N°190 février 2016

GNU LINUX MAGAZINE / FRANCE

DÉVELOPPEMENT ET ADMINISTRATION SUR SYSTÈMES OPEN SOURCE

LES « HOW-TO » DU SYSADMIN

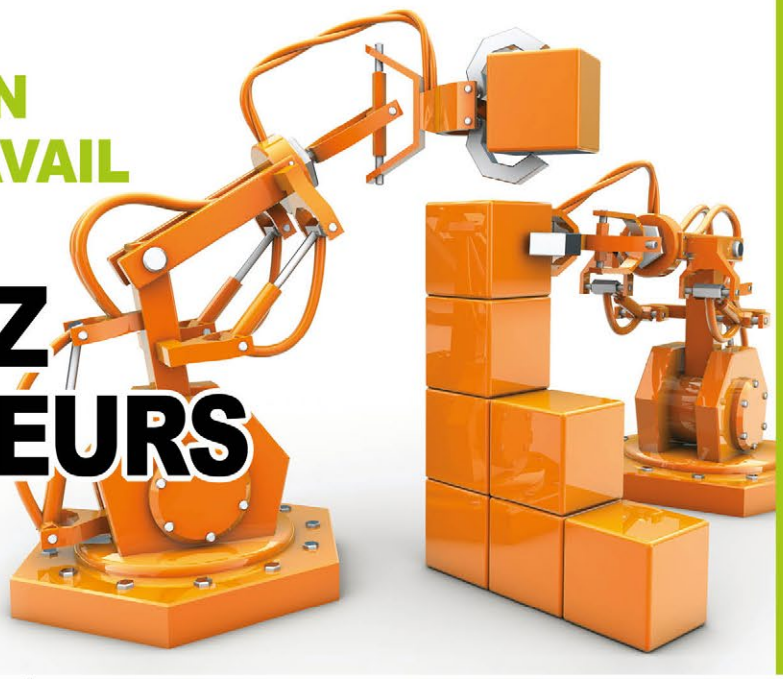
Sauvegardez les données de votre smartphone Android avec SSH p.16

AUTOMATISATION / PLAYBOOKS

N'ENVOYEZ JAMAIS UN HUMAIN FAIRE LE TRAVAIL D'UNE MACHINE !

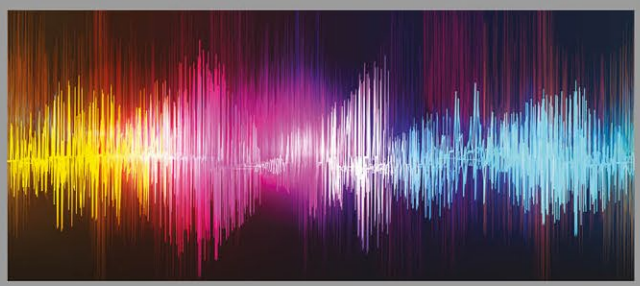
AUTOMATISEZ VOS CONTENEURS

AVEC DOCKER & ANSIBLE ! p.30



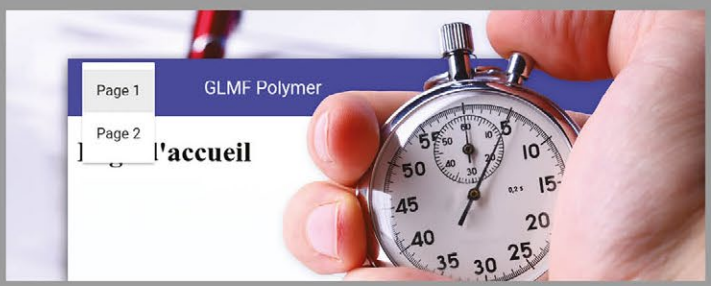
LANGAGE C / SON

Créez des fichiers audio en C au format WAV p.44



JAVASCRIPT / POLYMER

Développez un menu ergonomique en 5 minutes ! p.78



ASCIIDOCTOR / DOC

Écrivez votre documentation avec Asciidoc p.64

UDEV / SYSTEMD

Déclenchez automatiquement vos sauvegardes sur clé USB p.20

PHARO / ARTEFACT

Générez simplement des documents PDF p.68

ET AUSSI : *Interfaces Python en CLI – Multicast avec MariaDB*

L 19275 - 190 - F: 7,90 € - RD

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

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

~~4,99~~ **0,99** € HT/mois (1,19 € TTC)*
À partir de

MANAGED WORDPRESS

100 % PERFORMANCE

- **NOUVEAU !** Espace Web **illimité avec SSD**
- **NOUVEAU !** Bases de données **illimitées sur SSD**
- **NOUVEAU ! PHP 7 avec OPcache**
- Trafic **illimité**
- Comptes email **illimités**
- **2 Go de RAM** garantis

100 % SÉCURITÉ

- **NOUVEAU ! Protection contre les attaques DDoS avec NGINX** pour encore plus de performance, de fiabilité et de sécurité
- **Géo-redondance :** hébergement simultané dans des data centers distincts
- 1&1 CDN avec Railgun™
- 1&1 SiteLock

100 % CONFORT

- **NOUVEAU !** L'**Assistant WP** vous guide dans l'installation et dans le choix du design
- **Inclus : thèmes prêts à l'emploi**
- **Assistance 24/7**
- Support Expert WordPress
- 1&1 Community

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

*Les packs Managed WordPress sont à partir de 0,99 € HT/mois (1,19 € TTC) pour un engagement minimum de 12 mois. À l'issue des 12 premiers mois, les prix habituels s'appliquent. Certaines fonctionnalités citées ne sont pas disponibles dans tous les packs. Offres sans durée minimum d'engagement également disponibles. Conditions détaillées sur 1and1.fr. Rubik's Cube® utilisé avec l'accord de Rubik's Brand Ltd. 1&1 Internet SARL, RCS Sarreguemines B 431 303 775.

GNU/Linux Magazine France
est édité par Les Éditions Diamond

LES ÉDITIONS DIAMOND

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

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

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

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

Périodicité : Mensuel
Prix de vente : 7,90 €

MIXTE Papier issu de sources responsables FSC® C015136 www.fsc.org

Membre April www.april.org

ÉDITORIAL

Je vais revenir ce mois-ci sur un problème récurrent : l'ergonomie et l'utilisabilité des interfaces Web. De plus en plus de services sont proposés en ligne et c'est à se demander si leurs concepteurs les utilisent ou les ont seulement testés. Dans les dernières pages qu'il m'ait été donné de tester, on peut trouver :

- un formulaire qui disparaît à la validation lorsque l'un des champs contient une entrée incorrecte ;
- un formulaire dont l'ascenseur se trouve intégré à la page de façon tellement esthétique qu'il est invisible : on ne sait donc pas comment accéder au bas du formulaire et au bouton de validation ;
- des messages sur un site marchand (plutôt connu) où, sous prétexte de convivialité, les concepteurs ont glissé un peu d' « humour » : « Et hop ! C'est dans votre panier ! » ou encore « Veuillez patienter, la page suivante va se charger automatiquement... ou pas :-) ». Et bien en tant que client je ferai pareil : je reviendrai... ou pas !
- un panier qui indique qu'il contient un article alors que tous les articles ont été supprimés ;
- un bouton « page suivante » qui renvoie sur la page d'accueil ;
- un formulaire de contact qui, une fois connecté et donc identifié, redemande à l'utilisateur de saisir ses nom, prénom, adresse, numéro de client, etc. Faire une requête dans la base de données est sans doute trop compliqué !
- j'ai gardé le meilleur pour la fin avec un formulaire contenant un champ obligatoire sous la forme de boutons radio... et dont tous les boutons sont désactivés ! Je dois faire partie des quelques personnes en France ayant pu valider le formulaire en l'éditant et en modifiant les lignes `<input type="radio" disabled="disabled" ...>`. Ce formulaire n'est pas censé s'adresser aux seuls informaticiens, mais peut-être s'agit-il d'un test caché ?

Comment tout cela est possible ? Comment ces erreurs de conception peuvent-elles perdurer ? Pourquoi continuer d'accepter de tels agissements de la part de certains développeurs ? Si votre garagiste répare votre voiture et vous explique que pour démarrer il faudra dorénavant tourner trois fois sur vous-même avant d'ouvrir le capot, ajouter un peu d'huile, faire tourner le moteur deux minutes puis l'éteindre avant de redémarrer... est-ce que vous trouverez ça normal ? Certes non, pourtant nous n'avons pas tous des connaissances en mécanique auto. Alors pourquoi en informatique trouve-t-on normal d'avoir des programmes qui fonctionnent « à peu près » ou que les clients se voient proposer des solutions abracadabrantesques de contournement des problèmes ? Le logiciel libre est un début de réponse, mais n'est plus suffisant : de nombreuses entreprises surfent sur l' « open source », détournant à leur profit le système économique du logiciel libre en produisant des logiciels au code obscur et non documenté de manière à pouvoir revendre par la suite des formations ou de nouveaux développements.

Il n'y aurait donc aucune solution ? Si ! Râler comme on le ferait avec notre garagiste... et continuer à lire et faire lire *GNU/Linux Magazine* pour acquérir de nouveaux savoirs :-)

Tristan Colombo

COURRIER DES LECTEURS

Écrivez-nous à lecteurs@gnulinuxmag.com !

Bonjour Tristan,

Cool, « Ils sont fous ces romains ».

J'ai recopié le code (mais je me suis peut-être planté !).

Pour la valeur **14022**, j'obtiens bien : **(XIV)(1)XXII**

Par contre pour **100**, j'obtiens : **(C)(1)C**

Était-ce bien le résultat attendu ?

Olivier S.

Bonjour,

Non, ce n'est effectivement pas le résultat attendu ! :-)

Il s'agit d'une erreur de partitionnement et j'aurais dû tester un peu plus... mais ça me permettra d'écrire un article sur les tests unitaires.

Pour résoudre le problème, il suffit de modifier la fin du **range** de la ligne 33 :

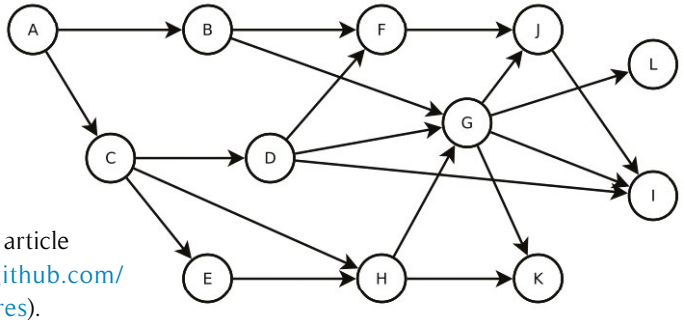
```
...
[strDecimal[i:] for i in range(-3, -len(strDecimal)-3, -3)]))
```

Merci d'avoir porté cette erreur à ma connaissance.

T.C.

Errata

Dans l'article « Parcourir des graphes en largeur » de Nicolas Patrois, en p. 26 de *GNU/Linux Magazine n°189*, la figure 6 n'est pas la bonne. Il s'agit en fait de la figure ci-contre.



J'en profite pour signaler que les codes de cet article ont été mis à disposition sur GitHub (https://github.com/GLMF/GLMF189/tree/master/Rep%C3%A8res).

SOMMAIRE

GNU/LINUX MAGAZINE FRANCE N°190

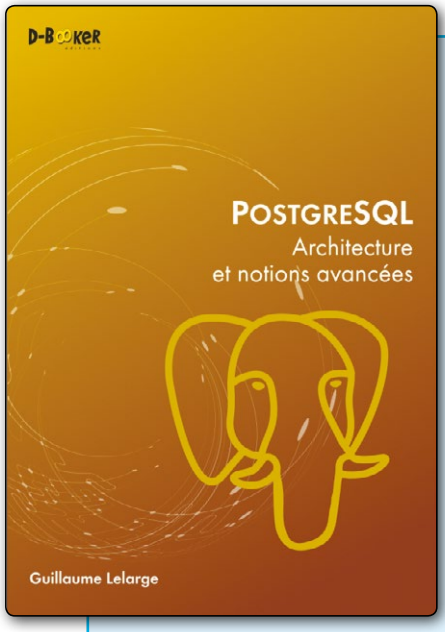
POSTGRESQL - ARCHITECTURE ET NOTIONS AVANCÉES

Jean-Michel Armand [Cofondateur Hybird, Développeur Crème CRM et Djangonaute]

Au moment où ces lignes sont écrites, la version 9.5 de PostgreSQL est en RC, sa sortie est donc imminente. C'est un moment parfait pour vous parler d'un livre sorti tout récemment et qui vous dévoilera ce qui se passe sous le capot de cet excellent moteur de base de données.

Résumé

PostgreSQL est un serveur de base de données aussi puissant que pléthorique en terme de fonctionnalités. Mais, et ce n'est pas un mystère, ce n'est pas forcément le moteur de base de données le plus simple à appréhender. *PostgreSQL Architecture et notions avancées* se propose de vous expliquer en détail comment fonctionne et s'architecture chaque partie de PostgreSQL. Chacune de ses 398 pages vous fera découvrir une partie du fonctionnement de l'un des moteurs de base de données libre les plus utilisés et les plus performants qui soient.



L'auteur, Guillaume Lelarge

Guillaume Lelarge est un contributeur important de PostgreSQL. Il dirige la traduction en français du manuel de Pgsql et a participé très activement au développement de l'outil d'administration **pgAdmin**. Actif dans la communauté, il s'implique à la fois dans la gestion de l'association PostgreSQL Europe ainsi que dans les organisations des conférences européennes sur PostgreSQL. Enfin si vous êtes un lecteur assidu de *GNU/Linux Magazine*, vous avez forcément déjà lu sa prose que ce soit pour parler de réplication, de nouveautés des nouvelles versions de PostgreSQL, etc.

1 Présentation de PostgreSQL Architecture et notions avancées

Le livre, publié aux Éditions D-Booker, passe en revue chaque notion importante pour comprendre comment fonctionne PgSQL. Bien que cela ne soit pas effectivement le cas, le livre se découpe pour moi en trois grands ensembles, trois parties. On va avoir une première partie

regroupant les chapitres 1 à 5. Puis une courte partie contenant les chapitres 6 et 7 et enfin la dernière partie commençant au chapitre 8 pour finir avec le dernier chapitre, le 15.

On commence la première partie par une rapide explication de ce qu'est une instance PG. On passe ensuite à l'étude des fondamentaux en terme de ressources consommées par un programme, quel qu'il soit. Un programme utilise des fichiers où il va stocker les informations nécessaires à son fonctionnement. Un programme est composé d'au moins un, mais souvent plusieurs, processus qui vont utiliser de la mémoire pour stocker ou échanger des informations au cours de son fonctionnement.

La première partie du livre dissèque donc PgSQL comme l'on pourrait disséquer le fonctionnement de n'importe quel programme. On commence par étudier quels sont les fichiers qui sont utilisés par le moteur de base de données pour fonctionner et où ils se trouvent. On plonge un peu plus en profondeur en étudiant la structure de chacun des fichiers utilisés par PgSQL. On passe ensuite à la description des différents processus lancés. Un après l'autre, leur utilité sera explicitée ainsi que le moment où ils se lancent. Ensuite on découvre comment PgSQL utilise la mémoire, que ce soit pour mettre en place des caches ou des communications entre les différents processus.

Mais PostgreSQL n'est pas seulement un processus, c'est aussi un serveur. Il doit donc gérer les connexions des clients ainsi que celle des autres serveurs. Cela implique de gérer différents protocoles. Les deux chapitres suivants, composant la deuxième partie, nous expliquent cela. Protocole de communication client ↔ serveur et serveur ↔ serveur sont passés à la loupe ainsi que les différentes possibilités de connexions.

La dernière partie, qui est à mon sens la plus intéressante, se focalise sur l'étude de ce qui définit un moteur de base de données. L'ordonnancement des chapitres de cette partie est très bien fait. On commence par trois chapitres détaillant les notions que l'on manie à chaque utilisation courante de PostgreSQL. On verra donc la gestion des transactions puis on découvrira ensuite les différents types d'objets de PgSQL et leur utilité et on finira par une explication détaillée de comment fonctionne la planification des requêtes et l'optimiseur de celles-ci. On finira avec des chapitres plus axés administrateurs ou devops. Ils traiteront tour à tour du couple sauvegarde et restauration, de la réplication, de la manière de mettre en place des statistiques d'activités, des opérations de maintenance nécessaires à mettre en place pour garder un serveur de base de données sain. Le livre se clôt sur un court chapitre, mais ô combien important, sur la sécurité.

2 Mon avis : est-ce que ce livre est pour vous ?

Si vous espérez trouver un catalogue de tutoriels de configuration prêt à l'utilisation en achetant ce livre, alors ne l'achetez pas. *PostgreSQL Architecture et notions avancées* ne vous apprendra pas à configurer un serveur PostgreSQL.

Par contre, si vous voulez comprendre ce que vous avez vraiment fait en suivant aveuglement les tutoriels que vous avez pu trouver sur le net ou dans la littérature disponible sur le sujet, si vous voulez savoir à quoi sert précisément la variable `random_page_cost` et quel peut être l'impact de sa modification, alors ce livre est pour vous. Chaque notion est clairement expliquée. Chaque variable de configuration ainsi que les choix possibles et leurs impacts sont décrits et placés dans le contexte qui convient.

Si vous êtes un développeur, vous pourriez penser que ce livre n'est pas pour vous. Et même si certains chapitres ne seront intéressants que pour améliorer votre compréhension globale, ce qui n'est jamais un mal, ils vous seront vraiment très utiles. Combien de fois n'ai-je entendu quelqu'un dire « Tiens, pour que cela soit plus rapide, je ne vais pas mettre de transaction, ça sera moins lourd ». Après avoir lu ce livre, vous saurez que PostgreSQL ne fonctionne qu'avec des transactions. Si elles ne sont pas explicitement déclarées, il les générera de manière implicite... tant pis pour l'optimisation espérée. Comment fonctionne l'optimiseur de requêtes, quels sont les différents types d'index, autant de notions qui sont expliquées et, j'insiste, d'une manière très claire, très fluide.

