptant un formatage adapté, elle facilite considérablement la lecture des données gérées dans votre application.

```
TCMessage >> printOn: aStream
  aStream
    << self sender;
    << separator;
    << self text;
    << String crlf
```

1.5 Construire un message à partir d'une chaîne de caractères

Nous devons également définir deux méthodes pour créer un message à partir d'une chaîne ayant, par exemple, la forme: **'olivier>tinychat est cool'**. Tout d'abord, créons une méthode de classe qui sera invoquée de la manière suivante: **TCMessage fromString: 'olivier>tinychat est cool'**, puis la méthode d'instance remplissant les variables de l'objet préalablement créé.

```
TCMessage class >> fromString: aString
  ^ self new
    fromString: aString;
    yourself
TCMessage >> fromString: aString
  "Compose a message from a string of
this form 'sender>message'."
  | items |
  items := aString subStrings: separator.
  self sender: items first.
  self text: items second.
```

Maintenant, nous sommes prêts pour la définition du serveur REST.

2 Gestion des messages

Dans le serveur, nous allons ajouter une classe pour gérer une queue de messages. Ce n'est pas vraiment nécessaire, mais cela permet de bien identifier les responsabilités.

2.1 Stockage des messages

Créez la classe **TCMessageQueue** dans le package **TinyChat-Server** :

```
Object subclass: #TCMessageQueue
instanceVariableNames: 'messages'
classVariableNames: ''
category: 'TinyChat-server'
```

La variable d'instance **messages** est une collection ordonnée dont le contenu est composé d'instances de **TCMessage**. Une **OrderedCollection** est une collection qui s'agrandit dynamiquement lors d'ajouts.

```
TCMessageQueue >> initialize
  super initialize.
  messages := OrderedCollection new.
```

2.2 Opérations de base sur la liste des messages

On doit pouvoir ajouter un message avec **add:**, effacer la liste avec **reset** et connaître le nombre de messages avec **size** :

```
TCMessageQueue >> add: aMessage
  messages add: aMessage

TCMessageQueue >> reset
  messages removeAll

TCMessageQueue >> size
  ^ messages size
```

2.3 Obtenir la liste des messages à partir d'une position

Lorsqu'un client demande au serveur la liste des derniers messages échangés, il indique au serveur l'index du dernier message qu'il connaît. Le serveur répond alors la liste des messages reçus depuis cet index.

```
TCMessageQueue >> listFrom: aIndex
  ^ (aIndex > 0 and: [ aIndex <= messages
size])
    ifTrue: [ messages copyFrom: aIndex
to: messages size ]
    ifFalse: [ #() ]
```

2.4 Formatage des messages

La classe **TCMessageQueue** doit pouvoir formater une liste de messages (à partir d'un index) en une chaîne de caractères que le serveur pourra transmettre au client. On ajoute ensuite une méthode à la classe **TCMessageQueue** pour construire une seule chaîne de caractères à partir de chaque chaîne de caractères produite par chaque message :

```
TCMessageQueue >> formattedMessagesFrom:
aMessageNumber
  ^ String streamContents: [
:formattedMessagesStream |
    (self listFrom: aMessageNumber) do: [
:m |
      formattedMessagesStream << m
printString ]]
```

3 Le Serveur de Chat

Le cœur du serveur est basé sur le *framework* REST Teapot, permettant l'envoi et la réception des messages. Il maintient également une liste de messages qu'il communique aux clients.

3.1 Définition du routage HTTP

Créez la classe **TCServer** dans le package **TinyChat-Server** :

```
Object subclass: #TCServer
  instanceVariableNames: 'teapotServer
messagesQueue'
  classVariableNames: ''
  category: 'TinyChat-Server'
```

La variable d'instance **messagesQueue** référence la liste des messages reçus et envoyés par le serveur :

```
TCServer >> initialize
  super initialize.
  messagesQueue := TCMessageQueue new.
```

La variable d'instance **teapotServer** référence l'instance du serveur TeaPot que l'on crée à l'aide de la méthode **initializePort:**.

```
TCServer >> initializePort: anInteger
  teapotServer := Teapot configure: {
  #defaultOutput -> #text.
  #port -> anInteger.
  #debugMode -> true}.
  teapotServer start.
```

Le routage HTTP est défini dans la méthode **registerRoutes**. Trois opérations sont définies :

- **GET messages/count** : retourne au client le nombre de messages reçus par le serveur ;
- **GET messages/<id:IsInteger>** : le serveur retourne les messages à partir de l'index indiqué dans la requête HTTP ;
- **POST /message/add** : le client envoie un message au serveur.

```
TCServer >> registerRoutes
  teapotServer
  GET: '/messages/count' -> (Send
message: #messageCount to: self);
  GET: '/messages/<id:IsInteger>' ->
(Send message: #messagesFrom: to: self);
  POST: '/messages/add' -> (Send message:
#addMessage: to: self)
```

Nous exprimons ici que le chemin **/message/count** va donner lieu à l'exécution du message **messageCount** sur le serveur lui-même. Le pattern **<id:IsInteger>** indique que l'argument doit être exprimé sous forme de nombre et qu'il sera converti en un entier. La gestion des erreurs est construite dans la méthode **registerErrorHandlers**. Ici, on voit comment est construite une instance de la classe **TeaResponse**.

```
TCServer >> registerErrorHandlers
  teapotServer
    exception: KeyNotFound -> (TeaResponse
notFound body: 'No such message')
```

Le démarrage du serveur est confié à la méthode **TCServer class>>startOn:** qui reçoit le numéro de port TCP en paramètre.

```
TCServer class >> startOn: aPortNumber
  ^self new
    initializePort: aPortNumber;
    registerRoutes;
    registerErrorHandlers;
    yourself
```

Il faut également gérer l'arrêt du serveur. La méthode **stop** met fin à l'exécution du serveur TeaPot et vide la liste des messages.

```
TCServer >> stop
  teapotServer stop.
  messagesQueue reset.
```

Comme il est probable que vous exécutiez plusieurs fois l'expression **TCServer startOn:**, nous définissons la méthode de classe **stopAll** qui stoppe tous les serveurs en récupérant toutes les instances de la classe. La méthode **TCServer class>>stopAll** demande l'arrêt de chaque instance du serveur.

```
TCServer class >> stopAll
  self allInstancesDo: #stop
```