Par contre, si je devais faire un reproche, ce serait au niveau de l'ordre des chapitres. Alors oui, définir tous les fichiers utilisés par PgSQL et étudier le contenu de chacun d'entre eux est quelque chose d'important. Mais c'est loin d'être le plus amusant et motivant à lire. Et les deux chapitres suivants qui parlent de processus et de gestion de la mémoire sont à peine plus motivants. On se retrouve donc à commencer à lire des choses qui sont claires, peut-être, mais surtout arides. Et il faut tenir, pendant 150 pages pour enfin arriver à des choses excitantes. C'est tout de même une sacrée barrière à l'entrée. Pour éviter cela, suivez mon conseil : ne lisez pas dans l'ordre de la table des matières. Faites-vous plaisir et commencez par les chapitres vraiment fonctionnels. Vous lirez les premiers chapitres, un peu plus arides, par petits morceaux. ■

UNE SI DISCRÈTE INSTITUTION

Tris Acatrinei [Projet Arcadie]

Née très rapidement après l'Hadopi, alors qu'elle dispose de pouvoirs particulièrement étendus, l'ARJEL (Autorité de Régulation des Jeux En Ligne) ne fait que très peu parler d'elle. Pourtant, un récent colloque qu'elle a organisé laisse présager de nouveaux pouvoirs qui ne sont guère encourageants.

Mots-clés : ARJEL, HADOPI, Blocage de sites, Régulation, Jeux en ligne

Résumé

À chaque secteur, son autorité administrative indépendante et le jeu en ligne n'a pas échappé à ce qui semble devenu une marotte du législateur. Mais le dernier Gouvernement avait bénéficié d'une sorte de trêve médiatique, passant ainsi presque sous silence les prérogatives d'une institution pas si inoffensive que ça.

1 Une création opportun(ist)e

Le 7 octobre 2009 s'ouvre à l'Assemblée nationale la discussion du projet de loi relatif à l'ouverture à la concurrence et à la régulation du secteur des jeux d'argent et de hasard en ligne **[1]**, projet présenté par Éric Woerth, ministre du budget, des comptes publics, de la fonction publique et de la réforme de l'État. Le ministre justifie son action par la prolifération des offres illégales, faisant concurrence aux offres légales, mais affirme dès les premières minutes que l'interdiction pure et simple des jeux d'argent sur Internet serait inutile et inefficace. Le texte s'appuie donc sur deux éléments :

- « *permettre l'émergence d'une offre légale structurée autour d'acteurs visibles et connus des joueurs* » ;
- « *des outils de lutte qui visent à accumuler les obstacles sur le chemin des sites illégaux.* »

Après cette introduction, c'est le rapporteur de la commission des finances, de l'économie générale et du contrôle budgétaire, Jean-François Lamour qui prend la parole et il est amusant de noter qu'il revient sur l'expérience de la loi HADOPI 2 en disant : « *c'est l'ARJEL qui demandera aux fournisseurs d'accès à internet et aux hébergeurs de bloquer l'accès à ces sites illégaux [...]. Alors que la discussion du projet de loi HADOPI 2 avait suscité beaucoup de craintes parmi nous, je tiens à souligner avec force que c'est l'accès aux sites illégaux – et seulement lui – qui sera bloqué. Il ne s'agira ni de couper l'accès à internet de nos concitoyens, ni de collecter leurs adresses IP, ni de restreindre leur liberté constitutionnelle de communication et d'expression. Ce sera probablement difficile tant l'inventivité des gérants de sites illégaux est grande pour contourner toute tentative de blocage; reste que je suis convaincu que les FAI sauront trouver des techniques efficaces pour faire respecter la loi française.* ».

Allant plus loin dans le détail des différentes actions qui pourront être menées par la future autorité, Étienne Blanc, rapporteur pour avis de la commission des lois constitution-

nelles, de la législation et de l'administration générale de la République, contredisant presque accidentellement le rapporteur précédent, énonce sobrement que « *L'ARJEL comprendra une commission des sanctions qui pourra infliger des amendes ou une suspension ou un retrait de l'agrément en cas de non-respect des obligations. Pour exercer son contrôle, elle se fera communiquer toutes les données relatives aux joueurs et aux sessions de jeu et elle disposera d'agents assermentés qui pourront mener des enquêtes administratives.* »

Seule différence entre l'ARJEL et l'Hadopi : l'absence de sanctions envers les joueurs alors que le dispositif de la réponse graduée ne cible que les internautes qui n'auraient pas sécurisé leur accès à Internet. En dehors de cette différence de destinataires de la sanction pénale et évidemment d'objet, peu de nuances entre ces deux autorités administratives. L'ARJEL est donc une autorité administrative indépendante, disposant d'un pouvoir de sanction – le blocage des sites non labellisés – et d'un pouvoir de régulation – l'agrément des sites de jeux en ligne, qui ne s'obtient qu'à partir du moment où le site fournit l'ensemble des informations la concernant, y compris les informations complètes relatives aux joueurs.

Curieusement, il n'y a pas eu autant d'indignation qu'au moment des discussions sur la loi HADOPI 1 & 2, voire pas du tout alors même que les pouvoirs dont disposent l'ARJEL sont très intrusifs.

De la même manière, lors des débats sur HADOPI 1 & 2, nous avions assisté à des mises en accusation de lobbys de l'industrie musicale, qui avaient fait pression sur le législateur. Dans le cas de l'ARJEL, le député Gaëtan Gorce parle à plusieurs reprises de lobbys *« alors que nous n'avons face à nous que des lobbies financiers [...] Nous sommes face à des lobbies organisés [...] La pression des lobbies, on le sait fort bien, est quasi quotidienne. ».*

Comme pour la loi HADOPI 1, la loi sur la régulation du jeu en ligne a souffert d'une suspicion de protection d'intérêts personnels.

En mai 2010, l'autorité est en place et le collège rend ses premières décisions et homologations, dont une délivrance d'agrément à la société Partouche **[2]** en juin 2010.

2 Un fonctionnement obscur

Le postulat concernant le jeu en ligne est simple : si le site n'est pas labellisé ARJEL, il n'est pas légal et devrait donc être bloqué par les FAI. La question est la suivante : un internaute français peut-il tomber par hasard sur un site non labellisé et y avoir accès sans difficulté ?

La réponse est oui. Pour les besoins de la démonstration, j'ai déconnecté mon VPN, j'ai utilisé un moteur de recherche ayant pignon sur Web et j'ai cherché « poker ». Sur la première page de résultats, uniquement des sites labellisés, mais pas sur le panneau latéral droit avec les annonces qui me proposent deux sites non labellisés et accessibles sans difficulté technique. Dans la deuxième page de résultats, deux sites non labellisés, dont un site appartenant au groupe Partouche, qui a perdu son agrément après une affaire rocambolesque **[3]**.

L'ARJEL peut saisir le président du Tribunal de Grande Instance de Paris pour demander le blocage des sites de jeux illicites accessibles sur le territoire français, sans mise en cause préalable de l'opérateur du site. Dans les tiroirs de Bercy, une procédure simplifiée de blocage des sites de jeux sans agrément attend également son arrivée au Parlement.

De ce qui ressort des informations mises en ligne sur son site Web, l'ARJEL surveille en interne les différents sites de paris en ligne et autres officines de poker, avec mise en place d'architectures et de systèmes particuliers lors de grands événements sportifs comme les Jeux olympiques de Sotchi **[4]**. Au contraire de l'Hadopi, la procédure de surveillance n'est pas externalisée à un acteur privé qui rapporterait ensuite à l'institution les fruits de ces investigations. Mais faisant moins l'objet d'interrogations de la part des organisations citoyennes et des médias, les informations concernant le fonctionnement même de cette procédure de surveillance sont quasiment inexistantes, ce qui est curieux quand on sait à quel point les questions de blocage de sites peuvent déclencher des réactions épidermiques.

Néanmoins, dans le premier rapport d'activité de l'institution, on retrouve l'ensemble des informations que les sites de jeux doivent fournir pour obtenir l'agrément et donc l'ensemble des informations nécessaires pour s'inscrire sur un site « légal » :

- n° de l'opérateur (l'identifiant du site de jeux) ;
- date de l'événement ;
- n° de l'événement ;
- identification du joueur chez l'opérateur ;
- ID de session ;
- IP du joueur ;

- n° du coffre utilisé (dispositif de l'ARJEL dans les systèmes d'information des sites de jeux pour permettre à l'institution de consulter sur place ou à distance les informations) ;
- agréments concernés ;
- login du joueur ;
- pseudo du joueur ;
- nom du joueur ;
- prénom du joueur ;
- genre ;
- date de naissance ;
- ville de naissance ;
- département de naissance ;
- pays de naissance ;
- adresse ;
- code postal ;
- ville ;
- pays ;
- téléphone mobile ;
- adresse e-mail.

Il en ressort donc que pour être labellisés, les sites de jeux doivent accepter de mettre en place des mouchards de l'institution qui captent l'ensemble de ces informations, payer les différents droits fixes (de 5000€ à 40 000€ selon les cas) et vérifier l'identité des joueurs pour bloquer les personnes interdites de jeux, car tous les sites labellisés doivent mettre en place des moyens pour lutter contre « le jeu excessif ou pathologique ».

Le ministre du budget Christian Eckert, dans un communiqué en date du 27 avril 2015, a fait part de sa volonté d'inclure dans les projets de loi numérique d'Emmanuel Macron et d'Axelle Lemaire un amendement permettant à l'institution de surveiller les joueurs sur leurs habitudes de jeu **[5]** afin qu'ils ne sombrent pas dans l'addiction.

Outre le questionnement idéologique sur le rôle de l'État dans le comportement des individus, cette idée interroge sur son déploiement. Il semblerait que l'institution se bornera à mener une étude statistique, ce qui en soi n'est pas franchement dérangeant, mais que fera-t-elle des résultats de cette étude ?

Quel est l'avenir de l'institution ? Contrairement à l'Hadopi, elle n'est pas secouée par des querelles intestines qui grippent son fonctionnement, mais elle détient clairement des pouvoirs inquiétants, dont le champ pourrait être é tendu.

3 Un avenir sombre

En effet, le mercredi 28 octobre, l'institution a organisé le colloque « 2010 - 2015 - 2020 : la régulation des jeux en ligne en France ». Concernant spécifiquement le poker, quatre nouvelles variantes de jeu seraient autorisées (seulement deux le sont actuellement). Le ministre du budget – outre son amendement sur la détection des joueurs pathologiques – souhaiterait que les sites de jeu mettent en place des modérateurs afin de limiter le temps passé sur les sites. Autre point qui risque de faire bondir tous les fiscalistes : une révision à la baisse de l'assiette fiscale ou – en langage clair – une baisse des taxes appliquées aux sites de jeux en ligne. Néanmoins, le ministre du budget a fait part de son opposition sur le sujet, ce qui n'empêche pas de continuer à garder ce point en mémoire pour les prochains mois. Le véritable malaise se trouve dans les propositions d'Éric Woerth qui souhaiterait élargir les pouvoirs de régulation au tarot, à la belote et aux sites de *Fantasy League*. Les « Fantasy League » sont des ligues sportives virtuelles jouées sur Internet couvrant un large éventail de sports collectifs et individuels et se déclinent sous le format du football américain, du soccer, du base-ball, du basket-ball, etc. **[6]**. Or, ce marché n'est même pas émergeant en France, malgré les tentatives du PMU en la matière. Quant au président de Winamax, il a clairement fait savoir qu'il souhaitait que les jeux comme *Hearthstone: Heroes of Warcraft*, *League of Legends* et autres *Candy Crush* soient inclus dans la régulation opérée par l'ARJEL, point de vue initialement défendu par Alexandre Dreyfus, fondateur de Global Poker Index **[7]**.

Je laisse au lecteur le soin d'imaginer ce que pourrait donner une ARJEL deuxième génération appliquée aux jeux comme *Candy Crush* ou *WoW*. Mais les finances de l'État étant particulièrement dévastées, il n'est pas impossible que pour des raisons purement fiscales, des avancées législatives en la matière voient le jour avec à la clef, de nouveaux blocages de sites Web et l'obligation pour les joueurs en ligne de changer leurs habitudes.

Conclusion

Pour le moment, l'ARJEL ne fait « que » demander des blocages de sites Web, mais détient un grand nombre d'informations sur les joueurs français et les velléités du ministre du budget en matière de lutte contre le blanchiment d'argent font craindre autre chose : l'interconnexion entre le fichier des joueurs de sites légaux avec leurs établissements bancaires. Le site bancaire français est d'une telle opacité que rares sont les détenteurs de comptes à savoir ce que leur banquier sait d'eux. Dans le cas de la télémédecine et de la médecine connectée, on a souvent évoqué l'hypothèse des personnes qui se verraient appliquer des malus par leurs mutuelles en fonction de leurs dossiers médicaux. Rien n'empêche de spéculer sur la possibilité pour les banques de refuser l'ouverture d'un compte si la personne aime jouer sur Internet. Enfin, contrairement à l'Hadopi, l'ARJEL a les faveurs des politiques – peu importe leur parti d'adhésion – il conviendra donc de surveiller quelles seront les avancées législatives en la matière. ■

Références

[1] Lire le compte-rendu intégral ici : http://www.assemblee-nationale.fr/13/cri/2009-2010/20100005.asp#ANCR200900000038-00332

[2] Liste des décisions du collège de l'ARJEL (agréments et homologations) : http://www.arjel.fr/-Decisions-de-l-Arjel-2010-agrement-.html

[3] Mediapart révèle une enquête de l'ARJEL sur le groupe Partouche : http://www.poker-academie.com/poker-actu/marche-poker/legislation-poker/3961-scandale-poker-partouche-touche-le-fond.html

[4] Communiqué de presse sur l'accord de coopération entre le CIO et l'ARJEL : http://www.arjel.fr/IMG/pdf/20140207CP.pdf

[5] Voir le communiqué de presse : http://proxy-pubminefi.diffusion.finances.gouv.fr/pub/document/18/19128.pdf

[6] Les « Fantasy League », ce marché périphérique du sport aux enjeux colossaux : http://blog.lefigaro.fr/sport-business/2014/05/les-fantasy-league-ce-marche-peripherique-du-sport-aux-enjeux-colossaux.html

[7] Hearthstone and Video Games Are the Biggest Threat to Online Poker : http://www.huffingtonpost.co.uk/alexandre-dreyfus/video-games-online-poker_b_8285436.html

LA PROGRAMMATION ORIENTÉE OBJET POUR TOUS ?

Tristan Colombo

La Programmation Orientée Objet ou POO ? Nous l'utilisons pour la plupart tous les jours. Certains langages d'ailleurs ne permettent pas d'alternative : c'est de la POO ou rien ! Mais est-on obligé de maîtriser et d'utiliser le concept d'objet pour être un véritable développeur ?

Mots-clés : POO, Réflexion, Python, Apprentissage, Développement

Résumé

POO et programmation impérative : deux méthodes de programmation dont les tenants s'affrontent fréquemment. Dans cet article, je vous convie à une réflexion sur ces méthodes et, ô sacrilège, à la façon de réutiliser des objets de manière à faire de la programmation impérative.

Prenons deux langages à titre d'exemple : Java oblige le développeur à utiliser la programmation orientée objet alors que Python permet d'employer soit la POO, soit la programmation impérative. La question que l'on peut alors se poser est la suivante : est-ce qu'un développeur ne développant pas en objet est fatalement un mauvais développeur ? Cette question peut nous amener plus loin dans un contexte où l'on ne cesse de nous dire qu'il faut « enseigner le code » à l'école : les enfants doivent-ils commencer leur apprentissage par la POO ?

1 La POO en pratique

La POO est une programmation très agréable à pratiquer lorsqu'elle a été correctement assimilée. Par contre, force est de constater qu'elle n'a absolument rien d'intuitif... on peut même dire que pour certains elle est contre-intuitive ! Très tôt dans la scolarité, les fonctions sont définies comme des objets mathématiques pouvant accepter des paramètres et retournant un résultat. Lorsque l'on apprend une table de multiplication par `n`, il ne s'agit jamais que d'appliquer une fonction `table_n(x) = x * n`. Ainsi, pour la table de `5`, `2 * 5 = 10` provient de `table_5(2) = 2 * 5`. Bien entendu, les enfants ne vont pas le formaliser ainsi, mais il s'agit du calcul sous-jacent... et ce calcul n'est pas orienté objet. S'il fallait définir une classe `Table_5` contenant une méthode `get()` permettant d'obtenir une valeur de la table de `5`, `2 * 5` proviendrait de `table_5 = new Table_5()` puis `table_5.get(2)`... Cela n'a rien de « logique » et du coup je ne vois donc pas comment commencer à enseigner un langage orienté objet à un enfant ne sachant pas (ou peu) programmer.

Si nous nous plaçons du côté des développeurs, suivant les langages qu'ils emploient, certains n'ont jamais eu besoin de POO de toute leur carrière et d'autres n'en saisissent

pas l'intérêt. À partir du moment où l'on peut parvenir à ses fins en utilisant les deux styles de programmation, si le code est clair et bien écrit il n'y a pas à partir en guerre pour imposer un concept plutôt qu'un autre.

Mon opinion est forcément biaisée par la pratique de Python qui permet d'utiliser les deux styles de programmation, mais j'utilise indifféremment la programmation orientée objet ou impérative en fonction des projets. Il ne me viendrait pas à l'idée de me lancer dans la création d'une classe dans un programme qui ne nécessitera qu'une dizaine de lignes et inversement, pour un projet de plusieurs centaines de lignes de codes, je partirai dans une optique objet. Tous les langages ne permettent pas cette souplesse et les développeurs Java seront coincés dans leurs raisonnements objet alors que les développeurs C ne sortiront pas des programmes impératifs. Le « hello world » de Java est particulièrement ridicule :

```
01: class HelloWorld {
02:     public static void main(String[] args) {
03:         System.out.println("Hello world !");
04:     }
05: }
```

En programmation impérative, avec Python, on écrit :

```
01: print('Hello world !')
```

Ceci ne signifie pas que Python est supérieur à Java ou inversement, mais tout simplement que c'est au développeur d'adapter le langage et le type de programmation qu'il va employer en fonction de ses objectifs.

2 Masquer la POO et faire de l'impératif

Dans le cadre de la programmation d'un robot Lego Mindstorms EV3 en Python (après avoir passé la bête sous GNU/Linux grâce au projet **ev3dev [1]**), je me suis retrouvé confronté au problème suivant :

- un module **python-ev3 [2]** orienté objet vieillissant (certaines commandes ne fonctionnent plus) ;
- grâce à ev3dev, une communication avec les moteurs et sondes qui se fait via une écriture dans des fichiers de **/sys/class** (les corrections à apporter à python-ev3 portent donc essentiellement sur le nom des périphériques cibles ou les valeurs à y écrire) ;
- un enfant de sept ans (allez, presque huit), sachant programmer en Logo, bidouiller avec le « langage » graphique EV3, et souhaitant apprendre à programmer en Python (non, personne ne l'influence...) ;
- un papa qui veut trouver une solution qui lui prenne le moins de temps possible.

J'ai bien entendu voulu commencer par corriger python-ev3... mais il y a des choses qui ne me plaisent pas dans l'architecture et j'ai préféré une réécriture. La première version, ne portant que sur la commande des leds, a été directement écrite en programmation impérative puisque l'objectif était de s'abstraire de la POO. Je me suis rapidement rendu compte qu'instinctivement je me dirigeai quand même vers une approche objet qui serait de toute manière bien plus simple à maintenir. Mais le problème était que, justement, je ne voulais pas de programmation orientée objet ! « *On tourne en rond, merde ! On tourne en rond, merde ! ...* » **[3]**. Donc autant réécrire python-ev3 en POO et créer un module de *binding* permettant de masquer l'aspect objet. Je vous présente ma solution (triviale) pour contrôler les leds, la même philosophie s'appliquant ensuite à l'ensemble des moteurs et capteurs. Ce procédé est ensuite applicable pour permettre aux personnes plus à l'aise en programmation impérative d'utiliser du code objet.

2.1 Un aperçu du code initial

Avant toute chose, il peut être intéressant de voir le code développé dans le projet python-ev3. Pour cela, on va le cloner :

```
$ git clone https://github.com/topikachu/python-ev3.git
```

Le code principal se trouve dans le répertoire **python-ev3/ev3** et plus particulièrement dans le fichier **ev3dev.py**. Si vous aimez la programmation orientée objet, fermez les yeux vous risqueriez d'attraper une forte migraine : toutes les classes dans un seul fichier, des variables globales... bref, une horreur !

Inutile de tester le module en éclairant les leds du boîtier... ça ne fonctionne pas. Vu la simplicité du développement, autant développer notre propre module. Bien entendu, il s'agit là d'un cas particulier lié à ce module. Si vous souhaitez récupérer du code en POO pour l'utiliser en impératif cette étape sera le plus souvent inutile.

2.2 Notre module

Avant de commencer à développer, il faut savoir que le boîtier EV3 dispose de quatre leds : une led rouge et une led verte sur la gauche et une led rouge et une led verte sur la droite. Pour allumer une led, il faut transmettre une valeur comprise entre 0 et 255 au fichier de *device* correspondant **[4]**. Par exemple, pour allumer la led verte de gauche au maximum, il faut exécuter :

```
$ echo 255 > /sys/class/leds/ev3:left:green:ev3dev/brightness
```

Pour les leds de droite, il suffit de remplacer **left** par **right** et pour du rouge au lieu du vert ce sera **red** au lieu de **green**.

Pour la construction de notre module, nous allons utiliser une classe **EV3Py** qui sera la classe mère de toutes les autres. Elle stockera dans l'attribut **device** le chemin du fichier qu'il faut employer et la méthode **write** permettra d'y écrire des commandes passées en paramètre :

```
01: class EV3Py:
02:     def __init__(self, device):
03:         self.device = device
04:
05:     def write(self, commands):
06:         for cmd, value in commands:
07:             with open(cmd, 'w') as device:
08:                 device.write(str(value))
```

Nous pouvons ensuite nous attacher à définir la classe **EV3Py_LED** permettant de manipuler les leds :

```
01: from EV3Py import EV3Py
02:
03: class EV3Py_LED(EV3Py):
04:     '''
05:        Color are defined using code on the two leds
06:        on each side : (red, green)
07:     '''
08:     RED = (255, 0)
09:     GREEN = (0, 255)
10:     YELLOW = (25, 255)
11:     ORANGE = (255, 180)
12:     BLACK = (0, 0)
13:
14:     def __init__(self, color=(0, 0), side='left'):
15:         super().__init__('/sys/class/leds/ev3:')
16:         self.color = color
17:         self.side = side
18:
19:     def change_color(self, color):
20:         self.color = color
21:
22:     def change_side(self):
23:         if self.side == 'left':
24:             self.side = 'right'
25:         else:
26:             self.side = 'left'
27:
28:     def __send_value(self, value):
29:         cmd = self.device + self.side + ':'
30:         commands = []
31:
32:         for i, color in enumerate(('red', 'green')):
33:             commands.append((cmd + color + ':ev3dev/
brightness', value[i]))
34:
35:         super().write(commands)
36:
37:     def on(self):
38:         self.__send_value(self.color[i])
39:
40:     def off(self):
41:         self.__send_value(EV3Py_LED.BLACK[i])
```

Je me suis focalisé sur les actions principales :

- **__init__()** dans les lignes 14 à 17 : constructeur de **EV3Py_LED** faisant appel au constructeur de la classe mère **EV3Py** et définissant les attributs **color** (pour la couleur) et **side** (pour le côté de la led) ;
- **change_color()** dans les lignes 19 et 20 : modifie la valeur de l'attribut **color**. Les couleurs sont représentées sous la forme de tuples de deux entiers indiquant la composition en rouge et en vert (valeurs entre **0** et **255**) ;
- **change_side()** dans les lignes 22 à 26 : change le côté de la led que nous manipulons. Pour simplifier, on considère qu'il n'y a qu'une seule led même si, techniquement, nous savons que les couleurs sont gérées à partir de la composition des deux leds rouge et verte ;
- **__send_value()** dans les lignes 28 à 35 : il s'agit d'une méthode privée permettant de factoriser le code d'envoi des commandes à exécuter. La méthode **write()** de la classe mère recevra une liste de tuples composés du chemin du *device* sur lequel écrire et de la valeur à écrire ;
- **on()** et **off()** dans les lignes 37 et 38 puis 40 et 41 : méthodes faisant appel à **__send_value()** pour allumer ou éteindre les leds.

On peut maintenant tester cette classe dans un shell Python sur l'EV3 :

```
johan@ev3dev$ python3
> from EV3Py_LED import EV3Py_LED
> led = EV3PY_LED()
> led.change_color(EV3Py_LED.YELLOW)
> led.on()
```

Tout fonctionne correctement comme le montre la figure 1.

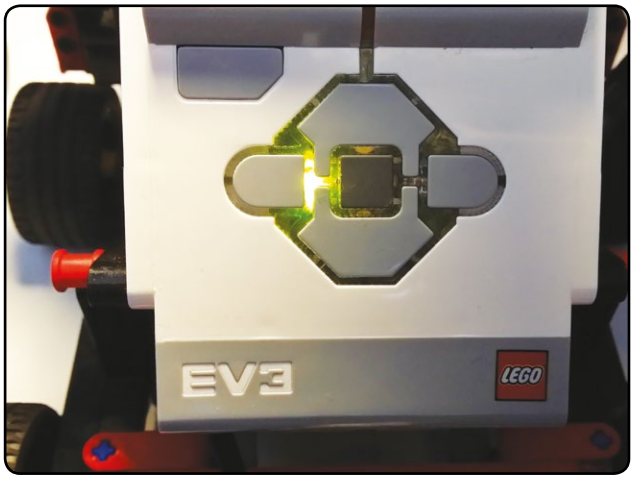


Fig. 1 : Éclairage des leds rouge et verte de gauche de manière à produire une lumière jaune (ça ne se voit pas sur la photo, mais pour un œil humain c'est bien du jaune...).

On peut donc passer à la partie permettant de masquer la POO.

2.3 Binding impératif

Pour simplifier les accès aux attributs statiques, nous allons créer une fonction **`get_colors()`** renvoyant un dictionnaire. Ceci permet de s'affranchir de variables globales du type **`RED = EV3Py_LED.RED`**. Les autres méthodes prennent essentiellement en paramètre l'objet et appellent sur celui-ci une méthode définie précédemment. Le code suivant est placé dans un fichier **`LED.py`** :

```
01: from EV3Py_LED import EV3Py_LED
02:
03: def get_colors():
04:     return {
05:         'red': EV3Py_LED.RED,
06:         'green': EV3Py_LED.GREEN,
07:         'yellow': EV3Py_LED.YELLOW,
08:         'orange': EV3Py_LED.ORANGE,
09:     }
10:
11: def LED(s='left'):
12:     return EV3Py_LED(side=s)
13:
14: def change_color(led, color):
15:     led.change_color(led)
16:
17: def on(led):
18:     led.on()
19:
20: def off(led):
21:     led.off()
```

Notre code de test précédent devient :

```
johan@ev3dev$ python3
> from LED import *
> color = get_colors()
> led = LED('left')
> change_color(led, color['yellow'])
> on(led)
```

Cela ne change pas grand-chose a priori pour quelqu'un maîtrisant la POO et il sera même tenté de dire qu'il est parfaitement ridicule d'écrire ce code qui consiste en fait à passer l'objet en paramètre... Pourtant au niveau de la compréhension ça change tout : on retrouve une fonction classique avec des paramètres et une valeur de retour. L'utilisation de Python dans ces exemples permet d'autant plus de montrer l'aspect impératif derrière la programmation orientée objet puisque **`self`** doit être passé en paramètre de toutes les méthodes.

Au final, le *binding* POO-impératif n'est guère complexe.

Conclusion

Un peu comme avec la guéguerre **vi** contre **emacs**, on rencontre beaucoup de partisans du « tout objet ». Pourtant, dans un monde manquant cruellement de tolérance, il n'y a aucune honte à développer de manière impérative. Il est même possible d'aider les développeurs ne maîtrisant pas l'objet ou les plus jeunes souhaitant apprendre à programmer (et non pas à réaliser des puzzles graphiques **[5] [6]**) en écrivant de petits *bindings* permettant un usage de la POO dans un code impératif... ■

Références

[1] Projet ev3dev : http://www.ev3dev.org/

[2] Module python-ev3 : http://www.ev3dev.org/

[3] *« Le grand blond avec une chaussure noire »*, Gaumont, 1972 : https://www.youtube.com/watch?v=4JngFSWyRLU

[4] Utilisation des leds avec ev3dev : https://github.com/ev3dev/ev3dev/wiki/Using-the-LEDs

[5] Le langage Scratch : https://scratch.mit.edu/

[6] Programmation Lego Mindstorms EV3 : http://www.lego.com/fr-fr/mindstorms/learn-to-program

RÉALISEZ LES SAUVEGARDES DE VOTRE TÉLÉPHONE ANDROID AVEC SSH

Frédéric Le Roy [Ingénieur informaticien, touche à tout et bidouilleur]

Votre smartphone, il ne vous quitte presque jamais : il contient votre vie numérique, vos photos, etc. Si vous faites partie d'une des sectes cloudistes, il est probable qu'une image de tout son contenu existe sur le serveur d'une ou plusieurs sociétés (suivez mon regard). Mais dans le cas contraire, comment réalisez-vous vos sauvegardes ?

sauvegarde Android rsync SSH

L'OBJECTIF

Nous allons voir comment j'ai mis en place la sauvegarde automatique d'une partie du contenu de mon téléphone, à savoir les fichiers comme des photos et des prises de notes. Étant fainéant de nature, le caractère automatique était indispensable !

LES OUTILS

- un système GNU/Linux Debian (version testing) ;
- `aptitude` (paquet `aptitude`) ;
- `vim` (paquet `vim`) ou tout autre éditeur de code ;
- `ssh` en tant que client ;
- `sshpass` pour gérer le mot de passe du client `ssh` ;
- `rsync` pour la sauvegarde.

PHASE 1 Installons un serveur ssh sur le téléphone

Tout d'abord, il nous faut un serveur ssh sur le téléphone ! Comme j'étais radin ce jour-là, j'ai choisi (après des tests quand même) **SSHDroid**. Je vous passe l'installation d'une application, vous devriez y arriver.

Lancez l'application, elle va directement lancer une instance du serveur ssh.

Fig. 1 : SSHDroid

Vous pouvez commencer par cliquer sur le bouton ***Arrêter*** en haut à droite. Maintenant allez dans le menu de gauche, puis ***Options*** et modifiez les paramètres suivants :

- Cochez ***Démarrer au lancement*** ;
- Décochez ***Écran actif*** ;
- Cochez ***Démarrer auto. en wifi*** ;
- Allez sur ***Mot de passe*** et modifiez le mot de passe par un mot de passe que vous n'utilisez nulle part ailleurs ! Ce mot de passe peut être lisible depuis l'application, mais nous devrons aussi le mettre en clair dans un script plus tard.

Et c'est tout. Cette configuration permet de n'avoir le serveur ssh actif que lorsque vous êtes connecté à un wifi. La version payante permet d'avoir une liste blanche des wifi à autoriser.

J'entends quelqu'un qui se demande où il va ajouter ses clés ssh. Et bien nulle part ! Seule la version payante le permet. Et nous allons voir plus loin comment nous en passer.

Relancer le service ssh en cliquant sur le bouton ***Démarrer***.

Pour la suite de l'article, la connexion ssh que j'ai configurée est la suivante :

- ip : **192.168.1.199** ;
- port : **2222** ;
- login : **root** (même si l'appareil n'est pas rooté...) ;
- mot de passe : **topsecret**.

Faisons un test :

```
$ ssh root@192.168.1.199 -p 2222
SSHDroid
Use 'root' as username
Default password is 'admin'
root@192.168.1.199's password:
u0_a266@t03g:/data/data/berserker.android.apps.sshdroid/home $
```

La connexion fonctionne.

PHASE 2 Se connecter sans saisir manuellement le mot de passe

Avant de continuer, il faut vous poser la question de la sécurité :

- le mot de passe sera par la suite transmis en clair dans une ligne de commandes. Une simple commande **ps** sur le serveur de sauvegarde permettra d'intercepter ce mot de passe pendant la durée de la session ssh. J'ai pour ma part considéré ce risque comme minime, mon réseau local étant théoriquement protégé et de toute façon si mon serveur est compromis (condition nécessaire pour faire un **ps**), j'aurais d'autres problèmes plus importants que l'accès à mon téléphone !
- pensez à bien utiliser un mot de passe que vous n'utilisez pas ailleurs de par le fait que si on vous vole le téléphone, le mot de passe peut malheureusement être lu dans les options de l'application.

Le client ssh classique ne permet pas, pour des raisons de sécurité, de saisir le mot de passe en paramètre de la commande. Il ne permet pas non plus d'utiliser **stdin**. Il existe toutefois une application qui permet de faire croire au client ssh que le mot de passe est bien saisi de manière classique : **sshpass**. Installons-la :

```
# apt-get install sshpass
```

Nous allons utiliser cet outil ainsi : **sshpass -p 'motdepasse' <commande ssh>**. Testons :

```
$ sshpass -p topsecret ssh root@192.168.1.199 -p 2222
SSHDroid
Use 'root' as username
Default password is 'admin'
u0_a266@t03g:/data/data/berserker.android.apps.sshdroid/home $
```

Ça fonctionne.

PHASE 3 Identifier les fichiers et dossiers à sauvegarder

Pour les besoins de l'article, nous allons sauvegarder un unique dossier, mais il vous suffira d'adapter le script final pour sauvegarder autant de dossiers/fichiers que vous le souhaitez.

Vous pouvez utiliser la console ssh pour retrouver le chemin vers vos fichiers. Par exemple, sur mon téléphone, mes photos sont dans **/storage/sdcard0/DCIM/Camera/**.

Choisissez également un dossier cible pour y déposer la sauvegarde.

PHASE 4 Test d'une sauvegarde à la main

Pour réaliser les sauvegardes, nous allons utiliser l'outil **rsync**.

```
# apt-get install rsync
```

Pour faire une copie via ssh, il est possible de passer à rsync l'option **-e "ssh"**. Voici la commande à exécuter pour le test :

```
$ rsync -e "ssh -p 2222" -avzn root@192.168.1.199:/storage/
sdcard0/DCIM/Camera/ /tmp/
SSHDroid
Use 'root' as username
Default password is 'admin'
root@192.168.1.199's password:
receiving incremental file list
./
2015-09-10 09.28.17.jpg
20150801_172241.jpg
20150801_172251.jpg
20150801_174241.jpg
20150801_182953.jpg
20150801_202602.jpg
[...]
20150915_163209.jpg
20150915_163336.jpg

sent 794 bytes  received 5,242 bytes  447.11 bytes/sec
total size is 3,023,856,484  speedup is 500,970.26 (DRY RUN)
```

Notez tout d'abord l'option **n** qui permet de lancer la commande en mode simulation (*DRY RUN*). Tant que le script final n'est pas prêt, il est conseillé de garder cette option !

L'option **-e** voit ici un paramètre apparaître pour ssh : le port du serveur ssh du téléphone. Le reste est très classique :

- **v** : mode verbeux ;
- **a** : mode archivage. Il s'agit d'un raccourci pour activer plusieurs options dont la copie récursive, la préservation des droits, des dates/heures, etc. ;
- **z** : compresse les données pendant le transfert.

Avec cette commande nous avons dû saisir le mot de passe. Pour ne plus avoir à le saisir, il va falloir ajouter **sshpass** non pas dans l'option **e**, mais avant la commande **rsync** :

```
sshpass -p  topsecret rsync -e "ssh -p 2222" -avzn
root@192.168.1.199:/storage/sdcard0/DCIM/Camera/ /tmp/
```

PHASE 5 Scriptons ça !