3.2 Traitements réalisés par le serveur

La méthode **addMessage** extrait de la requête du client le message posté. Elle ajoute à la liste des messages une nouvelle instance de **TCMessage**.

```
TCServer >> addMessage: aRequest
  messagesQueue add: (TCMessage from:
(aRequest at: #sender) text: (aRequest
at: #text))
```

La méthode **messageCount** retourne le nombre de messages reçus.

```
TCServer >> messageCount
  ^ messagesQueue size
```

La méthode **messageFrom:** retourne la liste des messages reçus par le serveur depuis l'index indiqué par le client. Les messages sont retournés au client sous la forme d'une chaîne de caractères. Ce point sera définitivement à améliorer (JSON ou XML sont plus adaptés si vous voulez faire évoluer ce projet).

```
TCServer >> messagesFrom: request
  ^ messagesQueue formattedMessagesFrom:
(request at: #id)
```

Nous en avons fini avec le serveur. À ce stade, nous pouvons le tester un peu. Commençons par le lancer :

```
TCServer startOn: 8181
```

Pour émettre des requêtes, nous pouvons utiliser un navigateur ou le client HTTP disponible par défaut dans Pharo (voir résultat en figure 1).

```
ZnClient new url: 'http://localhost:8181/messages/count' ; get
```

Les amateurs du shell peuvent également utiliser la commande **curl** :

```
curl http://localhost:8181/messages/count
```

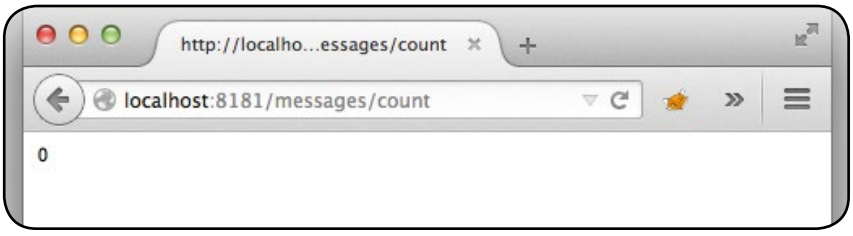


Fig. 1 : Le serveur n'a reçu aucun message.

Dans Pharo, nous pouvons aussi ajouter un message de la manière suivante :

```
ZnClient new
url: 'http://localhost:8181/messages/add';
formAt: 'sender' put: 'olivier';
formAt: 'text' put: 'Super cool ce tinychat';
post
```

Conclusion

Nous avons montré que la création d'un serveur REST est extrêmement simple avec Teapot. À ce stade, vous pouvez tester le bon fonctionnement du serveur et pourquoi pas, expérimenter en modifiant certaines méthodes ou en ajoutant de nouvelles fonctionnalités. Certes, le produit est encore brut de fonderie en termes de confort d'utilisation. Pour cette raison, dans le prochain article, vous réaliserez un client graphique pour le serveur. L'objectif sera de concevoir avec un minimum de code, un client de chat basé sur le *framework* **Spec**. ■

Pour aller plus loin

- Le site du projet Pharo : http://www.pharo.org
- L'ouvrage collectif « *Pharo par l'exemple* », Square Bracket Associates, 2011
- L'ouvrage collectif « *Deep inside Pharo* », Square Bracket Associates, 2013

Références

[1] Site officiel du projet Pharo : http://www.pharo.org

[2] Site officiel du projet Zinc : http://zn.stfx.eu/zn/index.html

[3] Le framework Teapot :http://smalltalkhub.com/#!/~zeroflag/Teapot

[4] Télécharger le projet TinyChat : http://www.smalltalkhub.com/#!/~olivierauverlot/TinyChat

UN CHAT EN PHARO : LE CLIENT

Stéphane DUCASSE [Responsable de l'équipe INRIA RmoD, laboratoire CRIStAL de l'Université Lille 1]
Olivier AUVERLOT [Membre de l'équipe RmoD, laboratoire CRIStAL de l'Université Lille 1]

Poursuivons la découverte de Pharo et de quelques-uns de ses principaux frameworks. Vous avez déjà fait connaissance avec Teapot permettant de concevoir des services. Dans cet article, vous allez étudier la construction de requêtes HTTP à l'aide de Zinc [1] et construire une interface graphique à l'aide de Spec.

Mots-clés : Pharo, REST, HTTP, Spec, Zinc

Résumé

L'article précédent a été l'occasion d'implémenter un serveur de chat basé sur HTTP et la philosophie REST. Le problème est que notre solution est pour l'instant assez peu conviviale. Pour dialoguer, il est nécessaire d'utiliser des requêtes HTTP et même si vous disposez d'un client HTTP tel que Curl ou Postman, il n'est guère pratique de communiquer de cette manière. Pour cette raison, vous allez maintenant créer un client graphique pour Pharo [2] qui vous permettra d'échanger des messages avec les autres personnes connectées au serveur.

Nous allons nous concentrer sur la partie client de TinyChat. Pour la construire, nous allons simplement ajouter deux classes au projet :

- **TinyChat** est la classe contenant la logique métier (connexion, envoi et réception des messages) ;
- **TCConsole** est une classe définissant l'interface graphique.

La logique du client est la suivante :

- au lancement du client, celui-ci demande au serveur l'index du dernier message reçu ;
- toutes les deux secondes, le client se connecte au serveur pour lire les messages échangés depuis sa dernière connexion. Pour cela, il transmet au serveur l'index du dernier message dont il a eu connaissance.

De plus, lorsque le client transmet un message au serveur, il en profite également pour lire les messages échangés depuis sa dernière connexion.

1 La classe TinyChat

Nous créons la classe **TinyChat** dans le *package* **TinyChat-client** :

```
Object subclass: #TinyChat
  instanceVariableNames: 'url login exit messages console
lastMessageIndex'
  classVariableNames: ''
  category: 'TinyChat-client'
```

Cette classe définit les variables suivantes :

- **url** contient l'url HTTP permettant au client de se connecter au serveur ;
- **login** est une chaîne de caractères identifiant le client ;
- **messages** est une variable d'instance contenant les messages lus par le client ;

- **lastMessageIndex** est le numéro du dernier message lu par le client ;
- **exit** est une valeur booléenne. Tant que cette valeur est vraie, le client se connecte à intervalle régulier au serveur pour lire les messages échangés depuis sa dernière connexion ;
- **console** est une instance de **TCConsole** : une console graphique permettant à l'utilisateur de saisir et de consulter les messages.