Voici un exemple de script, nommé **sync_android_fred.sh**, qui va simplifier le portage de la commande :

```
#!/bin/bash
# Stockage de la sauvegarde
LOCAL_DIR=/tmp/photos

# Telephone
REMOTE_IP=192.168.1.199
REMOTE_PORT=2222
REMOTE_LOGIN=root
REMOTE_PASSWORD=topsecret
REMOTE_DIR=/storage/sdcard0/DCIM/Camera/

# Mode test
# vide : pas de simulation
# TEST=n : simulation
TEST=n

# On tente de creer le dossier local
mkdir -p $LOCAL_DIR
if [ $? -ne 0 ] ; then
    echo "Erreur a la creation du dossier : $LOCAL_DIR"
    exit 1
fi

# Sauvegarde
sshpass -p $REMOTE_PASSWORD rsync -e "ssh -p $REMOTE_PORT"
-avz$TEST $REMOTE_LOGIN@$REMOTE_IP:$REMOTE_DIR $LOCAL_DIR
```

Testons-le :

```
$ ./sync_android_fred.sh
SSHDroid
Use 'root' as username
Default password is 'admin'
receiving incremental file list
./
2015-09-10 09.28.17.jpg
20150801_172241.jpg
[...]
20150915_163209.jpg
20150915_163336.jpg

sent 794 bytes  received 5,242 bytes  2,414.40 bytes/sec
total size is 3,023,856,484  speedup is 500,970.26 (DRY RUN)
```

PHASE 6 Planification

Il ne reste plus qu'à planifier le lancement du script quand vous dormez en ajoutant par exemple une règle crontab, ici à 2 heures du matin tous les jours :

```
0 2 * * * /media/stock/sauvegardes/sync_android_fred.sh
```

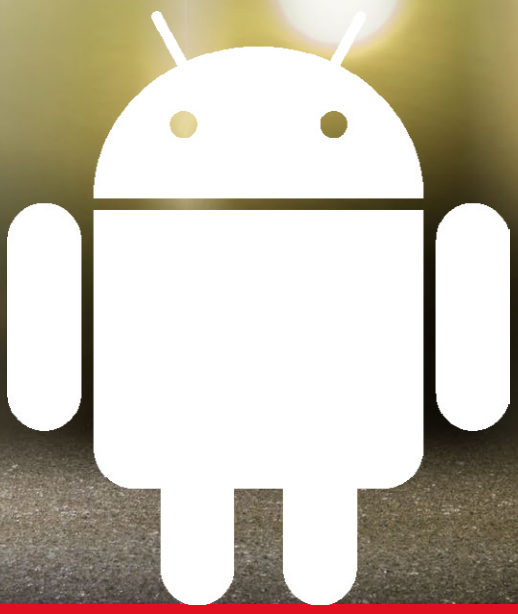
LE RÉSULTAT

Et voilà, mes photos (entre autres) sont sauvegardées de manière automatique. Je conseille toutefois à chacun de personnaliser les options de rsync en fonction des besoins (suppression dans la sauvegarde des fichiers supprimés sur le téléphone ou pas, etc). Vous pouvez aussi créer un script en remplacement de la règle crontab qui attendra que votre téléphone soit présent sur le réseau local (via un ping toutes les minutes par exemple) pour lancer une sauvegarde. ■

ACTUELLEMENT DISPONIBLE

GNU/LINUX MAGAZINE HORS-SÉRIE N°82 !

LE GUIDE POUR
APPRENDRE À DÉVELOPPER DES APPLICATIONS ANDROID



NE LE MANQUEZ PAS

CHEZ VOTRE MARCHAND DE JOURNAUX ET SUR :

www.ed-diamond.com

SAUVEGARDE AUTOMATISÉE DE SES DONNÉES PERSONNELLES

Hervé [Passionné par l'open source depuis sa rencontre avec Tux]

Il y a deux types de personnes dans le monde : ceux qui ont déjà perdu des données et ceux qui vont bientôt en perdre. Comme beaucoup de monde, j'avais déjà entendu cette maxime à plusieurs reprises, mais j'étais persuadé qu'elle ne s'appliquait qu'aux autres...

Mots-clés : Sauvegarde, LVM, Systemd, Udev

Résumé

Dans cet article, je vais vous présenter comment utiliser LVM pour sauvegarder instantanément l'ensemble de vos données en n'allouant que l'espace disque nécessaire. Ensuite, nous verrons comment tirer parti de systemd/udev pour implémenter un mécanisme de sauvegarde externe se déclenchant automatiquement au branchement d'un disque dur USB.

Un jour, alors que je comptais formater ma clé USB, je me suis retrouvé après un moment d'inattention en train de formater mon disque dur principal. C'est par cette stupide erreur humaine que j'ai failli entrer au club de *ceux qui ont déjà perdu des données*. Cette subite prise conscience de la valeur de mes données m'a alors convaincu qu'il fallait que je me protège de moi-même (!), mais également de problèmes matériels. En d'autres termes, j'avais besoin d'automatiser des sauvegardes régulières et de trouver par ailleurs un moyen d'externaliser mes sauvegardes.

1 Sauvegardes complètes en local avec LVM

Il existe principalement deux types de sauvegardes locales :

- copie des données vers un autre dossier : malgré un grand nombre d'outils optimisés pour permettre des sauvegardes incrémentales, cela reste une activité très intensive pour le disque et ne passe pas inaperçu pour l'utilisateur ;
- snapshot d'un volume : cette méthode fournit une copie conforme au volume source correspondant au moment où le snapshot est créé. Un snapshot va enregistrer toutes les modifications qui divergent entre lui-même et le volume source, les données identiques pointant directement sur le volume source. Hormis le fait que cette technique est totalement transparente pour l'utilisateur, elle nous garantit également que les données sont consistantes, c'est-à-dire que l'ensemble des données est précisément dans l'état où elles étaient au moment de la création du snapshot. Ceci nous met à l'abri des risques de corruption (en particulier sur des bases de données).

La fonctionnalité permettant de créer des snapshots LVM est connue de tous, mais elle présente un inconvénient majeur : il est nécessaire d'indiquer à la création du snapshot la taille qui va lui être immédiatement allouée. Bien qu'il soit possible de réajuster la taille du snapshot plus tard, il s'agit d'une manipulation considérée comme risquée, c'est donc pourquoi il faut être capable d'estimer l'espace disque nécessaire, sachant que la taille du snapshot est proportionnelle aux données modifiées. Ainsi si le snapshot est trop petit pour stocker les données divergentes, il devient invalide et n'est plus utilisable. A contrario, un snapshot trop grand va engendrer un gaspillage d'espace disque.

LVM

LVM pour *Logical Volume Manager* est un gestionnaire de volumes logiques. Cet outil apporte une grande souplesse dans la gestion des espaces de stockage et offre des fonctionnalités intéressantes (agrandissement ou réduction à chaud, snapshot, *thin-provisionning*, RAID). Actuellement très en vogue, son avenir est menacé par le système de fichiers prometteur BRTFS qui inclut nativement la plupart des fonctionnalités apportées par LVM.

1.1 Thin-provisionning avec LVM

LVM fournit une fonctionnalité peu répandue offrant du *thin-provisionning*, c'est-à-dire d'allocation à la demande d'espace, ce qui permet de mettre en œuvre une politique de sur-allocation. Dénommée LVM thin, cette technologie est présente dans Debian depuis la version 2.02.98-1 de **lvm2**, soit à partir de Debian Jessie.

La sur-allocation d'espace disque peut être rebutante de prime abord, il est indéniable que cela demande une gestion rigoureuse, mais en y réfléchissant, cette politique est déjà appliquée sur les autres ressources matérielles : le CPU et la mémoire. Par exemple, sur votre système vous avez probablement laissé le paramètre **overcommit_memory** à sa valeur par défaut, autorisant ainsi Linux à sur-allouer de la RAM :

```
$ sudo sysctl vm.overcommit_memory
vm.overcommit_memory = 0
```

Note

J'ai un jour essayé de vivre sans sur-allocation mémoire (**vm.overcommit_memory = 2**) : j'ai vite abandonné, car au bout de quelques jours je ne pouvais plus lancer Firefox suite à des refus d'allocation mémoire. Malgré pas mal de RAM libre, beaucoup d'applications comme Firefox allouent en effet d'importantes quantités de mémoire sans l'utiliser en totalité. Lorsque Linux a alloué trop de RAM et que la quantité réellement disponible devient faible, c'est alors qu'intervient le fameux *Out Of Memory Killer*.

LVM thin ajoute une nouvelle couche d'abstraction matérialisée par un *pool* : il s'agit d'un volume qui va servir uniquement à contenir d'autres volumes, appelés « thin volumes ». Les thin volumes vont puiser dynamiquement dans le pool l'espace disque nécessaire, ainsi un volume vide ne consommera pas d'espace... mais attention, s'il n'y a plus d'espace libre dans le *pool*, les thin volumes deviennent tous invalides. La commande **lvs** permet de connaître le niveau de remplissage (colonnes **Data%** pour les données et **Meta%** pour les métadonnées, aucune de ces deux valeurs ne doit valoir **100%**) :

```
$ sudo lvs -S segtype='thin-pool'
  LV       VG         Attr       LSize   Pool Origin Data%  Meta%  Move Log Cpy%Sync Convert
  THIN_POOL ARIANE_VG1 twi-a-t--- 147,23g             55,71  43,08
```

Heureusement, à l'instar des snapshots classiques, LVM est capable de surveiller les thin pool et de les étendre si nécessaire :

```
$ grep "thin_pool_autoextend.*=" /etc/lvm/lvm.conf
    thin_pool_autoextend_threshold = 80
    thin_pool_autoextend_percent = 20
```

Ici j'indique à LVM d'étendre automatiquement le *pool* lorsque son utilisation dépasse 80% de sa taille et de lui ajouter 20% de sa taille (puisé dans l'espace libre du volume **group**).

Pour utiliser **lvmthin**, il faut d'abord créer un *pool* en indiquant sa taille maximale :

```
# lvcreate -L100G -T ARIANE_VG1/THIN_POOL
```

Puis, la création d'un thin volume se fait en appelant l'option **-V** (*Virtual size*) qui représente une limite pour le volume, mais ne provoque aucune allocation d'espace (ce qui permet donc de sur-allouer) :

```
# sudo lvcreate -V 100T -T ARIANE_VG1/THIN_POOL
```

Par ailleurs, il est important de garder à l'esprit que les données supprimées ne sont pas automatiquement nettoyées du thin pool. Pour cela, il faut envoyer une commande **TRIM**, indépendamment de l'utilisation d'un disque SSD, pour informer le *pool* des blocs à supprimer :

```
$ df -h .
Filesystem                   Size  Used Avail Use% Mounted on
/dev/mapper/ARIANE_VG1-TV_HOME   79G   64G   16G  81% /home
dud@ariane:~$ sudo lvs -S lv_name='TV_HOME'
  LV      VG         Attr       LSize  Pool      Origin Data%  Meta%  Move Log Cpy%Sync Convert
  TV_HOME ARIANE_VG1 Vwi-aotz-- 80,00g THIN_POOL        81,31


$ dd if=/dev/zero of=zero.img bs=1M count=1024
1024+0 records in
1024+0 records out
1073741824 bytes (1,1 GB) copied, 2,94005 s, 365 MB/s
$ df -h .
Filesystem                   Size  Used Avail Use% Mounted on
/dev/mapper/ARIANE_VG1-TV_HOME   79G   65G   15G  82% /home
dud@ariane:~$ sudo lvs -S lv_name='TV_HOME'
  LV      VG         Attr       LSize  Pool      Origin Data%  Meta%  Move Log Cpy%Sync Convert
  TV_HOME ARIANE_VG1 Vwi-aotz-- 80,00g THIN_POOL        82,22


$ rm zero.img
$ df -h .
Filesystem                   Size  Used Avail Use% Mounted on
/dev/mapper/ARIANE_VG1-TV_HOME   79G   64G   16G  81% /home
$ sudo lvs -S lv_name='TV_HOME'
  LV      VG         Attr       LSize  Pool      Origin Data%  Meta%  Move Log Cpy%Sync Convert
  TV_HOME ARIANE_VG1 Vwi-aotz-- 80,00g THIN_POOL        82,56


$ sudo fstrim /home
$ sudo lvs -S lv_name='TV_HOME'
  LV      VG         Attr       LSize  Pool      Origin Data%  Meta%  Move Log Cpy%Sync Convert
  TV_HOME ARIANE_VG1 Vwi-aotz-- 80,00g THIN_POOL        81,31
```

On s'aperçoit que même la suppression du fichier augmente l'espace consommé du thin volume (colonne **%Data**). Après avoir envoyé la commande **TRIM**, on revient à l'état nominal.

Note

Pour planifier l'envoi régulier de commandes **TRIM**, je vous suggère de récupérer le *timer* et le service *systemd* installés dans **/usr/share/doc/util-linux/examples/**.

1.2 Comparatif entre LVM « classique » et LVM thin

En plus des éléments cités précédemment, il faut savoir qu'avec LVM « classique », les snapshots ont un impact très important sur les performances : lorsqu'un snapshot existe et que le volume source est modifié, LVM doit d'abord sauvegarder la donnée originale dans le snapshot avant de pouvoir enregistrer la modification dans le volume source. En d'autres termes, il s'agit d'une sauvegarde à la demande, le snapshot pointant directement sur le volume source pour toutes les données non modifiées. À chaque modification, il y a autant de copies à faire qu'il existe de snapshots. Avec LVM thin, ce comportement est atténué, car l'ensemble des données des thin volumes est centralisé dans le pool. À chaque modification, LVM thin va copier la donnée originale dans un nouvel emplacement, laissant ainsi l'emplacement initial et tous les thin volumes (ou snapshots) pointant sur cet endroit tels quels. Le coût d'une modification est constant, quel que soit le nombre de snapshots. Par ailleurs, LVM thin est censé être plus performant.

L'espace disque d'un volume « classique » va être alloué immédiatement, ce qui non seulement garantit la disponibilité du stockage, mais contribue également à disposer d'un espace contigu. En revanche, avec LVM thin les emplacements sont alloués au fur et à mesure, ce qui augmente la probabilité d'un volume fragmenté, ce n'est donc pas optimal si vous disposez d'un vieux disque dur magnétique. Comme dit précédemment, il est important de bien surveiller l'utilisation du *pool* pour éviter qu'il ne se remplisse.

En résumé, voici un tableau comparatif entre les deux technologies :

VOICI TOUTES LES OFFRES COUPLÉES AVEC GNU/LINUX MAGAZINE !

POUR LE PARTICULIER ET LE PROFESSIONNEL ...

Prix TTC en Euros / France Métropolitaine

CHOISISSEZ VOTRE OFFRE !

SUPPORT

Prix en Euros / France Métropolitaine

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

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

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

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

	LVM « classique »	LVM thin
Avantages	▪ espace disque alloué garanti ▪ performant en cas d'absence de snapshot, car espace disque contigu (en particulier pour HDD) ▪ récupération possible d'un volume effacé récemment (commande **vgcfgrestore**)	▪ plus performant et pas de dégradation de performance avec des snapshots ▪ n'utilise que l'espace réellement consommé ▪ par défaut, efface préalablement les données existantes
Inconvénients	▪ nécessite de bien prévoir la taille des volumes sous peine de gaspillage d'espace ▪ performances dégradées en cas de présence de snapshots	▪ impose une gestion rigoureuse du *pool* (auto-extension) ▪ libération d'espace disque effectif après envoi de la commande **TRIM** ▪ fragmentation importante, performances dégradées sur HDD

Je vous suggère de lire l'excellent article écrit par G. Danti pour plus de détails **[1]**.

Vu que nous souhaitons conserver plusieurs snapshots (pour ma part cela doit couvrir une semaine) et limiter l'espace disque nécessaire tout en évitant tout impact négatif sur les performances au quotidien, LVM thin est adapté pour ce besoin.

1.3 Automatisation des sauvegardes en local

Il ne reste plus qu'à automatiser la création de snapshots. Pour cela, je vous propose le script suivant à placer par exemple dans **/usr/local/bin/LVM_backup.sh** :

```
#!/bin/bash -eu

VOLS_TO_BACKUP=("ARIANE_VG1/TV_ROOT_JESSIE" "ARIANE_VG1/TV_HOME")

day=$(date +%u)
for vol in "${VOLS_TO_BACKUP[@]}"
do
  snap=$(echo "${vol}" | sed -r "s|([^/]*)/(.*)|\1/SNAP-${day}_\2|")
  lvcreate_opts=
  if [ $(lvs -o layout --noheadings "${vol}") = "thin,sparse" ]
  then
    lvcreate_opts=
  elif [ $(lvs -o layout --noheadings "${vol}") = "linear" ]
  then
    lvcreate_opts="-l 50%ORIGIN"
  fi
  (set +e ; lvs "${snap}" && sudo lvremove -f "${snap}" ; exit 0) &>
/dev/null
  lvcreate -s -n "${snap}" ${lvcreate_opts} "${vol}" &> /dev/null
done
```

Il vous suffit simplement d'adapter la valeur de la variable **VOLS_TO_BACKUP**. Notez que le script gère également les volumes « classiques » en réservant la moitié de la taille du volume source pour le snapshot.

Note

D'après le manuel de **lvcreate**, 15 à 20% de la taille du volume source est généralement suffisante pour un snapshot.

Et ensuite, il ne reste plus qu'à planifier ce script :

▪ Dans le fichier **/etc/systemd/system/backup.service** :

```
[Unit]
Description=Backup LVM devices

[Service]
Type=oneshot
ExecStart=/usr/local/bin/LVM_backup.sh
```

▪ Dans le fichier **/etc/systemd/system/backup.timer** :

```
[Unit]
Description=Backup LVM devices daily

[Timer]
OnCalendar=daily
Persistent=true

[Install]
WantedBy=multi-user.target
```

Enfin, le *timer systemd* est activé avec la commande **sudo systemctl enable backup.timer**.

Mais pourquoi s'embêter avec systemd, que nous avons déjà prévu d'utiliser dans la deuxième partie de l'article, et ne pas employer une simple crontab ? Je dois avouer que *l'avantage principal est quand même de faire rager les anti-systemd en le mettant partout et à toutes les sauces* **[2].**

Plus sérieusement, après avoir utilisé les deux, je préfère systemd, car on peut facilement obtenir des informations sur la planification des timers à l'aide de la commande **systemctl list-timers**. Sur un ordinateur non allumé en permanence, on est obligé de s'appuyer sur **anacrontab**, qui ajoute des temporisations pour éviter que tout ne s'exécute en même temps, et malheureusement j'ai déjà constaté que des exécutions de sauvegardes programmées par **cron.daily** pouvaient passer à la trappe. systemd est quant à lui beaucoup plus agressif dans la planification des *timers* (option paramétrable, par défaut 1min).

2 Sauvegardes externes déclenchées à la demande

On rencontre habituellement deux méthodes de sauvegarde externe chez un particulier : le stockage sur un serveur NAS, ce qui nécessite un petit investissement financier, ou dans le « cloud », ce qui implique de donner sa confiance à l'hébergeur ainsi que de disposer d'une bonne connexion Internet (oui, oui, à une dizaine de kilomètres de la rédaction de *GNU/Linux Magazine*, un village a eu l'honneur de découvrir l'ADSL il y a à peine quelques mois...). Partisan de solutions économiques, je me suis contenté de simplement réutiliser un ancien disque dur que j'ai branché dans un boîtier externe USB et de tirer parti de systemd/udev pour automatiser le déclenchement du processus de sauvegarde.

2.1 udev : détection du branchement du disque externe

Udev est le nom du système chargé de détecter toute nouvelle apparition de composant matériel et de déterminer le module noyau à utiliser. C'est également lui qui gère les *devices* apparaissant dans **/dev**. Pour voir comment cela fonctionne, vous pouvez essayer la commande **udevadm monitor** :

```
$ udevadm monitor
monitor will print the received events for:
UDEV - the event which udev sends out after rule processing
KERNEL - the kernel uevent

KERNEL[37396.709332] remove   /devices/pci0000:00/0000:00:1d.0/
usb2/2-1/2-1.2/2-1.2:1.0/0003:046D:C050.0003/input/input82/mouse0
(input)
UDEV  [37396.710437] remove   /devices/pci0000:00/0000:00:1d.0/
usb2/2-1/2-1.2/2-1.2:1.0/0003:046D:C050.0003/input/input82/mouse0
(input)
[...]
KERNEL[37399.059535] add      /devices/pci0000:00/0000:00:1d.0/
usb2/2-1/2-1.2/2-1.2:1.0/0003:046D:C050.0004/input/input83/mouse0
(input)
[...]
UDEV  [37399.081210] add      /devices/pci0000:00/0000:00:1d.0/
usb2/2-1/2-1.2/2-1.2:1.0/0003:046D:C050.0004/input/input83/mouse0
(input)
```

On voit clairement qu'udev se contente de reporter les événements générés par Linux.

Udev est simple à prendre en main : l'outil lit séquentiellement des règles contenant des critères de sélection et applique la première qui correspond. Les critères s'appuient sur des clés permettant d'exprimer différentes caractéristiques (type d'événement, nom du *device*, du driver, attributs sysfs, résultat de l'exécution d'un programme, variable d'environnement, etc.). Une fois qu'une règle est sélectionnée, udev peut modifier un certain nombre d'attributs (par exemple renommer ou changer les droits d'un *device*, etc.), lancer un programme ou déclencher le lancement d'un service systemd. Comme l'indique clairement le manuel, la clé **RUN**, permettant de lancer des programmes, n'est pas prévue pour exécuter des processus qui durent longtemps, comme des *daemons* par exemple. En effet, l'événement déclenchant l'exécution du programme indiqué par **RUN** bloquera tous les autres événements relatifs à ce *device* et tous les processus qui auront été lancés seront tués une fois l'événement traité. Nous allons donc utiliser un service systemd et tirer parti de sa capacité à gérer des dépendances.

La commande **udevadm info** permet d'obtenir des informations sur un *device* pouvant être utilisé dans les règles udev :

```
$ sudo udevadm info /dev/sdb
[...]
E: DEVNAME=/dev/sdb
E: DEVPATH=/devices/pci0000:00/0000:00:1d.0/usb2/2-1/2-1.3/2-
1.3:1.0/host21/target21:0:0/21:0:0:0/block/sdb
E: DEVTYPE=disk
E: ID_BUS=usb
E: ID_FS_TYPE=crypto_LUKS
```

```
E: ID_FS_USAGE=crypto
E: ID_FS_UUID=32b8e9f7-c92c-4120-bac2-c9c23b8af0ed
[...]
E: ID_MODEL=HTS542512K9SA00
[...]
E: ID_TYPE=disk
E: ID_USB_DRIVER=usb-storage
E: ID_USB_INTERFACES=:080650:
E: ID_USB_INTERFACE_NUM=00
E: ID_VENDOR=HITACHI
[...]
E: MAJOR=8
E: MINOR=16
E: SUBSYSTEM=block
[...]
```

L'option **-a** permet d'obtenir les attributs sysfs en détaillant toute l'arborescence udev :

```
$ udevadm info /dev/sdb -a
[...]
  looking at device '/devices/pci0000:00/0000:00:1d.0/usb2/2-1/2-
1.3/2-1.3:1.0/host21/target21:0:0/21:0:0:0/block/sdb':
    KERNEL=="sdb"
    SUBSYSTEM=="block"
    DRIVER==""
    ATTR{ro}=="0"
    ATTR{size}=="234441648"
[...]
  looking at parent device '/devices/pci0000:00/0000:00:1d.0/
usb2/2-1/2-1.3/2-1.3:1.0/host21/target21:0:0/21:0:0:0':
    KERNELS=="21:0:0:0"
    SUBSYSTEMS=="scsi"
    DRIVERS=="sd"
[...]
    ATTRS{model}=="HTS542512K9SA00 "
[...]
    ATTRS{vendor}=="HITACHI "
[...]
```

Comme vous pouvez le constater, nous avons l'embarras du choix pour exprimer nos critères de sélection.

À la lumière de ces informations, nous pouvons créer notre règle udev dans **/etc/udev/rules.d/90-external-backup.rules** :

```
ACTION=="add", SUBSYSTEM=="block", ATTRS{model}=="HTS542512K9
SA00", ENV{ID_FS_UUID}=="32b8e9f7-c92c-4120-bac2-c9c23b8af0ed",
ENV{SYSTEMD_WANTS}+="external-backup.service"
```

Lorsque udev détecte l'ajout d'un périphérique de type bloc dont le modèle est **HTS542512K9SA00** et qui héberge un système de fichiers dont l'uuid est celui indiqué, il lancera le service **external-backup.service**.

udev

udev est un gestionnaire de périphérique qui fonctionne à l'aide d'un système de règles. On peut définir des actions à effectuer lors de la survenue d'un événement (ajout ou suppression d'un périphérique). Le projet a fusionné avec systemd en 2012.

2.2 systemd : gestion des dépendances et lancement de la sauvegarde

Bien entendu, les données sont stockées chiffrées sur le disque externe. J'ai donc créé un container LUKS pouvant être ouvert grâce à un fichier stocké dans **/root/keyfile** ou un mot de passe (pour me permettre de récupérer mes données). L'utilisation d'un fichier de déchiffrement est nécessaire pour permettre une sauvegarde non interactive, par conséquent, il convient de bien protéger ce fichier.

Voici le paramétrage à effectuer dans **/etc/crypttab** pour déchiffrer automatiquement le disque externe à l'aide du fichier **keyfile** :

```
HTS542512K9SA00_120G UUID=32b8e9f7-c92c-4120-bac2-c9c23b8af0ed
/root/keyfile luks,noauto
```

Le volume déchiffré est ensuite monté sur **/mnt/EXTERNAL_DISK_BACKUP** dans **/etc/fstab** :

```
/dev/mapper/HTS542512K9SA00_120G/mnt/EXTERNAL_DISK_BACKUP ext4
noatime,nosuid,nodev,noauto  0 2
```

Enfin, il ne reste plus qu'à exécuter **systemctl daemon-reload** pour lancer les générateurs d'unité systemd et constater la création de la configuration correspondante (tout se trouve dans **/var/run/systemd/generator**) : **systemd-cryptsetup@HTS542512K9SA00_120G.service** pour le chiffrement et **mnt-EXTERNAL_DISK_BACKUP.mount** pour le point de montage. Les dépendances entre les deux services sont automatiquement configurées :

```
$ systemctl cat mnt-EXTERNAL_DISK_BACKUP.mount
[...]
[Mount]
What=/dev/mapper/HTS542512K9SA00_120G
[...]
$ systemctl cat systemd-cryptsetup@HTS542512K9SA00_120G.service
[...]
BindsTo=dev-mapper-%i.device
[...]
```

Il ne reste plus qu'à créer le service effectuant la sauvegarde à proprement parler dans **/etc/systemd/system/external-backup.service** :

```
[Unit]
Description=On-plug backup to external USB disk
BindsTo=mnt-EXTERNAL_DISK_BACKUP.mount
After=mnt-EXTERNAL_DISK_BACKUP.mount

[Service]
Type=oneshot
# Udisks2 ne parvient pas à ouvrir le volume chiffré et bloque à
100 % du CPU
ExecStartPre=/bin/systemctl restart udisks2.service
ExecStart=/bin/cp /home /mnt/EXTERNAL_BACKUP/ -rf
ExecStartPost=/bin/sync
ExecStartPost=/usr/bin/wall "Démarrage de la sauvegarde sur
disque externe"
ExecStop=/bin/systemctl start external-backup_cleanup.target
ExecStopPost=/usr/bin/wall "Sauvegarde sur disque externe
terminée"
```

La dépendance au point de montage **/dev/EXTERNAL_DISK_BACKUP**, et par extension au service de déchiffrement du disque externe, est indiquée dans la section **[Unit]**. Dans la section suivante, la commande exécutant la sauvegarde est indiquée. Suite à un bug que j'ai constaté sur Udisk2 (il monopolise 100% du CPU), je contourne le problème en redémarrant le service au préalable. Enfin, je prends soin de synchroniser le cache sur disque à la fin de la sauvegarde pour éviter toute corruption. De plus, l'avancement de la sauvegarde est indiqué à l'aide de la commande **wall** (j'étais parti sur un outil graphique **notify-send**, mais je n'ai pas réussi à l'utiliser dans ce cadre).

Une fois cette étape terminée, la cible systemd **external-backup_cleanup.target** dont le rôle est de démonter proprement le volume est appelée. Il n'est pas possible dans ce cas de définir des dépendances pour démonter le volume, car cela crée une boucle avec les dépendances que nous avons configurées précédemment pour monter le volume. La cible **external-backup_cleanup.target** va simplement servir à exprimer un conflit avec le service **systemd-cryptsetup@HTS542512K9SA00_120G.service**, ce qui va provoquer le démontage du volume ainsi que le rechiffrement du conteneur LUKS. Voici le contenu de **/etc/systemd/system/external-backup_cleanup.target** :

```
[Unit]
Description=Unmount external backup disk
Conflicts=systemd-cryptsetup@HTS542512K9SA00_120G.service
StopWhenUnneeded=yes
```

Le paramètre **StopWhenUnneeded** permet de repasser automatiquement la cible **external-backup_cleanup.target** à l'état **stopped** une fois son travail terminé (sinon elle reste dans l'état **started** en permanence, empêchant toute nouvelle sauvegarde de démonter le disque proprement).

Note

À partir de systemd v220 (Jessie est en v215, par contre Stretch a une version plus récente), le paramètre **TimeoutIdleSec** peut être utilisé pour démonter automatiquement un point de montage à partir d'un certain temps d'inactivité. Cette fonctionnalité pourrait rendre inutile l'utilisation de la cible **external-backup_cleanup.target** si le disque n'était pas chiffré.

Et voilà ce qu'il se passe lorsqu'on branche le disque :

```
$

Diffusion de message de root@ariane (somewhere) (Wed Oct 14
01:53:33 2015) :

Démarrage de la sauvegarde sur disque externe


Diffusion de message de root@ariane (somewhere) (Wed Oct 14
01:57:46 2015) :

Sauvegarde sur disque externe terminée
```

C'est vraiment lent un disque 5400RPM ;)

Pour accéder aux sauvegardes, je souhaitais conserver l'utilisation de l'utilitaire *Udisks2* qui ouvre une pop-up demandant interactivement un mot de passe de déchiffrement. La gestion du déchiffrement déléguée à systemd s'impose face à *Udisks2* (c'est sans doute pour cette raison qu'il devient instable et qu'un redémarrage est nécessaire), donc pour rétablir le fonctionnement d'*Udisks2*, il suffit de désactiver le service **systemd-cryptsetup@HTS542512K9SA00_120G.service** (par exemple, en commentant la ligne présente dans **/etc/crypptab** et en relançant le générateur).

3 Pour aller plus loin

La simple commande **cp** peut être remplacée par quelque chose de plus évolué comme **rsync**, mais la solution la plus optimisée est peut-être du côté du très prometteur **btrfs** qui présente une fonctionnalité de copie

Systemd

Il s'agit d'un gestionnaire de démarrage inspiré de son pendant sur Apple : **launchd**. Systemd tire parti de fonctionnalités avancées présentes sur le noyau Linux et est capable de démarrer les services très agressivement en les parallélisant au maximum. De plus, systemd permet de réagir à des événements : requête DBUS, connexion à une socket réseau, action sur un fichier (`inotify`), date et heure, exécution de règle udev, lancement d'un autre service. La gestion des services est également consolidée, en effet systemd garde la trace de chaque processus lancé grâce aux *cgroups* (en d'autres termes, `systemctl stop` marche à tous les coups) et est capable de redémarrer automatiquement un service qui a planté (*watchdog*). Par ailleurs, il apporte un certain nombre d'outils additionnels, comme un gestionnaire de logs (`journald`) dont l'avantage est d'indexer les logs ou encore un utilitaire permettant d'isoler des services dans un container (`nspawn`).

incrémentale permettant de surpasser les outils tels que `rsync` (par exemple, `btrfs` est capable d'identifier des changements affectant uniquement les métadonnées entre deux volumes, tels que le renommage d'un fichier).

Conclusion

Sauvegardez vos données, comme vous avez pu le constater, il n'y a rien de difficile ;) ■

Références

[1] DANTI G., *« LVM thin volume explained »* : http://www.ilsistemista.net/index.php/linux-a-unix/46-lvm-thin-volume-explained.html

[2] J.-P GARCIA BALLESTER. *« Aujourd'hui c'est déjà demain : systemd dans l'initrd sous Arch Linux »* : http://linuxfr.org/news/aujourd-hui-c-est-deja-demain-systemd-dans-l-initrd-sous-arch-linux

ANSIBLE & DOCKER SONT DANS UN BATEAU

Philippe Poumaroux [Architecte SI à la DIRISI et contributeur éclectique (LibrePlan, CMDBuild, Moonraker ...)]

Les conteneurs Docker offrent la possibilité inestimable de générer de multiples machines en quelques secondes. Ansible permet de provisionner de multiples machines très facilement et très rapidement. Imaginez qu'on associe les deux...

Mots-clés : Docker, Ansible, Conteneur, Galaxy, Python, CentOS

Résumé

L'objectif est de créer des playbooks Ansible et de pouvoir les tester rapidement dans des conteneurs Docker, au lieu de le faire sur sa machine avec tous les effets de bord que cela peut induire. En bonus, ces conteneurs Docker permettront d'obtenir des images réutilisables pour d'autres tests. De surcroît ces *playbooks* - ou plus exactement ces rôles Ansible, faisons propre jusqu'au bout - pourront également être utilisés sur des hôtes qui ne seront pas des conteneurs. En résumé, le beurre, l'argent du beurre et la crémière.

Les conteneurs (ou *containers*) Docker ouvrent des perspectives énormes, en développement comme en exploitation. Construire des images représentatives des environnements cibles pour ensuite les utiliser afin de tester ce qui a été fait et vérifier que cela fonctionnera sans problème en production, le tout sans devoir sortir l'artillerie lourde, en quelques secondes, voilà une avancée spectaculaire.

Toutefois, construire des images à l'aide des *Dockerfile* n'est pas toujours une sinécure dès qu'on veut dépasser la simple copie de fichiers ou l'exécution de commandes. Sans compter que la lisibilité n'y est pas, l'idempotence limitée, et qu'il manque plein de commodités qu'on prend vite goût à utiliser quand on a un peu joué avec Ansible.

La vie étant bien faite (ou plus exactement les développeurs d'Ansible étant des gars malins, d'ailleurs Red Hat ne s'y est pas trompé), Ansible possède des modules pour Docker. La porte est donc ouverte.

Donc voilà où je veux en venir : faire en sorte que ma machine hôte lance un *playbook* Ansible qui crée un conteneur Docker. Ceci fait, ce même *playbook* se connecte sur le conteneur qu'il vient de faire apparaître et y provisionne un rôle de notre choix, indiqué lors du lancement d'Ansible.

Évidemment, je ne suis pas assez malin pour avoir imaginé ça tout seul, c'est l'article « *The marriage of Ansible & Docker* » de Thomas Steinbach du 27 mai 2015 **[1]**, et notamment sa première partie, qui m'a enfin permis de trouver un moyen de parvenir à mes fins. Ce qui suit est la transcription de mon parcours pour parvenir à mettre en œuvre la solution proposée, mais aussi à la comprendre, à l'adapter et, occasionnellement, à la corriger ici et là pour qu'elle fonctionne dans mon environnement.

Enfin, je vais volontairement utiliser dans ce qui suit des noms français afin d'éviter toute ambiguïté avec les commandes ou mots clefs.

1 Les outils

Vous l'aurez compris, nous utiliserons principalement Ansible et Docker. Pour vous éviter des prises de têtes, synchronisons nos montres (ou plutôt les versions utilisées).

Concernant Ansible, j'utilise la version fournie par les paquets de ma distribution (une Ubuntu Wily - et je rappelle que Willy est une orque, rien à voir avec une baleine qui n'a pas besoin d'être sauvée par conséquent).

```
$ sudo apt-get install ansible
...
$ ansible --version
ansible 1.9.2
  configured module search path = None
```

Nous allons utiliser la dernière version disponible de Docker. On commence par supprimer la version déjà installée si nécessaire (le paquet s'appelle **docker.io** sous Ubuntu, le nom « docker » étant déjà utilisé pour un lanceur d'application) :

```
$ sudo apt-get remove docker.io
```

On récupère ensuite la bonne version dans un dépôt spécifique et on l'installe :

```
$ wget -qO- https://get.docker.com/ | sh
```

Pour vérifier que tout est en ordre, lancez la commande suivante :

```
$ docker run hello-world
```

Il se peut que vous ayez besoin de vous ajouter au groupe **docker** pour exécuter Docker autrement qu'en tant que **root** (ce n'était pas votre intention, n'est-ce pas ?) :

```
$ sudo usermod -aG docker poum
```

Évidemment, remplacez **poum** par votre nom d'utilisateur. Vous devez également quitter votre session et vous reconnecter pour que la modification soit prise en compte. Reprenons :

```
$ docker --version
Docker version 1.9.1, build a34a1d5
```

Si vous obtenez une erreur concernant l'absence du module **docker.client**, vous devrez l'installer :

```
$ sudo apt-get install python-docker
```

2 Prélude

Commençons par créer un répertoire pour notre petite affaire (**bateau**), un sous-répertoire pour le conteneur (**conteneur**) et un sous-sous-répertoire **clefs** pour les clefs ssh nécessaires à la communication entre Ansible et le conteneur Docker.

Pour mémoire, Ansible n'utilise pas d'agent sur la machine cible : il communique avec elle via ssh.

```
$ mkdir bateau
$ cd bateau
$ mkdir -p conteneur/clefs
$ ssh-keygen -t rsa -C "ansible@poum.org" -f conteneur/clefs/id_rsa
```

Aucun mot de passe n'est indiqué afin de ne pas avoir à le saisir quand Ansible initiera la communication.