Nous initialisons les variables qui le nécessitent dans la méthode **initialize** :

```
TinyChat >> initialize
  super initialize.
  exit := false.
  LastMessageIndex := 0.
  messages := OrderedCollection new.
```

2 Définir les commandes HTTP

Nous devons maintenant définir les méthodes permettant au client HTTP de communiquer avec le serveur.

Deux méthodes permettent d'assembler les chemins d'accès. L'une n'a pas d'argument et permet de construire les requêtes **/messages/add** et **/messages/count**. L'autre méthode a un argument qui est utilisé pour la lecture des messages à partir d'une position.

```
TinyChat >> command: aPath
  ^'{1}{2}' format: { url . APath }

TinyChat >> command: aPath argument: anArgument
  ^'{1}{2}/{3}' format: { url . aPath . anArgument asString }
```

Il suffit ensuite de définir les trois commandes HTTP du client :

```
TinyChat >> cmdLastMessageID
  ^ self command: '/messages/count'

TinyChat >> cmdNewMessage
  ^self command: '/messages/add'

TinyChat >> cmdMessagesFromLastIndexToEnd
  "Returns the server messages from my current last index to the
last one on the server."
  ^ self command: '/messages' argument: lastMessageIndex
```

3 Gérer les opérations du client

Nous avons besoin d'émettre ces commandes et de pouvoir récupérer des informations à partir du serveur. Pour cela, nous définissons deux méthodes. La méthode **readLastMessageID** retourne l'index du dernier message reçu par le serveur :

```
TinyChat >> readLastMessageID
  | id |
  id := (ZnClient new url: self cmdLastMessageID; get) asInteger.
  id = 0 ifTrue: [ id := 1 ].
  ^ id
```

La méthode **readMissingMessages** ajoute les derniers messages reçus par le serveur à la liste des messages connus par le client. Cette méthode retourne le nombre de messages récupérés :

```
TinyChat >> readMissingMessages
  "Gets the new messages that have been posted since the last
request."
  | response receivedMessages |
  response := (ZnClient new url: self
cmdMessagesFromLastIndexToEnd; get).
  ^ response
    ifNil: [ 0 ]
    ifNotNil: [
      receivedMessages := response subStrings: (String crlf).
      receivedMessages do: [ :msg | messages add: (TCMessage
fromString: msg)].
      receivedMessages size.
    ]
```

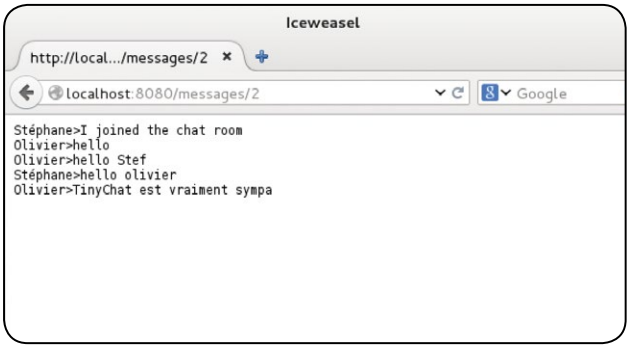


Fig. 1 : Lecture de l'historique des messages reçus par le serveur.

Nous sommes prêts à définir le comportement de rafraîchissement du client avec la méthode **refreshMessages**. Elle utilise un processus léger pour lire à intervalle régulier les messages reçus par le serveur. Le délai est fixé à deux secondes. Le message **fork**, envoyé à un bloc (une fermeture lexicale en Pharo), exécute ce bloc dans un processus léger. La logique est de boucler tant que le client ne spécifie pas qu'il veut s'arrêter via la variable **exit**. L'expression **(Delay forSeconds: 2) wait** suspend l'exécution du processus léger dans lequel elle se trouve, pendant un certain nombre de secondes :

```
TinyChat >> refreshMessages
  [
    [ exit ] whileFalse: [
    (Delay forSeconds: 2) wait.
    lastMessageIndex := lastMessageIndex + (self
readMissingMessages).
    console print: messages.
    ]] fork
```

La méthode **sendNewMessage:** poste le message de l'utilisateur au serveur :

```
TinyChat >> sendNewMessage: aMessage
  ^ ZnClient new
    url: self cmdNewMessage;
    formAt: 'sender' put:  (aMessage sender);
    formAt: 'text' put: (aMessage text);
    post
```

Cette méthode est utilisée par la méthode du client **send:** qui reçoit en paramètre le texte saisi par l'utilisateur. La chaîne de caractères est alors convertie en une instance de **TCMessage**. Le message est ensuite envoyé. Le client met à jour l'index du dernier message connu et déclenche l'affichage du contenu du message dans l'interface graphique :

```
TinyChat >> send: aString
  | msg |
  msg := TCMessage from: login text: aString.
  self sendNewMessage: msg.
  lastMessageIndex := lastMessageIndex + (self
readMissingMessages).
  console print: messages.
```

La déconnexion du client est gérée par la méthode **disconnect** qui envoie un message au serveur pour signaler le départ de l'utilisateur. Elle met fin à la boucle de récupération périodique des messages.

```
TinyChat >> disconnect
  self sendNewMessage: (TCMessage from: login text: 'I exited from
the chat room.').
  exit := true
```

3.1 Fixer les paramètres du client

Pour initialiser les paramètres de connexion, on définit une méthode de classe **TinyChat class>>connect:port:login:**. Cette méthode permet de se connecter de la manière suivante :

```
TinyChat connect: 'localhost' port: 8080 login: 'username'
```

Le code de cette méthode est donc :

```
TinyChat class >> connect: aHost port: aPort login: aLogin
  ^ self new host: aHost port: aPort login: aLogin; start
```

Le code appelle la méthode **host:port:login:**. Cette méthode met à jour la variable d'instance **url** en construisant l'URL et en affectant le nom de l'utilisateur à la variable d'instance **login** :

```
TinyChat >> host: aHost port: aPort login: aLogin
  url := 'http://' , aHost , ':' , aPort asString.
  login := aLogin
```

La méthode **start** envoie un message au serveur pour présenter l'utilisateur, récupérer l'index du dernier message reçu par le serveur et mettre à jour la liste des messages connus par le client. C'est également cette méthode qui initialise l'interface graphique de l'utilisateur. Une évolution pourrait consister à séparer le modèle de son interface graphique en utilisant une conception basée sur des événements.

```
TinyChat >> start
  console := TCConsole attach: self.
  self sendNewMessage: (TCMessage from: login text: 'I joined the
chat room').
  lastMessageIndex := self readLastMessageID.
  self refreshMessages.
```

4 Création de l'interface graphique

L'interface graphique est composée d'une fenêtre contenant une liste et un champ de saisie :

```
ComposableModel subclass: #TCConsole
  instanceVariableNames: 'chat list input'
  classVariableNames: ''
  category: 'TinyChat-client'
```

La variable d'instance **chat** est une référence à une instance de la classe **TinyChat** et nécessite uniquement un accesseur en écriture.

Les variables d'instance **list** et **input** disposent chacune d'un accesseur en lecture. Ceci est imposé par le constructeur d'interface Spec :

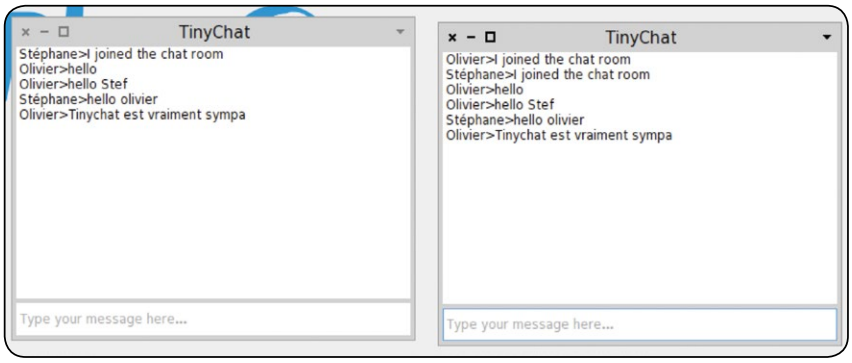


Fig. 2 : L'interface graphique de TinyChat.

```
TCConsole >> input
  ^ input

TCConsole >> list
  ^ list

TCConsole >> chat: anObject
  chat := anObject
```

L'interface graphique possède un titre pour la fenêtre. Pour le définir, il faut écrire une méthode **title** :

```
TCConsole >> title
  ^ 'TinyChat'
```

La méthode de classe **TCConsole class>>attach:** reçoit en argument l'instance du client de chat avec laquelle l'interface graphique va être utilisée. Cette méthode déclenche l'ouverture de la fenêtre et met en place l'événement gérant la fermeture de celle-ci ainsi que la déconnexion du client :

```
TCConsole class >> attach: aTinyChat
| window |
window := self new chat: aTinyChat.
window openWithSpec whenClosedDo: [
aTinyChat disconnect ].
^ window
```

La méthode **TCConsole class>>defaultSpec** définit la mise en page des composants contenus dans la fenêtre. Nous avons une colonne avec une liste et un champ de saisie placé juste en dessous :

```
TCConsole class >> defaultSpec
  <spec: #default>
  ^ SpecLayout composed newColumn: [ :c |
    c add: #list; add: #input height: 30
]; yourself
```

La méthode **initializeWidgets** spécifie la nature et le comportement des composants graphiques. Ainsi, le code passé à **acceptBlock:** permet de définir l'action à exécuter lorsque le texte est entré dans le champ de saisie. Nous l'envoyons à **chat** et nous effaçons son contenu lorsque l'utilisateur appuie sur la touche **<Entrée>**.

```
TCConsole >> initializeWidgets
  list := ListModel new.
    input := TextInputFieldModel new
    ghostText: 'Type your message here...';
    enabled: true;
  acceptBlock: [ :string |
    chat send: string.
    input text: '' ].
  self focusOrder add: input.
```

La méthode **print:** affiche les messages reçus par le client en les affectant au contenu de la liste.

```
TCConsole >> print: aCollectionOfMessages
  list items: (aCollectionOfMessages
collect: [ :m | m printString ])
```

Notez que cette méthode est invoquée par la méthode **refreshMessages** et que changer tous les éléments de la liste à chaque ajout d'un nouveau message est peu élégant mais l'exemple se veut volontairement simple.

Conclusion

Et voilà ! Quelques classes regroupant quelques dizaines de lignes de code vous ont permis de construire un sympathique petit outil de chat. La définition de TinyChat donne un cadre ludique à l'exploration de la programmation en Pharo et nous espérons que vous avez apprécié cette ballade. TinyChat est une petite application que nous avons développée de manière très simple afin de vous permettre de l'étendre et d'expérimenter. Il y a de nombreuses améliorations possibles : gestion parcimonieuse des ajouts d'éléments dans la liste graphique, gestion d'accès concurrents dans la collection sur le serveur (en effet, si le serveur pouvait recevoir des requêtes concurrentes la structure de données utilisée n'est pas adéquat), gestion des erreurs de connexion, rendre les clients robustes à la fermeture du serveur, obtenir la liste des personnes connectées, pouvoir définir le délai de récupération des messages, utiliser JSON ou XML pour le transport des messages, afficher le nom de la personne connectée dans la fenêtre. Le projet est disponible sur le site de dépôt Smalltalkhub **[3]**. À vous de jouer! ■

Pour aller plus loin

- L'ouvrage collectif « *Pharo par l'exemple* », Square Bracket Associates, 2011
- L'ouvrage collectif « *Deep inside Pharo* », Square Bracket Associates, 2013

Références

[1] Site officiel du projet Zinc: http://zn.stfx.eu/zn/index.html

[2] Site officiel du projet Pharo : http://www.pharo.org

[3] Télécharger le projet TinyChat : http://www.smalltalkhub.com/#!/~olivierauverlot/TinyChat

FAISONS UN PEU DE CHIMIE HTML AVEC POLYMER

Tristan COLOMBO

Un polymère est une « molécule géante », encore appelée macromolécule, qui est constituée de la répétition de molécules formant un motif générique. Polymer est un framework de développement Web qui se base sur la même logique : des éléments qui peuvent être combinés pour développer plus rapidement. Partons à la découverte de cet outil.

Mots-clés : Polymer, Javascript, Google, Web components, HTML, CSS

Résumé

Nous allons découvrir le *framework* Polymer et comprendre son fonctionnement de manière à pouvoir créer nos propres *y*.

Polymer est un *framework* de développement Web développé par Google. Sa première version stable, l'actuelle version 1.0, a été publiée en juin 2015. L'objectif de ce *framework* est de proposer un développement rapide, adapté aux différents supports (tablettes, smartphones, ordinateurs) et navigateurs tout en respectant les principes d'ergonomie du **Material Design**. Pour fournir des éléments divers directement réutilisables dans vos codes, Polymer utilise les *Web components* qui sont des composés de quatre spécifications du W3C :

- les *custom elements* pour définir de nouveaux types (des types personnalisés) d'éléments du DOM **[1]** ;
- les *html imports* pour inclure et réutiliser des documents html à l'intérieur d'autres documents html **[2]** ;
- les *templates* qui, comme leur nom l'indique, permettent d'utiliser des modèles, c'est-à-dire des zones dans lesquelles des fragments html pourront être inclus à l'aide de scripts **[3]** ;
- le *shadow DOM* pour injecter des éléments dans le DOM en respectant les interdépendances entre les arbres DOM **[4]**.