3 Le conteneur

Occupons-nous maintenant du conteneur, dans sa version la plus minimaliste possible sur base de CentOS 6 (ça changera un peu de Debian et permettra d'avoir du **yum** plutôt que du **apt-get**). Ce conteneur doit être à jour, le plus petit possible et pouvoir communiquer avec Ansible, donc disposer d'un serveur SSH initialisé. Tout ceci se traduit par le *Dockerfile* suivant :

```
FROM centos:centos6
MAINTAINER poum <poum@cpan.org>

ADD clefs/id_rsa.pub /root/.ssh/authorized_keys
RUN yum -y update \
 && yum -y install openssh-server \
 && yum clean all \
 && service sshd start \
 && service sshd stop

EXPOSE 22
CMD ["/usr/sbin/sshd", "-D"]
```

On va placer ce fichier dans **bateau/conteneur**.

Ici, pour ceux qui sont moins habitués aux *Dockerfiles*, petite révision :

1. On part d'une image officielle CentOS en version 6 (**FROM**) ;
2. Je flatte mon ego en disant que c'est moi qui l'ai fait (**MAINTAINER**) ;
3. On copie dans le conteneur la clef publique ssh qu'on vient de générer pour Ansible (**ADD**) ;

4. On met à jour la distribution puis on installe le serveur ssh, on fait un peu de nettoyage pour réduire l'empreinte et enfin on lance et on arrête ssh (**RUN**). On chaîne les commandes avec **&&** et on les répartit sur plusieurs lignes avec **** pour plus de lisibilité. On utilise l'option **-y** avec les commandes **yum** vu qu'on ne sera pas en mode interactif pour les exécuter. En clair, on dit à **yum** : ne demande pas, c'est toujours **yes** ;
5. On demande au conteneur d'être à l'écoute sur le port **22**, celui de ssh (**EXPOSE**) ;
6. Enfin, on indique que par défaut, le conteneur lancera le serveur ssh en tâche de fond (**CMD**).

On ne fera rien de plus avec un *Dockerfile*, tout le reste sera réalisé avec Ansible.

Ajoutons également que l'image CentOS 6 utilisée ici contient par défaut un interpréteur Python (version 2.6), l'autre prérequis avec ssh pour qu'Ansible puisse frayer avec. Si tel n'avait pas été le cas, il aurait fallu ajouter l'installation de python dans notre *Dockerfile*.

Il est également clair ici, que même avec un *Dockerfile* très simple, on est loin de l'expressivité offerte par les *playbooks* d'Ansible. Pourquoi croyiez-vous que je voulais utiliser ce dernier ?

Revenons à nos moutons (oui, il y en a aussi en mer) et testons notre *Dockerfile* pour nous assurer que tout est bon. Depuis le répertoire **conteneur**, lancez :

```
$ docker build .
...
Successfully built 97bca8973936
```

À ce stade, nous avons la capacité de créer une image Docker sur une base CentOS 6 qui sait communiquer via ssh à l'aide de la clef **id_rsa**.

4 Ansible

4.1 Derniers préparatifs

Oui, je sais, encore ! Mais rassurez-vous, je ne vous mène pas en bateau, on en a bientôt fini avec ça. Nous allons configurer Ansible pour qu'il utilise ces clefs. Pour ce faire, retour à bord dans le répertoire… **bateau** où nous créons le fichier **ansible.cfg** suivant :

```
[defaults]
hostfile = inventaire
private_key_file = ./conteneur/clefs/id_rsa
host_key_checking = false
```

Nous disons à Ansible d'utiliser la clef privée **id_rsa** et de ne pas faire de contrôle de la clef par rapport à l'hôte, pour éviter d'avoir à confirmer à ssh que, pas de panique, tout va bien, même si l'hôte qu'on essaie de contacter a un peu changé depuis la dernière fois.

Quant à l'inventaire, il indique le fichier dans lequel seront définis les hôtes cibles auxquels Ansible doit s'adresser. Ici, ce sera du local, donc nous mettons dans ce fichier :

```
localhost ansible_connection=local
```

4.2 Le playbook générique

Maintenant, le vif du sujet : le *playbook*. Celui-ci va comporter deux parties. La première pour créer le conteneur, la seconde pour appliquer à ce conteneur un rôle de notre choix.

4.2.1 Création du conteneur

Commençons donc par la première partie de ce fichier **site.yml** créé dans le répertoire racine **bateau** :

```
---
- name: création du conteneur
  hosts: localhost
  sudo: True
  tasks:

    - name: application des droits requis par ssh
      file:
        path: "{{ ansible_env.PWD }}/conteneur/clefs/id_rsa"
        mode: 0600

    - name: mise à disposition de l'image docker
      docker_image:
        name: image_du_conteneur
        path: conteneur
        state: present

    - name: création du conteneur
      docker:
        image: image_du_conteneur
        name: "{{ nom_conteneur }}"
        detach: False
        expose: "{{ ports }}"
        ports: "{{ ports }}"
        state: running

    - name: ajout du conteneur à l'inventaire des hôtes
      add_host:
        hostname: "{{ item.Name }}"
```

```
    groups: docker
    ansible_ssh_host: "{{ item.NetworkSettings.IPAddress }}"
    ansible_ssh_port: 22
  when: item.State.Running == True
  with_items: docker_containers
```

Ce premier **play**, « création du conteneur », comporte 4 tâches :

1. « application des droits requis par ssh » : pour que ssh (donc Ansible) puisse se connecter en utilisant la paire de clefs générées précédemment, il faut les protéger, sinon ssh va refuser la connexion. Comme Ansible exécute le *playbook* sur la machine cliente, i.e. le conteneur, nous ne pouvons pas lui indiquer un chemin relatif. **ansible_env.PWD** fournit le chemin du répertoire de travail depuis lequel nous avons lancé le *playbook*. Comme le client *distant* est notre machine hôte, le chemin reste valide. Ne reste plus qu'à appliquer les droits que ssh va exiger : lecture/écriture pour le seul propriétaire, autrement dit le mode **600** ;
2. « mise à disposition de l'image docker » : Ansible doit ensuite vérifier que l'image que l'on veut utiliser est bien disponible, au besoin en la créant à partir du *Dockerfile* ;
3. « création du conteneur » : on utilise l'image mise à disposition dans l'étape précédente pour créer le conteneur dont le nom ne sera pas figé, mais passé en paramètre au moment de l'exécution d'Ansible via le paramètre **nom_conteneur**. De même, la liste des ports sera passée à l'exécution via le paramètre éponyme ;
4. « ajout du conteneur à l'inventaire des hôtes » : cette dernière opération va nous permettre d'ajouter ce conteneur tout nouveau, dès qu'il est actif, à la liste des machines connues, ce qui nous permettra ensuite de lui appliquer le rôle de notre choix (attention au **==**, il s'agit ici d'un test, pas d'une affectation). Le module **docker** d'Ansible stocke les informations relatives aux conteneurs dans la variable **docker_containers**. Cette variable contient une liste puisque plusieurs instances peuvent être lancées par la tâche. En utilisant le module **add_host**, nous rendons notre client Docker visible pour Ansible en passant son adresse IP dans son inventaire dynamique et plus précisément dans le groupe **docker**.

4.2.2 Application du rôle

Ajoutons maintenant à ce même fichier sa charge utile, celle qui permet l'application d'un rôle dans notre conteneur.

Pour rendre les choses génériques, on ne nomme pas ce rôle statiquement, mais on utilise encore une variable que l'on renseignera à l'exécution, **"{{ role }}"** :

```
- name: application du rôle
  hosts: docker
  remote_user: root
  roles:
- "{{ role }}"
```

4.3 Le rôle Ansible

Nous voilà enfin au moment de personnaliser notre conteneur pour avoir un peu plus qu'une simple CentOS 6. Dans le répertoire de départ (**bateau**), nous allons générer le squelette de notre rôle à l'aide de **ansible-galaxy** (connexion Internet requise) :

```
$ ansible-galaxy init laReponse
- laReponse was created successfully
```

Le sujet de cet article n'étant pas la réalisation d'un *playbook* sophistiqué, notre rôle sera donc très simple : il va afficher le nom du conteneur. Éditons donc **laReponse/tasks/main.yml** :

```
---
- debug: msg="42 dit le conteneur {{ inventory_hostname }}"
```

5 Mise à feu

C'est là que les Athéniens s'atteignirent, lançons notre commande Ansible avec les paramètres attendus :

```
$ ansible-playbook -v --ask-sudo-pass -e "nom_conteneur=conteneur
ports=['80'] role=laReponse" site.yml
```

Cette commande lancée depuis le répertoire **bateau** utilise le *playbook* **site.yml**, le paramétrage **ansible.cfg** et l'inventaire **inventaire** pour s'exécuter. Le nom de conteneur, les ports et le rôle sont précisés via l'option **-e**.

Au passage, signalons qu'il n'y a pas de **=** entre **-e** et la chaîne de caractères décrivant les paramètres, sinon Ansible n'en tiendra pas compte, ne dira rien, le coquin, jusqu'à ce que, ayant besoin d'un paramètre, il se « petit-suisside ». Si vous tenez à utiliser un signe **=**, utilisez **--extra-vars=**.

--ask-sudo-pass permet de communiquer le mot de passe **sudo** à utiliser au moment où cette commande sera exécutée à l'intérieur du conteneur.

```
SUDO password:

PLAY [création du conteneur] ***************************************************

GATHERING FACTS ****************************************************************
ok: [localhost]

TASK: [application des droits requis par ssh] **********************************
ok: [localhost] => {"changed": false, "gid": 1000, "group": "poum", "mode":
"0600", "owner": "poum", "path":
"/opt/poum/Dev/ansible2docker/conteneur/clefs/id_rsa", "size": 1679, "state":
"file", "uid": 1000}

TASK: [mise à disposition de l'image docker] ***********************************
changed: [localhost] => {"changed": true, "failed": false, "image_id":
"a568b50fc91b", "msg": "Image built: a568b50fc91b"}

TASK: [création du conteneur] **************************************************
changed: [localhost] => {"ansible_facts": {"docker_containers": [{"Id":
"9442523999c86ac4b778d68d0ef573669de38bb18a05e62e4756b15d05612379", "Warnings":
null}]}, "changed": true, "containers": [{"Id":
"9442523999c86ac4b778d68d0ef573669de38bb18a05e62e4756b15d05612379", "Warnings":
null}], "msg": "started 1 container, created 1 container.", "reload_reasons":
null, "summary": {"created": 1, "killed": 0, "pulled": 0, "removed": 0,
"restarted": 0, "started": 1, "stopped": 0}}

TASK: [ajout du conteneur à l'inventaire des hôtes] ************************
fatal: [localhost] => error while evaluating conditional: item.State.Running ==
True

FATAL: all hosts have already failed -- aborting

PLAY RECAP *********************************************************************
to retry, use: --limit @/home/poum/site.retry

localhost                  : ok=4    changed=2    unreachable=0    failed=1
```

5.1 Le drame

Au début, tout se passe bien, Ansible nous demande le mot de passe sudo, comme prévu, puis les différentes étapes que nous avons définies se déroulent les unes après les autres. Comme nous avons demandé à Ansible d'être bavard (**-v**), il s'épanche en JSON... Arrive l'étape d'ajout du conteneur à l'inventaire et là, c'est le drame ! Le cachalot vient de percuter la planète !

Dans **site.yml**, nous utilisons les données retournées par Ansible, les **ansible_facts**, pour récupérer l'adresse IP du conteneur en vue de l'ajouter à l'inventaire. Mais si vous regardez de plus près la sortie précédente, vous verrez que dans ces **ansible_facts**, et dans la clef **docker_containers** plus précisément, on a bien une liste, mais cette liste ne contient qu'un **Id**.

Thomas Steinbach précise qu'il s'agit un bug des versions actuelles du module Docker d'Ansible, introduit avec la version 1.8. En allant voir sur GitHub, on s'aperçoit que la correction qu'il a proposée a été intégrée, mais qu'elle ne sera pas disponible avant une prochaine version majeure d'Ansible, le module Docker faisant partie de la distribution de base.

Nous allons donc examiner d'abord la solution proposée, essentiellement parce qu'elle nous permettra de jeter un œil sur **docker exec**. Mais, disons-le tout de go, cette solution n'a fonctionné pour moi qu'au prix d'acrobaties qui ne sont plus de mon âge. Nous verrons donc dans un second temps une solution plus radicale, mais plus propre de mon point de vue.

5.2 Solution acrobatique

Pour contourner le problème, Thomas Steinbach propose de modifier le *playbook* **site.yml** en replaçant la dernière tâche de la première partie (« ajout du conteneur à l'inventaire des hôtes ») par :

```
    - name: demande de l'adresse IP du conteneur
      shell: "docker exec {{ docker_
containers[0].Id }} hostname -I"
      register: _requete_conteneur

    - name: extraction de l'adresse IP du
résultat de la requête
      set_fact:
        _ip_conteneur: "{{ _requete_conteneur.
stdout | regex_replace('\\s','') }}"

    - debug:
        msg: "L'adresse IP du conteneur est {{
_ip_conteneur }}"

    - name: ajout du conteneur à l'inventaire
des hôtes
      add_host:
        hostname: "{{ item.Name }}"
        groups: docker
        ansible_ssh_host: "{{ _ip_conteneur }}"
        ansible_ssh_port: 22
      when: item.State.Running == True
      with_items: docker_containers
```

Expliquons un peu ce qui est fait ici. D'abord, on utilise **docker exec** pour exécuter une commande directement dans le conteneur. La commande en question est **hostname -I** qui affiche toutes les adresses IP de l'hôte. Dans le cas de notre conteneur, il n'y en aura qu'une et c'est tant mieux, car l'instruction suivante ne fonctionnerait pas sinon. Le résultat est enregistré dans la variable **_requete_conteneur**. Ensuite, on applique une expression rationnelle sur le résultat de la sortie standard de **_requete_conteneur** pour effacer

tous les espaces. Le résultat est placé dans une nouvelle variable **_ip_conteneur**. L'étape suivante, définitivement facultative, se contente d'afficher un message de débogage indiquant cette adresse à l'aide du module **debug**. On retrouve enfin la dernière étape vue précédemment, sauf que l'adresse IP n'est cette fois plus récupérée depuis les faits, mais via notre nouvelle variable **_ip_conteneur**.

Relançons notre commande pour vérifier que la solution de contournement fonctionne. On n'oubliera pas au préalable d'arrêter le conteneur qui est resté actif depuis notre première tentative :

```
$ docker kill conteneur
$ ansible-playbook -v --ask-sudo-pass -e "nom_conteneur=conteneur
ports=['80'] role=laReponse" site.yml
```

De deux choses l'une. Soit vous utilisez Ansible en version 1.7, et ça va bien se passer, soit vous utilisez Ansible en version 1.9 et, les mêmes causes produisant les mêmes effets, Ansible n'obtiendra ni **item.Name**, ni **item.State.Running** (item prenant successivement les valeurs des éléments de la liste **docker_containers**).

```
TASK: [demande de l'adresse IP du conteneur] **********************************
...
TASK: [ajout du conteneur à l'inventaire des hôtes] ***************************
fatal: [localhost] => error while evaluating conditional: item.State.Running == True

FATAL: all hosts have already failed -- aborting

PLAY RECAP ********************************************************************
          to retry, use: --limit @/home/poum/site.retry

localhost                  : ok=7    changed=2    unreachable=0    failed=1
```

Ne désespérons pas : dans la dernière tâche, nous allons apporter les modifications suivantes :

- Remplacer **{{ item.Name }}** par **{{ nom_conteneur }}** dont la valeur est passée en paramètre, donc connue ;
- Commenter la ligne **when: item.State.Running == True** en faisant le pari que le conteneur sera bien en cours d'exécution à ce moment-là :

```
    - name: ajout du conteneur à l'inventaire des hôtes
      add_host:
        #hostname: "{{ item.Name }}"
        hostname: "{{ nom_conteneur }}"
        groups: docker
        ansible_ssh_host: "{{ _container_ip }}"
        ansible_ssh_port: 22
      #when: item.State.Running == True
      with_items: docker_containers
```

Troisième tentative qui devrait aboutir, cette fois, parce que c'est assez :

```
$ docker kill conteneur
$ ansible-playbook -v --ask-sudo-pass -e "nom_conteneur=conteneur
ports=['80'] role=laReponse" site.yml
...
TASK: [{{ role }} | debug msg="42 dit le conteneur {{ inventory_
hostname }}"] ***
<172.17.0.2> ESTABLISH CONNECTION FOR USER: root
ok: [conteneur] => {
    "msg": "42 dit le conteneur conteneur"
}

PLAY RECAP ********************************************************************
conteneur                  : ok=2    changed=0    unreachable=0    failed=0
localhost                  : ok=8    changed=2    unreachable=0    failed=0
```

5.3 Solution en profondeur

La solution précédente fonctionne, mais impacte le *playbook* qui se trouve contaminé par le contournement. De ce fait, quand le problème du module Docker aura été résolu (d'une manière ou d'une autre, la livraison d'un module **docker_facts** étant évoquée), il faudra corriger le *playbook* pour revenir à la version initiale.

Nous allons donc opter pour une approche plus musclée, mais plus KISS : patcher Ansible pour restaurer le comportement souhaité en attendant la solution pérenne.