Il faut savoir que de ces quatre spécifications, seuls les *templates* ne sont plus à l'état de *draft*. Donc, forcément, tous les navigateurs n'implémentent pas entièrement les *Web components* et vous pourrez visualiser précisément les fonctionnalités prises en charge par votre navigateur fétiche en vous rendant sur la page http://caniuse.com/#search=web%20components. La figure 1 montre le tableau récapitulatif de l'implémentation des fonctionnalités des *Web components* en fonction du navigateur et de sa version.

Pour fonctionner sur la majorité des navigateurs, Polymer utilise le projet **webcomponents.js [5]** qui permet d'employer les *Web components* sur les navigateurs ne les prenant pas encore en charge. Si vous utilisez Firefox, vous pouvez également tenter d'activer les *Web components* en passant la valeur de `dom.webcomponents.enabled` à `true` dans about:config.

Je vous propose maintenant d'aller un peu plus loin en ayant un aperçu des composants proposés par Polymer, puis en l'installant et en comprenant comment créer un nouvel élément.

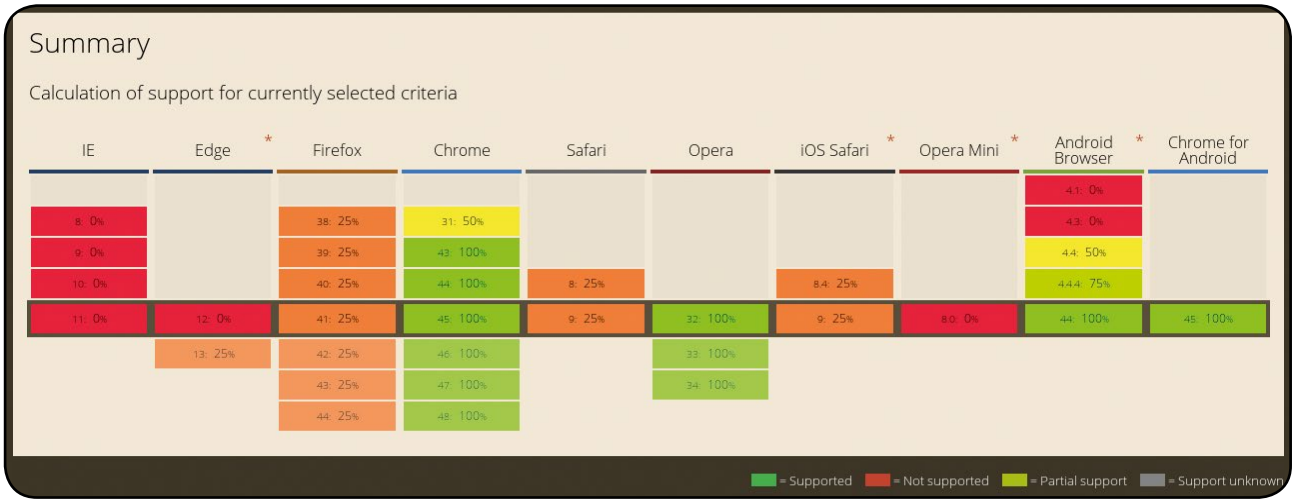


Fig. 1: Tableau récapitulatif du taux d'implémentation des fonctionnalités des Web components suivant les navigateurs et leur version (source http://www.caniuse.com, donnée de septembre 2015).

1 Les éléments de Polymer

Polymer propose des éléments regroupés dans sept grandes familles présentées comme des éléments du tableau périodique de Mendeleïev **[6]**. Ces familles sont réparties comme suit :

- **Fe** (fer): éléments de base, actuellement au nombre de 36. On y trouve des éléments permettant de simplifier les appels ajax (`iron-ajax`), simplifier la gestion des listes (`iron-list`) ou des formulaires (`iron-form`), etc. On peut citer également `iron-icon` et `iron-icons` pour l'affichage d'icônes.
- **Md** (papier) : éléments graphiques au « format » Material Design. Parmi les 31 éléments, on y trouve par exemple des boutons de différentes formes et couleurs. La documentation fournie sur le site de Polymer est d'ailleurs très complète et propose de nombreux exemples comme l'affichage des cases à cocher que vous pouvez voir en figure 2 (page suivante).
- **Go** (Google Web) : éléments d'accès aux services Googles, au nombre de 15. On peut ainsi insérer facilement une carte Google Map (`google-map`), gérer une authentification avec un compte Google (`google-signin`), etc.
- **Au** (Or) : 6 éléments dédiés au commerce électronique. On peut y trouver par exemple un champ de saisi de numéro de carte bleue (`gold-cc-input`).
- **Ne** (néon) : l'élément `neon-animation`, car il n'y en a qu'un, est dédié aux animations (transitions, listes, grilles, etc).
- **Pt** (platine) : 4 éléments pour transformer une page Web en véritable application Web.
- **Mo** (molécules) : pour l'instant un seul élément dans cette catégorie « fourre-tout » qui contiendra les « autres » bibliothèques javascript. On y trouve `marked-element` permettant d'interpréter du Markdown.

En passant rapidement en revue les familles d'éléments, on peut s'apercevoir que Polymer est déjà plutôt bien fourni. Jetons donc un œil à l'installation pour voir si la simplicité d'utilisation est au rendez-vous.

2 Installation

Il est recommandé d'installer Polymer via **Bower [7]**. Si vous ne disposez pas encore de celui-ci, vous trouverez ci-dessous la liste des commandes à exécuter sur un système Debian-like (y compris l'installation de **Node.js**) :

```
# aptitude install nodejs npm git
# npm install -g bower
```

Comme nous sommes dans un schéma d'application Web, les fichiers de Polymer devront être présents dans chaque application et la phase d'installation sera donc à répéter pour chaque nouveau projet. Nous allons débuter un projet nous permettant de tester ce *framework* en créant un répertoire `hello_world`. C'est dans ce répertoire qu'il faudra procéder à l'installation de Polymer :

```
$ mkdir hello_world
$ cd hello_world
$ bower init
```

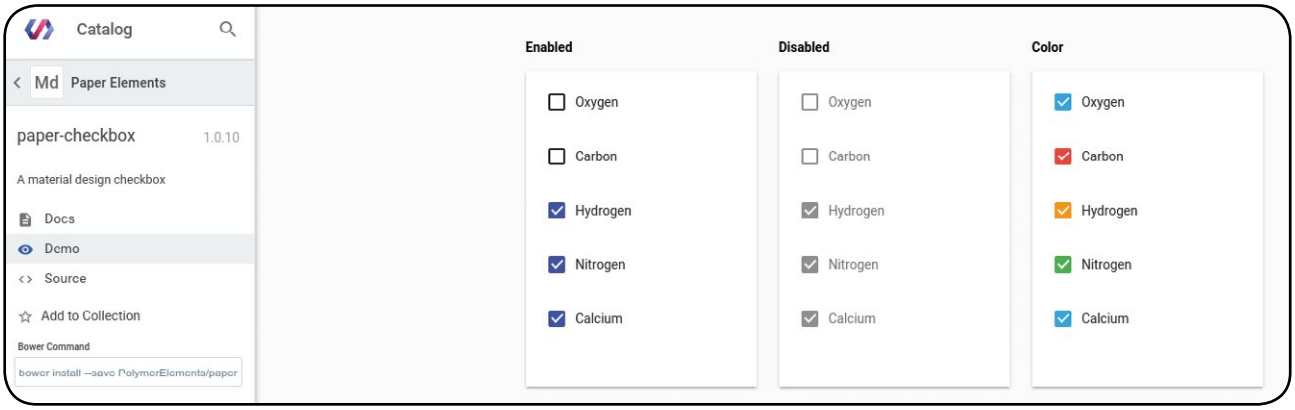


Fig. 2 : Cases à cocher paper-checkbox de la documentation de Polymer.

Note

Si vous souhaitez installer une version particulière d'un élément, vous pourrez le préciser en utilisant la syntaxe `nom_élément#^version`. Par exemple, pour la version 1.2.0 de Polymer, il faudrait taper :

```
$ bower install --save Polymer/polymer#^1.2.0
```

Vous devrez alors répondre à une série de questions permettant l'élaboration du fichier de configuration `bower.json`. À la question *« what types of modules does this package expose? »*, vous pouvez appuyer directement sur **<Return>** pour ne rien sélectionner. L'étape suivante consiste à installer Polymer :

```
$ bower install --save Polymer/polymer
```

Un nouveau répertoire `bower_components` a dû apparaître. Vous trouverez dans celui-ci les deux sous-répertoires `polymer` et `webcomponentsjs` pour la compatibilité de Polymer avec un maximum de navigateurs. Pour mes tests, pour m'abstraire le plus possible des risques de mauvaise interprétation du code, j'ai utilisé le navigateur **Google Chrome** (mieux connu sous le nom de **Chromium** sous Linux).

Note

L'option `--save` de la commande précédente a permis d'ajouter Polymer en tant que dépendance du projet dans le fichier `bower.json`.

3 Une page simple

Pour notre hello world nous allons rester sur quelque chose de simple : un bouton qui affiche le message « Hello World ! » lorsque l'on clique dessus.

3.1 Le démarrage d'un projet Polymer

Il va falloir charger Polymer pour pouvoir l'utiliser... mais en fait ce chargement va se faire élément par élément de manière à ne pas surcharger la page. Dans cet exemple, nous allons utiliser l'élément `paper-button` qui crée un bouton et `paper-toast` qui affiche un message sous la forme de « toast » : une petite zone apparaissant en bas de l'écran disparaissant automatiquement. Il faut donc commencer par installer les éléments Polymer nécessaires à notre programme :

```
$ bower install --save polymerelements/paper-button
$ bower install --save polymerelements/iron-toast
```

Nous pouvons maintenant écrire le code de notre application naissante :

```
01:  <!doctype html>
02:
03:  <html lang="fr">
04:      <head>
05:          <meta charset="utf-8" />
06:          <meta name="viewport" content="width=device-width,
minimum-scale=1.0, initial-scale=1.0, user-scalable=yes" />
07:          <title>GLMF Polymer</title>
08:          <script src="bower_components/webcomponentsjs/
webcomponents.js"></script>
09:
10:          <link rel="import" href="bower_components/paper-
button/paper-button.html" />
11:          <link rel="import" href="bower_components/paper-
toast/paper-toast.html" />
12:      </head>
```

```
14:     <body unresolved>
15:         <paper-button raised onclick="alert('Hello
World!');">Appuyez ici</paper-button>
16:     </body>
17: </html>
```

On retrouve le squelette d'un document html5 standard dans lequel on charge la bibliothèque **webcomponentsjs** (ligne 8) de manière à garantir la plus grande compatibilité possible entre navigateurs. Il faut également charger les éléments Polymer que l'on va utiliser. Ceci est réalisé dans les lignes 10 et 11.

L'attribut **unresolved** associé au tag **<body>** permet de prévenir tout affichage de contenu non stylé (les fameux FOUC pour *Flash Of Unstyled Content*). Il s'agit en fait d'une pseudo-classe qui masque l'affichage de l'élément **<body>** jusqu'au chargement complet de celui-ci.

Vous pouvez ensuite voir le tag personnalisé de Polymer associé à l'affichage d'un bouton en lignes 15 à 19. Pour l'instant nous avons fait les choses le plus simplement possible et le clic sur le bouton produit simplement l'affichage d'un message d'alerte sous forme de popup.

Si vous tentez d'ouvrir votre page dans un navigateur... vous n'obtiendrez rien. Au mieux, si vous avez une console de développement ouverte, vous pourrez voir un message *« Imported resource from origin 'file://' has been blocked from loading by Cross-Origin Resource Sharing policy: Invalid response. Origin 'null' is therefore not allowed access »*. Les imports html ne fonctionnent qu'avec des ressources issues de la même origine. Ici il nous faut donc servir nos pages html localement pour lever cette restriction. Le plus simple est d'utiliser **Python3** en lançant la commande suivante dans le répertoire du projet :

```
$ python3 -m http.server
```

Vous n'avez plus qu'à vous rendre à l'adresse localhost:8000 pour voir le bouton tel que présenté en figure 3.

Avant d'afficher notre message avec **paper-toast** nous allons prendre le temps d'appliquer un style sur le bouton pour le rendre plus visible.

Fig. 3 : Le bouton paper-button sans style personnalisé.