Téléchargeons donc la dernière version stable d'Ansible (1.9.4 au moment de l'écriture de cet article) sur http://releases.ansible.com/ansible/. On décompresse l'archive :

```
$ tar xzvf ansible-1.9.4.tar.gz
$ cd ansible-1.9.4/lib/ansible/modules/core/cloud/docker
```

Et là, on ouvre **docker.py** et à la fin du fichier, ligne 1561, on modifie ce que renvoie le module comme suit :

```
        module.exit_json(changed=manager.has_changed(),
                         msg=manager.get_summary_message(),
                         summary=manager.counters,
                         containers=containers.changed,
                         reload_reasons=manager.get_reload_
reason_message(),
                         ansible_facts=_ansible_facts(containers.
changed))
```

On supprime (commente) cette dernière ligne et on la remplace par :

```
ansible_facts=_ansible_facts(manager.get_inspect_
containers(containers.changed)))
```

On retourne à la racine de l'archive (**ansible-1.9.4**) pour lancer l'installation de cette version « réparée » :

```
$ sudo python setup.py install
```

Note

Il se peut que vous vous fassiez insulter par le serpent (beaucoup moins poli qu'un chameau) :

```
Ansible now needs setuptools in order to build. Install it using
your package manager (usually python-setuptools) or via pip (pip"
install setuptools).```
```

La commande suivante doit vous permettre de vous en sortir :

```
$ sudo apt-get install python-setuptools
```

Voilà, nous avons la dernière version stable d'Ansible :

```
$ ansible --version
ansible 1.9.4
```

Mais surtout, si nous retournons dans le répertoire **bateau** et que nous relançons notre *playbook* (version initiale), tout va enfin fonctionner de la façon attendue.

```
$ docker kill conteneur
$ ansible-playbook -v --ask-sudo-pass -e "nom_conteneur=conteneur
ports=['80'] role=laReponse" site.yml
...
TASK: [{{ role }} | debug msg="42 dit le conteneur {{ inventory_hostname
}}"] ***
<172.17.0.2> ESTABLISH CONNECTION FOR USER: root
ok: [conteneur] => {
    "msg": "42 dit le conteneur conteneur"
}

PLAY RECAP ********************************************************************
conteneur                  : ok=2    changed=0    unreachable=0    failed=0
localhost                  : ok=8    changed=2    unreachable=0    failed=0
```

Plus simple, moins invasif, plus pérenne et en bonus on peut utiliser la dernière version d'Ansible, histoire d'épater un petit peu la galerie.

Conclusion

Nous avons donc obtenu que notre rôle **laReponse** soit appliqué par Ansible à notre conteneur Docker CentOS 6. Le rôle est testé et pourra être appliqué sur tout hôte CentOS 6, qu'il soit dans un conteneur, une VM ou une machine physique.

Certes, ça fait beaucoup de choses à mettre en place pour tester ce premier rôle. Mais, d'une part, il devient très facile de tester ce rôle au fur et à mesure de sa construction (d'autant que les mises en cache de Docker et l'idempotence d'Ansible sont là pour accélérer les choses et ne pas refaire ce qui a déjà été fait) et, d'autre part, le gain est immédiat dès le deuxième rôle : il suffit de partir du paragraphe précédent, tout le reste est directement réutilisable pour peu que vous utilisiez la même image de base pour votre conteneur. Au pire, si vous voulez travailler sur une autre image, disons une Debian, vous n'avez qu'à modifier le *Dockerfile* du répertoire **conteneur** (nom de l'image et commandes de mise à jour et d'installation du serveur ssh).

J'ajoute enfin que les *playbooks* sont beaucoup plus simples à tester que les *Dockerfile*, notamment parce qu'Ansible repose sur le principe des états à atteindre. Donc si une tâche échoue, l'état visé ne sera pas atteint et Ansible tombera en erreur. De ce fait, il n'est pas nécessaire d'ajouter des tests dans la plupart des cas. La documentation d'Ansible consacre d'ailleurs une page **[2]** à ce sujet.

Nous nous sommes donc bien conformés à la Sainte Paresse **[3]**. Et il va être ensuite très facile de jouer avec ces conteneurs, de les mettre en réseau avec **docker-composer**, d'y installer des applications et de les soumettre à des tests fonctionnels avec **Moonraker** par exemple... une prochaine fois ? ■

Références

[1] Steinbach T., « *The marriage of Ansible and Docker* » : https://bildung.xarif.de/xwiki/bin/Articles/The+Marriage+of+Ansible+and+Docker

[2] *Testing strategies* : http://docs.ansible.com/ansible/test_strategies.html

[3] Les 3 vertus du programmeur : https://fr.wikipedia.org/wiki/Larry_Wall#Les_trois_vertus_du_programmeur

ACTUELLEMENT DISPONIBLE

LINUX PRATIQUE N°93 !

SOYEZ PRÊTS À REPOUSSER TOUTES LES ATTAQUES !

NE LE MANQUEZ PAS

CHEZ VOTRE MARCHAND DE JOURNAUX ET SUR :

www.ed-diamond.com

PROTOCOLE DE DÉCOUVERTE DE SERVICES

Pierre-Emmanuel Gros [Responsable Innovation, Neuresys]

Le but de cet article est de mettre au point un protocole de découverte de services (base de données, serveur apache, etc.) indépendant de la localisation réseau du service. Nous appliquerons ce protocole au cas particulier de MariaDB.

Protocole réseau JSON MariaDB Multicasting

L'OBJECTIF

Les services réseaux (bases de données, serveur web...) sont exécutés sur des machines ayant une adresse IP sur un port connu d'avance. Par exemple, **MariaDB** utilise le port TCP `3306`, PosgreSQL le port `5432` ou encore les serveurs Web le port `80`. Les architectures techniques des applications distribuées passent donc par une connaissance a priori des différentes localisations réseaux de ces services. Le but de cet article est de montrer comment remplacer cette connaissance a priori par un protocole réseau permettant de trouver dynamiquement les services. Ce protocole s'inspire du protocole générique SSDP implémenté, mais emploie plutôt le format JSON sans utilisation du HTTP.

L'objectif de cet article est donc de remplacer le paradigme où un client se connecte à un serveur sur un couple adresse IP/port TCP bien connu par un paradigme où le client demande au réseau où se trouve le service demandé avant la connexion.

LES OUTILS

Afin de mettre en place ce *how-to*, il nous faut les ingrédients suivants :

- une machine virtuelle Java (traditionnellement installée via le *package* `openjdk`) avec le compilateur associé :

```
$ sudo apt-get install openjdk-8-jdk
```

- un environnement de développement décent tel qu'**Eclipse** :

```
$ sudo apt-get install eclipse-platform
```

- les *packages* `jackson-core`, `annotations`, `databinding` ainsi que le driver `JDBC` de **MySQL** récupérable soit en téléchargement, soit via des *packages* **mavenun** ;
- une base de données MariaDB en version source et compilée par nos soins :

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

- la librairie JSON disponible sur https://github.com/json-c/json-c.

PRÉPARATION : définition d'un protocole

Nous allons commencer par définir un petit protocole de communication. Nous allons utiliser deux messages `ASK` et `HELLO`. Le message `ASK` est envoyé lorsqu'un service est recherché sur le réseau. Par exemple, lorsque le service MariaDB est recherché sur le réseau, un message `ASK "MariaDB"` est envoyé. Le message `HELLO` contient la localisation d'un service sur le réseau. Il peut au choix, soit être une réponse

à un message **ASK**, soit être envoyé de façon périodique sur le réseau.

Un message **ASK** contient le nom du service recherché ainsi qu'un identifiant unique pour le message.

Enfin, le message **HELLO** contient un nom de service, une localisation réseau (adresse IP, port TCP) ainsi qu'un identifiant. La règle de cet identifiant est la suivante : si le message **HELLO** est une réponse à une requête **ASK**, alors les deux identifiants sont identiques.

Les messages seront donc définis ainsi :

	Message ASK	Message HELLO
Type de message	Champ type : ASK	HELLO
Champ Identifiant	Champ ID	
Nom du service	Champ service	
Localisation IP du service	Champ host	
Port TCP du service	Champ port	

PHASE 1

Implémentation du protocole

Nos messages **ASK** et **HELLO** vont être implémentés en Java et en C. La partie Java sera utilisée par un client JDBC pour MariaDB et la partie serveur sera embarquée dans MariaDB. Du fait que notre protocole sera utilisé dans les deux langages, la solution la plus simple est de l'implémenter dans les deux langages.

En Java, la définition **MessageType** sera un énuméré de **ASK** et de **HELLO** dénotant ainsi le type du message.

La classe **ServiceInfo** contient le nom du service ainsi qu'un identifiant :

```
public class ServiceInfo{
  private String service;
  private String ID;
}
```

Un message est donc la composition du type du **Message** et d'un objet **ServiceInfo**.

```
@JsonInclude( JsonInclude.Include.NON_NULL)
public class Message {
  MessageType type;
  @JsonUnwrapped
    ServiceInfo info;
}
```

Dans notre protocole, le message **ASK** est une simple instance de **Message** (avec évidemment l'attribut **type** positionné à la valeur **ASK**).

Le lecteur remarquera les annotations de la librairie Jackson : **JsonInclude** et **JsonUnwrapped**. Ces dernières signifient respectivement que seuls les attributs non nuls peuvent être transformés en JSON, et que la transformation en JSON doit se faire « à plat ».

Le message **HELLO** est une extension du message **ASK** et se définit ainsi :

```
public class ServiceResponseInfo extends ServiceInfo {
  private String host;
  private Integer port;
}
```

L'implémentation en C se définit de façon assez directe à l'aide de **struct** :

- structure **ServiceInfo** définissant le message **ASK** :

```
typedef struct __serviceinfo {
  const char * type;
  const char * ID;
  const char * serviceName;
} serviceinfo;
```

Et la structure **ResponseInfo** définissant le message **Hello** :

```
typedef struct __responseinfo {
  const char * type;
  const char * ID;
  const char * serviceName;
  unsigned int port;
  const char * host;
} responseinfo;
```

PHASE 2

Gestion de la sérialisation/ désérialisation en JSON

Notre protocole utilise donc le format JSON pour transmettre les messages. Pour la partie Java, la librairie Jackson va effectuer le travail en utilisant les annotations :

- Sérialisation d'un message vers une chaîne de caractères :

```
public String serializeToString(Message message) {
  ObjectMapper mapper = new ObjectMapper();
```

```
  StringWriter writer=new StringWriter();
  return writer.toString();
}
```

- Désérialisation d'un message vers une chaîne de caractères :

```
public Message unserializeToString(String message) {
  ObjectMapper mapper = new ObjectMapper();
  Message m=mapper.readValue(message, Message.class);
  return m ;}
```

Concernant la partie C++, la sérialisation json se fait en utilisant les API suivantes :

- **json_object_new_object** : créer un objet json vide ;
- **json_object_object_add** : une fonction rajoute dans un objet json une clef et une valeur pour cette clef ;
- **json_object_object_get_ex** : permet de récupérer une valeur de clef dans un objet json ;
- **json_tokener_parse/json_object_to_json_string** : permet la transformation d'une chaîne de caractères en objet json et vice-versa.

PHASE 3

Envoi des messages sur un groupe multicast

Le monde de la communication réseau supporte caricaturalement trois types d'envois de message : l'envoi d'un émetteur à un récepteur (communication *unicast*), l'envoi d'un émetteur à tous les mondes (autrement appelé *broadcast*) et l'envoi d'un émetteur vers un groupe de récepteurs (envoi *multicast*).

L'avantage de ce dernier type d'envoi est qu'un message n'est envoyé qu'aux membres ayant adhéré à un groupe *multicast* bien connu.

L'idée de notre protocole est donc de prendre un message de type **ASK**, de le sérialiser en JSON et de l'envoyer sur un groupe *multicast*. Si d'aventure un service est amené à répondre à cette requête **ASK**, il répondra sur le groupe *multicast* via du JSON un message **HELLO**.

Les protocoles utilisant la communication *multicast* doivent pour des raisons assez évidentes utiliser une couche de transport UDP et utiliser des adresses IP dites **multicast**. En pratique, ces adresses vont de **224.0.0.0** à **239.255.255.255**, les 28 bits les moins significatifs constituent l'adresse du groupe.

Notre groupe utilisera pour la cause l'adresse *multicast* **224.0.0.3** sur le port **8888**.

Phase 3.1 - Un groupe multicast en Java

En Java, l'ouverture d'une tel socket se fait ainsi :

```
// la socket est créée sur le port 8888 et l'adresse 224.0.0.3
socket = new MulticastSocket(8888);
address = InetAddress.getByName(" 224.0.0.3 ");
// j'utilise ma carte wifi comme interface réseau
socket.setNetworkInterface(" wlan0 ");
// même si le port 8888 est déjà utilisé, nous nous autorisons à
le réutiliser
socket.setReuseAddress(true);
// les packets UPD envoyés sur la socket seront recopiés vers
l'émetteur
socket.setLoopbackMode(true);
//Enfin on rejoint le groupe Multicast
socket.joinGroup(address);
```

L'envoi et la réception de données se fait via des classiques paquets UDP (représentés en Java par la classe **DatagramPacket**) et les méthodes **receive**/**send** de l'API Java de haut niveau.

ANS s'utilise donc à travers une classe **GroupSocket** utilisant une socket *multicast*. Cette classe **GroupSocket** contient une méthode **sendMessage** envoyant un datagramme sur le réseau :

```
public void sendMessage(Message message) throws IOException {
  String msg=JsonSerializer.INSTANCE.serializeToString(message);
  DatagramPacket msgPacket = new DatagramPacket(msg.getBytes(),msg.
getBytes().length, InetAddress.getByName("224.0.0.3"), 8888);
  socket.send(msgPacket);
}
```

Afin de faciliter l'utilisation d'ANS, nous introduisons une interface **MulticastSocketListener** qui contient une méthode appelée lorsqu'un message arrive :

```
.public interface MulticastSocketListener {
  public void handleMessage(GroupSocket socket,Message message);
}
```

Ces dernières interfaces sont enregistrées dans une liste d'interfaces :

```
List<MulticastSocketListener> listener=new ArrayList<MulticastSocket
Listener>();
public void addMulticastSocketListener(MulticastSocketListener l) {
  listener.add(l);
}
```

Elles seront invoquées lors de la réception d'un message :

```
Thread receiver=new Thread() {
  public void run() {
    while(true) {
      DatagramPacket msgPacket = new DatagramPacket(buf, buf.length);
      try {
        socket.receive(msgPacket);
        String msg = new String(buf, 0, buf.length);
        Message message=JsonSerializer.INSTANCE.
unserializeToString(msg);
        for (MulticastSocketListener l:listener) {
          l.handleMessage(GroupSocket.this, message);}
      } catch (IOException e) {
        e.printStackTrace();
      }
    }
  }
};
receiver.setDaemon(true);
receiver.start();
```

Phase 3.2 - Implémentation en C

L'implémentation en C est quasi identique à celle du monde Java, si ce n'est l'initialisation de la socket **Multicast** qui est réalisée avec des fonctions de plus bas niveau.

L'implémentation étant un peu verbeuse, nous invitons le lecteur à consulter les (très) nombreuses pages sur le sujet tel que http://www.tenouk.com/Module41c.html.

Néanmoins, les étapes sont les suivantes :

- création d'une socket **sd** UDP via un appel à socket (**AF_INET**, **SOCK_DGRAM, 0**) ;
- appliquer l'équivalent en C du **setReuseAddress** Java en invoquant :

```
int reuse = 1;
setsockopt(sd, SOL_SOCKET, SO_REUSEADDR, (char *)&reuse, sizeof(reuse))
```

- faire un *bind* sur le port du groupe *multicast* ;
- rejoindre le groupe *multicast* via un appel :

```
group.imr_multiaddr.s_addr = inet_addr(addresse du groupe);
group.imr_interface.s_addr = inet_addr(addresse de l'interface réseau);
setsockopt(sd, IPPROTO_IP, IP_ADD_MEMBERSHIP, (char *)&group,
sizeof(group))
```

L'envoi et la réception des données se fait via la fonction **read** et **sendto** prenant en paramètre la socket du groupe *multicast*, et via un *buffer* de données ainsi que le groupe *multicast* (pour la fonction **sendto**).

PHASE 4

Implémentation du protocole dans MariaDB

L'idée de cette phase 4 est de faire démarrer le démon MariaDB avec la capacité de pouvoir envoyer de façon périodique des requêtes **HELLO**.

Nous allons donc passer via la notion de « daemon plugin » définit dans la documentation de MariaDB.

Développer un tel *plugin* se fait via quelques étapes : définir les données du *plugin* dans le fichier **my.ini**, définir le *plugin* en lui-même dans le code C et enfin envoi d'un paquet **HELLO** sur le réseau toutes les 2 secondes.

Afin de pouvoir lancer notre *plugin*, il nous faut à minima dans le fichier **my.ini** l'adresse IP du groupe *multicast*, le port du groupe, l'interface réseau et l'envoi du nom du service.

Le code d'un *plugin* dans MariaDB se définit en trois parties :

- une partie lisant les valeurs des variables système nécessaires au *plugin* ;
- une partie effectuant l'enregistrement, l'initialisation ainsi que le travail du *plugin* ;
- une partie dans MariaDB exécutant le *plugin*.

Phase 4.1 - Lecture des valeurs des variables système

Nous rajoutons dans le fichier **my.iniles** les variables suivantes :

- **ans_groupip** : qui définit l'adresse IP du groupe ;
- **ans_port** : qui définit le port du groupe ;
- **ans_host** : définissant le nom de l'hôte ;
- **ans_interface** : qui définit le nom de l'interface réseau ;
- **ans_servicename** : donnant le nom du service.

Pour que ces variables puissent être utilisées dans le code du *plugin*, il faut définir les variables C équivalentes. Par exemple, la variable système **ans_groupip** va être mise dans la variable C **groupip**.

La lecture des valeurs du fichier **my.ini** vers des variables C se fait via les macros **MYSQL_SYSVAR_X** (où **X** représente le type de la variable).

Ces macros prennent en paramètre :

1. l'identifiant de la variable système ;
2. le nom de la variable statique en C ;
3. des options concernant l'accès à la variable (lecture seule, visible à l'utilisateur) ;
4. les commentaires décrivant la variable ;
5. des pointeurs sur des fonctions permettant de vérifier et mettre à jour les variables ;
6. les bornes minimales et maximales des valeurs possibles ;
7. une valeur par défaut.

Si nous prenons la variable système **ans_groupip** devant être transformée dans la variable **groupip**, nous pouvons évoquer la macro suivante :

```
static MYSQL_SYSVAR_STR(groupip, groupip,
  PLUGIN_VAR_READONLY | PLUGIN_VAR_RQCMDARG,
  "ANS Group IP", NULL, NULL,
   "224.0.0.3);
```

Cette macro met la valeur de la variable système **ans_groupip** (composée du nom du *plugin* et du nom de la variable) dans la variable C **groupip**. Il s'agit d'une variable système en lecture seule et ayant une valeur au démarrage. Cette valeur est commentée en « ANS Group IP », n'a pas de fonction pour valider/mettre à jour la valeur et enfin la valeur par défaut est de **224.0.0.3**.

La liste des valeurs de macro doit se retrouver dans un tableau d'objets **st_mysql_sys_var** :

```
struct st_mysql_sys_var*vars_system_var[] = {  MYSQL_SYSVAR(groupip),
MYSQL_SYSVAR(interface), MYSQL_SYSVAR(hostname),...NULL}
```

Phase 4.2 - Initialisation du plugin

Les *plugins* possibles dans MariaDB sont de plusieurs sortes : *plugin* d'audit, de couche de stockage... qui ont potentiellement besoin d'informations pour démarrer.