3.2 Style personnalisé

Les éléments Polymer sont déjà créés mais rien ne vous empêche de modifier leur style en précisant l'attribut **is="custom-style"** dans une balise de style :

```
01:  <!doctype html>
02:
03:  <html lang="fr">
04:      <head>
...
11:          <link rel="import" href="bower_components/paper-
toast/paper-toast.html" />
12:
13:          <style is="custom-style">
14:              paper-button[raised].colorful {
15:                  background-color: #4285f4;
16:                  color: #fff;
17:              }
18:          </style>
19:      </head>
...
22:          <paper-button raised class="colorful"
onclick="alert('Hello World!');">Appuyez ici</paper-button>
...
```

Attention !

Si vous souhaitez charger votre style depuis une ou des feuilles externes, vous ne pourrez pas utiliser les méthodes classiques. Il vous faudra placer les directives de style dans un fichier html contenant la balise **<style>** puis importer ce dernier. Dans le cas de notre exemple, nous pourrions placer le style dans **styles/style.html** :

```
01:  <style is="custom-style">
02:      paper-button[raised].colorful {
03:          background-color: #4285f4;
04:          color: #fff;
05:      }
06:  </style>
```

L'import se fait ensuite comme un simple fichier html :

```
01:  <!doctype html>
02:
03:  <html lang="fr">
04:      <head>
...
11:          <link rel="import" href="bower_components/paper-
toast/paper-toast.html" />
12:
13:          <link rel="import" href="styles/style.html" />
14:      </head>
...
```

Nous avons indiqué ici que nous souhaitions que les boutons **paper-button** disposant de l'attribut **raised** et de la classe **colorful** soient bleus avec un texte blanc.

3.3 Ajout du toast

Nous pouvons maintenant ajouter l'affichage du message sous forme de *toast*. Tout étant déjà géré, il n'y a pas beaucoup de lignes à insérer ou à modifier :

```
01:  <!doctype html>
02:
03:  <html lang="fr">
04:      <head>
05:          <meta charset="utf-8" />
06:          <meta name="viewport" content="width=device-width,
minimum-scale=1.0, initial-scale=1.0, user-scalable=yes" />
07:          <title>GLMF Polymer</title>
08:          <script src="bower_components/webcomponentsjs/
webcomponents.js"></script>
09:
10:          <link rel="import" href="bower_components/paper-
button/paper-button.html" />
11:          <link rel="import" href="bower_components/paper-
toast/paper-toast.html" />
12:
13:          <link rel="import" href="styles/style.html" />
14:      </head>
15:
16:      <body unresolved>
17:          <paper-button raised class="colorful"
onclick="document.querySelector('#helloWorld').show()">Appuyez
ici</paper-button>
18:
19:          <paper-toast id="helloWorld" text="Hello world!"></
paper-toast>
20:      </body>
21:  </html>
```

La méthode **show()** de la ligne 17 permet d'afficher le *toast* **[8]** de la ligne 19 qui porte l'identifiant **helloWorld**. Le résultat obtenu pour cette page en cliquant sur le bouton est visible en figure 4.

Fig. 4 : Affichage du message après appui sur le bouton.

3.4 Création d'un élément

Imaginons maintenant que ce bouton avec son message « Hello world ! » soit quelque chose que l'on emploie souvent. Il serait dommage de faire des copier/coller ou de réécrire le comportement de ce bouton à chaque fois. Nous allons donc créer un nouvel élément ou *Web component*. Cet élément s'appellera par exemple **hello-world** et nous placerons le fichier permettant de le définir dans **my_components/hello-world/hello-world.html** de manière à adopter la même nomenclature de fichiers que celle utilisée dans Polymer. Sachant cela nous pouvons déjà écrire le contenu du fichier **index.html** :

```
01:  <!doctype html>
02:
03:  <html lang="fr">
04:      <head>
05:          <meta charset="utf-8" />
06:          <meta name="viewport" content="width=device-width,
minimum-scale=1.0, initial-scale=1.0, user-scalable=yes" />
07:          <title>GLMF Polymer</title>
08:          <script src="bower_components/webcomponentsjs/
webcomponents.js"></script>
09:
10:          <link rel="import" href="my_components/hello-world/
hello-world.html" />
11:      </head>
12:
13:      <body unresolved>
14:          <hello-world></hello-world>
15:      </body>
16:  </html>
```

Cette page ne produit bien sûr aucun affichage tant que nous n'avons pas créé le fichier **hello-world.html** définissant notre *Web component*.

Un nouvel élément est défini par **<dom-module>** et l'attribut **id** donne le nom du *Web component*. On trouve ensuite deux parties : **<template>** qui contient le fragment html qui sera inséré à l'endroit où nous utiliserons notre élément et **<script>** qui enregistre le *Web component* et définit son fonctionnement.

Dans la partie Javascript, pour enregistrer notre élément, nous devrons faire appel à **Polymer()()** en lui passant en paramètre un tableau associatif contenant *à minima* la clé **is** relative au nom du **dom-module** (le nom de notre élément). La clé **properties** permettra de définir les attributs attendus ainsi que leur type (facultatif). Voici le squelette d'un fichier permettant la création d'un élément :

```
<link rel="import" href="../../bower_components/polymer/polymer.
html">
<!-- import des autres éléments utilisés dans le Web component -->

<dom-module id="nom-élément">
```

```
    <template>
        <!-- description de la page html -->
    </template>

    <script>
        Polymer({
            is: 'nom-élément',
            properties: {              nom-attribut: Type,
                // autres noms d'attributs
            },
            // Définition éventuellement de fonctions
        })();
    </script>
</dom-module>
```

Pour mettre en place notre élément, d'après le code que nous avions déjà écrit vous devriez déjà voir que nous allons ajouter les imports de **paper-button** et **paper-toast** en début de fichier, que **nom-élément** sera pour nous **hello-world** et que les tags html **<paper-button>** et **<paper-toast>** seront placés dans la partie **<template>**. Voici ce que cela donne :

```
01:  <link rel="import" href="../../bower_components/polymer/
polymer.html">
02:  <link rel="import" href="../../bower_components/paper-
button/paper-button.html" />
03:  <link rel="import" href="../../bower_components/paper-toast/
paper-toast.html" />
04:
05:  <dom-module id="hello-world">
06:
07:      <template>
08:          <paper-button raised class="colorful"
onclick="document.querySelector('#helloWorld').show()">Appuyez
ici</paper-button>
09:
10:          <paper-toast id="helloWorld" text="Hello world!">
</paper-toast>
11:      </template>
12:
13:      <script>
14:          Polymer({
15:              is: 'hello-world',
16:          })();
17:      </script>
18:
19:  </dom-module>
```

Comme vous le voyez, ce n'est pas très compliqué... mais nous avons perdu notre style personnalisé ! Nous allons l'ajouter à notre élément pour qu'il soit transmis avec lui et nous allons en profiter pour améliorer notre *Web Component* en rendant le texte du bouton paramétrable et déporter le traitement du clic sur celui-ci dans une fonction.

3.5 Un élément amélioré

Nous allons améliorer notre élément par étapes de manière à bien comprendre ce que nous faisons.

3.5.1 Ajout d'un attribut

Pour commencer, nous allons ajouter à notre élément un attribut **text** qui permettra d'indiquer le texte à afficher sur le bouton. Pour le déclarer, nous avons vu qu'il fallait utiliser la clé **properties** dans le dictionnaire fourni à **Polymer()**. Pour l'utiliser dans **<template>** il faudra employer son nom entre deux accolades. Voici la modification à effectuer :

```
01:  <link rel="import" href="../../bower_components/polymer/
polymer.html">
...
05:  <dom-module id="hello-world">
06:
07:      <template>
08:          <paper-button raised class="colorful"
onclick="document.querySelector('#helloWorld').show()">{{text}}
</paper-button>
...
13:      <script>
14:          Polymer({
15:              is: 'hello-world',
16:              properties: {
17:                  text: String,
18:              },
19:          })();
20:      </script>
21:
22:  </dom-module>
```

Bien sûr, dans **index.html** il faudra modifier le tag **<hello-world>** en ajoutant l'attribut **text** :

```
14:          <hello-world text="Appuyez ici!"></hello-world>
```

3.5.2 Définition du comportement de l'élément

L'appel à la fonction **show()** se faisait jusqu'à présent dans l'élément **paper-button** mais nous allons maintenant placer le code définissant le comportement après un clic sur le bouton dans le dictionnaire de paramétrage de notre *Web component*. Pour cela nous créons une nouvelle clé faisant référence à une fonction :

```
01:  <link rel="import" href="../../bower_components/polymer/
polymer.html">
...
05:  <dom-module id="hello-world">
06:
07:      <template>
08:          <paper-button raised class="colorful" on-
tap="displayToast">{{text}}</paper-button>
09:
10:          <paper-toast id="{{toast}}" text="Hello world!">
</paper-toast>
11:      </template>
12:
13:      <script>
14:          Polymer({15:              is: 'hello-world',
```

```
16:             properties: {
17:                 text: String,
18:                 toast: String,
19:             },
20:             displayToast: function () {
21:                 document.querySelector('#' + this.toast).
show();
22:             },
23:         })();
24:     </script>
25:
26: </dom-module>
```

Vous remarquerez que j'ai modifié le nom de l'événement déclencheur de l'action lors du clic sur le bouton : il s'agit maintenant de **on-tap**, événement de Polymer correspondant à un appui et un relâchement (vous verrez que l'animation sur le bouton est plus fluide). De plus, j'en ai profité pour rendre l'identifiant de **paper-toast** paramétrable à l'aide de l'attribut **toast** (ce qui permet d'utiliser plusieurs éléments **hello-world** au sein d'une même page). Bien entendu, il ne faut pas oublier de définir l'attribut dans **index.html** :

```
14:         <hello-world text="Appuyez ici!"
toast="helloWorld"></hello-world>
```

3.5.3 Insertion du style

Pour finir, il nous reste à joindre le style que nous avions défini précédemment. Il va nous falloir créer un élément de style que nous importerons et que nous utiliserons dans notre élément. Cet élément se trouvera dans **my_components/hello-world/hello-world-style.html** :

```
01: <dom-module id="hello-world-style">
02:     <template>
03:         <style>
04:             paper-button[raised].colorful {
05:                 background-color: #4285f4;
06:                 color: #fff;
07:             }
08:         </style>
09:     </template>
10: </dom-module>
```

Il s'agit ici d'un élément minimaliste ne contenant que la partie **<template>** et les directives de style. Pour l'utiliser dans **hello-world.html** il va falloir ajouter deux lignes :

```
01: <link rel="import" href="../../bower_components/polymer/
polymer.html">
...
05: <link rel="import" href="hello-world-style.html" />
06:
07: <dom-module id="hello-world">
08:
09:     <template>
10:         <style include="hello-world-style"></style>
11:         <paper-button raised class="colorful" on-
tap="displayToast">{{text}}</paper-button>
12:
13:         <paper-toast id="{{toast}}" text="Hello world!"></
paper-toast>
14:     </template>
...
29: </dom-module>
```

On retrouve ainsi le comportement initial de notre élément tout en ayant une meilleure structure et la possibilité de le réutiliser.

Conclusion

Lorsque l'on a l'habitude de voir des documents html monolithiques cela peut paraître décousu... mais à l'usage cela permet de créer des éléments paramétrables et réutilisables. Nous n'avons pu découvrir ici qu'une infime partie de ce *framework* mais il est vraiment très intéressant. Plus d'excuse pour développer une interface complètement contraire aux règles d'ergonomie élémentaires et ce en un rien de temps. Bien sûr, il faut garder à l'esprit que Google est derrière tout cela et que ce framework est là pour imposer le style Google et va conduire à l'uniformisation des interfaces, mais cela apporte un grand confort au développeur et à l'utilisateur qui sait où chercher les informations.

Pour conclure, je ne peux m'empêcher de souligner le fait que créer des éléments à la suite les uns des autres fait forcément naître une chaîne et l'on peut donc dire que *Polymer has chain reaction* (les biologistes comprendront... :-)). ■

Références

[1] Spécifications des custom elements : http://w3c.github.io/webcomponents/spec/custom/

[2] Spécifications des imports html : http://w3c.github.io/webcomponents/spec/imports/

[3] Spécifications des templates : http://www.w3.org/TR/html5/scripting-1.html#the-template-element

[4] Spécifications du shadow DOM : http://w3c.github.io/webcomponents/spec/shadow/

[5] Projet Webcomponents.js : https://github.com/WebComponents/webcomponentsjs

[6] Catalogue des éléments de Polymer sous forme de tableau périodique : https://elements.polymer-project.org/

[7] Site officiel de Bower : http://bower.io/

[8] Documentation de paper-toast : https://elements.polymer-project.org/elements/paper-toast

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