Dans notre cas, la seule information pertinente pour notre *plugin* est la version de l'API que nous utilisons :

```
struct st_mysql_daemon ans_info={ MYSQL_DAEMON_INTERFACE_VERSION  };
```

Un *plugin* commence par l'évocation de **maria_declare_plugin** et termine sa déclaration par l'invocation de **maria_declare_plugin_end**.

```
maria_declare_plugin(ssdp) {
  MYSQL_DAEMON_PLUGIN,
  &ans_info,
  "ans",
  "Neuresys"
  "lancement du protocole ANS",
  PLUGIN_LICENSE_GPL,
  plugin_ans_init,
  plugin_ans_deinit,
  0x0100 /* 1.0 */,
  NULL,
  vars_system_var,
  "1.0",
    MariaDB_PLUGIN_MATURITY_EXPERIMENTAL
}
maria_declare_plugin_end;
```

La première ligne de cette structure signale que nous écrivons un *plugin* de type daemon et qui a besoin comme information de la structure précédemment définie.

Ce dernier *plugin* s'appelle « ans » (ligne 3), est écrit par la société Neuresys (ligne 4), concerne le lancement du protocole ANS (ligne 5) et se situe sous licence GPL (ligne 6). Les deux lignes suivantes (**plugin_ans_init** et **plug_ans_deinit**) sont des pointeurs vers des fonctions à exécuter, respectivement lors du chargement/déchargement du *plugin*.

Nous signalons en ligne 9 et 12 que notre *plugin* est en version 1.0. La ligne 11 est un pointeur vers le tableau des variables systèmes que nous allons lire pour exécuter notre *plugin*. Enfin, la ligne 13 signifie que notre *plugin* est au stade de maturité « expérimental ». Autrement dit, il s'agit d'un prototype.

Lorsque notre *plugin* va être chargé, MariaDB va exécuter la fonction **plugin_ans_init** déclarée comme telle :

```
static int plugin_ans_init(void *p __attribute__ ((unused))).
```

Cette fonction va créer le groupe *multicast* via la fonction **createsocket** en utilisant les variables **interface**, **groupip** et **port**. Cette socket est passée à un *thread* créé via la librairie **pthread** en mode *Join*.

Le code de la fonction exécuté par le *thread* est le suivant, et a pour but unique d'envoyer un paquet **HELLO** sur le groupe :

```
void * ans_thread(void *p) {
   groupsocket *group= (groupsocket *)p;
  serviceinfo * info=createserviceinfo(servicename);

  while(1) {
```

```
    sleep(2);
    sendServiceInfo(socket,info);
  }
  return 0;
}
```

Phase 4.3 - Installation du plugin

Le *plugin* ainsi créé doit être compilé et placé dans le sous-répertoire *plugin* du répertoire **lib** de MariaDB.

Afin de pouvoir le compiler proprement, ce dernier doit être vu comme une librairie dynamique (**-shared**), avec indépendance du positionnement de code (option **-fPic**) et en définissant la variable **MYSQL_DYNAMIC_PLUGIN**.

Afin de lier le *plugin* avec les API MariaDB, il faudra faire le *linkage* avec la librairie **libmysqlservice.a**.

Une fois compilé et installé, nous devons définir dans le fichier de configuration de MariaDB (**my.cnf** ou **my.ini** selon les plateformes) les variables systèmes **ans_groupip**, **ans_interface**, etc. dont nous donnons un extrait ici :

```
# The MySQL server
[mysqld]
ans_groupip=224.0.0.3
ans_interface=eth0
...
```

Si nous avons tout bien fait, il est possible de demander à MariaDB de charger le *plugin* via la commande **install plugin**.

```
MariaDB [(none)]> install plugin ans SONAME 'libplugin_ans';
Query OK, 0 rows affected (0.00 sec)
```

Une fois cette commande exécutée, normalement le *plugin* démarre et commence à envoyer des requêtes **HELLO** sur le réseau.

LE RÉSULTAT

Nous avons mis en place dans ce *how-to* un protocole de localisation de services sur un réseau. Ce protocole utilise le format JSON et est envoyé sur le réseau via un canal *multicast*.

Avec ce protocole, nous avons fait une implémentation dans deux langages (C et Java), puis nous l'avons transformée en *plugin* dans MariaDB.

Actuellement, ANS nous sert à retrouver facilement nos serveurs dans le réseau sans à avoir à reconfigurer chaque serveur. ■

FORMAT WAV : CRÉEZ DES ONDES SONORES EN C

Vincent Magnin [Maître de conférences à Polytech Lille et l'IEMN, administrateur du projet gtk-fortran]

Vous voulez revivre les balbutiements de la musique électronique et les émois acoustiques des pionniers des années 50-60 ? Je vous propose en une série de deux articles d'autopsier le vénérable format WAV et de revenir aux bases de la synthèse sonore à l'aide d'un court programme en C.

Mots-clés : Format WAV, Linear Pulse-Code Modulation, Little endian, Langage C, Synthèse sonore

Résumé

Un fichier WAV est un fichier binaire composé d'un en-tête suivi de signaux sonores qui, la plupart du temps, sont simplement codés en *Pulse Coded Modulation (PCM).* À chaque pas d'échantillonnage, l'amplitude de chaque canal sonore est codée par un entier signé sur deux octets, écrit en *little endian.* Il est donc très simple d'écrire un petit programme en C pour créer des ondes sonores. C'est ce que nous allons faire dans ce premier article avec comme modeste objectif d'obtenir un son sinusoïdal pur, brique de base qui nous permettra de générer des sons beaucoup plus complexes dans un second article.

Bien que l'on puisse faire remonter ses origines aux débuts de l'électricité, la première grande période d'exploration de la musique électronique se situe dans les années 50. Les pionniers se nomment Herbert Eimert (*Klangstudie II,* 1952), Karlheinz Stockhausen (*Elektronische Studie I,* 1953), Louis et Bebe Barron (B.O.F. de *Forbidden Planet,* 1956), etc. Ils synthétisent et manipulent des sons inouïs, c'est-à-dire que personne n'avait jamais entendu. Vous trouverez facilement sur YouTube leur travail exploratoire ainsi que les premières musiques populaires entièrement électroniques à la fin des années 60 et durant les années 70 (Gershon Kingsley, Kraftwerk, Tangerine Dream, Jean-Michel Jarre, etc.).

Soixante ans après ces pionniers, nous allons nous aussi nous initier à la synthèse sonore, en partant comme Stockhausen de la sinusoïde pure, à l'aide d'un court programme en C qui permet de créer des fichiers au format WAV. Nous allons donc explorer simultanément ce programme et la structure d'un fichier WAV classique, et terminer ce premier article par la synthèse d'un son sinusoïdal pur.

1 Le programme

Vous pouvez récupérer le fichier `synthe.c` sur le dépôt GitHub du magazine. Le code source est écrit en C, dans un style impératif simple et évitant autant que possible les syntaxes idiomatiques du C afin qu'il soit aisément compréhensible et traduisible dans tout autre langage. À vous de prendre possession du code, de l'améliorer ou de le réécrire au fil de vos explorations. Si vos souvenirs

du C sont lointains, replongez-vous dans les hors-séries de GLMF consacrés au C **[1, 2]**.

1.1 Programme principal

Rappelons que le son se propage dans l'air sous forme d'une onde de pression longitudinale, qui peut provenir des vibrations d'un corps matériel ou des mouvements de la membrane d'un haut-parleur. Un son peut donc être décrit par son amplitude en fonction du temps au niveau de sa source. Cette amplitude sera stockée dans des tableaux de réels : un pour le canal gauche et un pour le droit (ligne 10). La taille de ces tableaux sera calculée en fonction du taux d'échantillonnage et de la durée de la piste sonore. Ces variables sont globales :

```
01: #include <stdio.h>
02: #include <math.h>
03: #include <stdlib.h>
04:
05: #define PI 3.14159265358979323846264338327
06: #define AMPLITUDE_MAXI 32767
07:
08: unsigned int taux ;
09: unsigned long int taille ;
10: double *gauche, *droite ;
11: double duree ;
```

Le programme principal est simplement chargé de créer le fichier **synthe.wav** qui est un fichier binaire (paramètre **"wb"** ligne 108), d'y écrire l'en-tête du fichier WAV, de remplir les tableaux gauche et droite avec des signaux sonores ligne 111 (ici de la seconde 0 à la seconde 5 avec une fréquence de 440 Hz et une amplitude unitaire), qui sont ensuite normalisés selon **AMPLITUDE_MAXI** et écrits sous forme d'entiers signés dans le fichier (ligne 114). Avant de rendre la main, la mémoire allouée aux tableaux est libérée (lignes 117 et 118). Quant à la procédure **generateur_enveloppe()**, elle sera présentée dans l'article suivant consacré aux méthodes de synthèse sonore.

```
107: int main(void) {
108:     FILE *fichier_wav = fopen("synthe.wav", "wb") ;
109:     ecrire_en_tete_WAV(fichier_wav) ;
110:
111:     mon_signal(0.0, 5.0, 440.0, 1.0) ;
112:     //generateur_enveloppe(0.0, 5.0, 30.0, 20.0, 80.0, 30.0) ;
113:
114:     ecrire_donnees_normalisees_WAV(fichier_wav) ;
115:     fclose(fichier_wav) ;
116:
117:     free(gauche) ;
118:     free(droite) ;
119: }
```

1.2 Petit-boutiste

Dans un fichier WAV, tous les entiers sont stockés en *little-endian,* c'est-à-dire que l'on commence par écrire les octets de poids faible et que l'on termine par l'octet de poids fort **[3]**, contrairement à l'écriture habituelle. C'est le rôle de la procédure **ecrire_little_endian()** qui utilise un masque binaire **& 0x000000FF** pour récupérer l'octet de poids le plus faible (ligne 17), qui l'écrit dans le fichier WAV ligne 18 puis qui décale de huit positions vers la droite les bits restants (**>> 8**) ligne 19, jusqu'à ce que tous les octets composant l'entier aient été écrits :

```
13: void ecrire_little_endian(unsigned int octets, int taille,
FILE *fichier) {
14:     unsigned char faible ;
15:
16:     while(taille > 0) {
17:         faible = octets & 0x000000FF ;
18:         fwrite(&faible, 1, 1, fichier) ;
19:         octets = octets >> 8 ;
20:         taille = taille - 1 ;
21:     }
22: }
```

1.3 En-tête du fichier WAV

Avant que nous ne continuions à parcourir le code source, vous pouvez compiler et exécuter le programme, sans oublier l'option **-lm** qui est nécessaire pour lier le programme à la librairie mathématique du système :

```
$ gcc synthe.c -lm
$ ./a.out
```

Vous pouvez obtenir des informations sur le format du fichier de sortie **synthe.wav** avec la commande suivante :

```
$ file synthe.wav
synthe.wav: RIFF (little-endian) data, WAVE audio, Microsoft PCM,
16 bit, stereo 44100 Hz
```

La commande **hexdump** permet de visualiser facilement le contenu de ce fichier binaire :

```
$ hexdump synthe_a.wav -C | head
00000000  52 49 46 46 e4 7f a1 00  57 41 56 45 66 6d 74 20
|RIFF....WAVEfmt |
00000010  10 00 00 00 01 00 02 00  44 ac 00 00 10 b1 02 00
|........D.......|
00000020  04 00 10 00 64 61 74 61  c0 7f a1 00 00 00 00 00
|....data........|
00000030  04 00 04 00 01 00 01 00  ee 17 ee 17 c2 1f c2 1f
|................|
```

```
00000040  77 27 77 27 04 2f 04 2f  61 36 61 36 88 3d 88 3d
|w'w'././a6a6.=.=|
00000050  71 44 71 44 16 4b 16 4b  6e 51 6e 51 75 57 75 57
|qDqD.K.KnQnQuWuW|
00000060  24 5d 24 5d 75 62 75 62  63 67 63 67 ea 6b ea 6b  |$]$]
ububcgcg.k.k|
00000070  03 70 03 70 ac 73 ac 73  e1 76 e1 76 9e 79 9e 79
|.p.p.s.s.v.v.y.y|
00000080  e1 7b e1 7b a7 7d a7 7d  ee 7e ee 7e b7 7f b7 7f
|.{.{.}.}.~.~....|
00000090  fe 7f fe 7f c5 7f c5 7f  0c 7f 0c 7f d2 7d d2 7d
|.............}.}|
```

L'option **-C** (pour canonique) permet d'afficher les adresses (ou *offsets*) en hexadécimal à gauche, les octets en hexadécimal au milieu et les caractères correspondants à droite. Le tout est envoyé à la commande **head** à l'aide d'une *pipe,* afin de n'afficher que les dix premières lignes. Les 44 octets de l'en-tête précédant les données sont ici mis en couleur.

Nous découvrons à l'aide de ces commandes qu'un fichier WAV utilise le format RIFF (*Ressource Interchange File Format*) publié par Microsoft et IBM en 1991 **[4]**. Ce format peut en fait contenir des données audio ou vidéo sous différents formats. Une de ses caractéristiques est d'être organisé en *chunks* (terme que je traduirai par blocs) et *sub-chunks.* Chaque bloc commence par un tag de quatre caractères, suivi d'un entier sur quatre octets codant la taille en octets des données contenues à sa suite dans ce bloc.

Après avoir défini ou calculé les paramètres du fichier WAV, notre procédure **ecrire_en_tete_WAV()** va ainsi commencer ligne 35 l'écriture du fichier par le tag **RIFF**. On écrit ensuite sur quatre octets la taille de ce qui va suivre, c'est-à-dire les 36 octets terminant l'en-tête plus le nombre d'octets de données.

```
24: void ecrire_en_tete_WAV(FILE *fichier) {
25:     unsigned short int nb_canaux = 2 ;
26:     unsigned short int nb_bits = 16 ;
27:     taux = 44100 ; // En Hz
28:     duree = 60 ; // En secondes
29:     taille = taux * duree ;
30:     unsigned long int nb_octets_donnees = (nb_bits / 8) *
nb_canaux * taille ;
31:
32:     gauche = calloc(taille, sizeof(double)) ;
33:     droite = calloc(taille, sizeof(double)) ;
34:
35:     fwrite("RIFF", 4, 1, fichier) ;
36:     ecrire_little_endian(36 + nb_octets_donnees, 4, fichier) ;
```

On a également alloué, lignes 32 et 33, la mémoire nécessaire pour stocker les échantillons sonores dans les tableaux de réels double précision **gauche** et **droite**. La fonction **calloc()** initialise également le contenu de ces tableaux avec des octets nuls. En toute rigueur, la norme du C ne garantit pas que cela corresponde à la représentation du réel zéro, mais c'est le cas pour tous les microprocesseurs respectant la norme IEEE 754.

Ligne 37, on écrit ensuite le tag **WAVE** qui indique que le fichier va contenir des données audio au format WAVE (*Waveform audio file format*), plus communément appelé WAV d'après son extension **[5**, **6]**. Suit un sous-bloc commençant par le tag **fmt** (avec comme quatrième caractère un espace), suivi du nombre d'octets terminant ce sous-bloc (16 octets, **0x10** en hexadécimal). Le format audio proprement dit est ensuite codé ligne 41 sur deux octets : le WAV est en fait un conteneur pouvant contenir des données audio codées dans différents formats, éventuellement même en MP3. Nous utilisons ici le codage 1 (le plus courant) : le LPCM (*Linear Pulse-Code Modulation*), appelé plus simplement PCM, qui est également utilisé dans les CD audio et que nous décrirons plus loin.

```
37:     fwrite("WAVE", 4, 1, fichier) ;
38:
39:     fwrite("fmt ", 4, 1, fichier) ;
40:     ecrire_little_endian(16, 4, fichier) ;
41:     ecrire_little_endian(1, 2, fichier) ;
```

Viennent ensuite :

- le nombre de canaux, ici deux pour la stéréo (ligne 42) ;
- le taux (ou fréquence) d'échantillonnage, ici 44100 échantillons par seconde (**0x0002B110** en hexadécimal, ce qui donne **10 b1 02 00** en *little endian*), comme dans un CD audio ;
- le nombre d'octets par seconde (ligne 44), valeur proportionnelle au taux d'échantillonnage, au nombre de canaux et au nombre de bits utilisés pour stocker chaque échantillon, ici 16 bits (**0x10** en hexadécimal) comme dans un CD audio ;
- le nombre d'octets par échantillon (ligne 45), calculé à partir du nombre de canaux et du nombre de bits ;
- le nombre de bits par échantillon.

```
42:     ecrire_little_endian(nb_canaux, 2, fichier) ;
43:     ecrire_little_endian(taux, 4, fichier) ;
44:     ecrire_little_endian(taux * nb_canaux * (nb_bits / 8), 4,
fichier) ;
45:     ecrire_little_endian(nb_canaux * (nb_bits / 8), 2,
fichier) ;
46:     ecrire_little_endian(nb_bits, 2, fichier) ;
```

On remarquera au passage qu'il y a une certaine redondance dans ces données, certaines pouvant être calculées à partir d'autres (lignes 44 et 45).

1.4 Les données

Nous arrivons enfin au sous-bloc contenant les données et qui commence donc par le tag **data**, suivi par la taille des données qui vont suivre (ligne 49). Cette taille étant codée par un entier sur quatre octets sera donc limitée à 4 Gio.

```
48:     fwrite("data", 4, 1, fichier) ;
49:     ecrire_little_endian(taille * nb_canaux * (nb_bits / 8),
4, fichier) ;
50: }
```

La numérisation du son s'opère par échantillonnage (figure 1) : son amplitude est mesurée ou définie à des intervalles réguliers et codée sous forme d'un entier signé proportionnel à cette amplitude **[7, 8]**. C'est le format PCM déjà évoqué. On utilise ici deux octets (16 bits) pour le canal gauche suivis de deux octets pour le canal droit, et ce 44100 fois par seconde. Dans notre programme, la procédure **ecrire_donnees_normalisees_WAV()** s'occupe de normaliser le volume sonore et de le numériser sous forme d'un entier signé sur 2 octets, donc compris au maximum entre **-32768** à **+32767** (-2^{15} et $2^{15}-1$). La constante **AMPLITUDE_MAXI** est déclarée ligne 6 et vaut **32767**. Les lignes 56 à 59 sont chargées de trouver le niveau sonore maximum sur l'ensemble des deux canaux. L'écriture des entiers dans le fichier WAV se fait en *little-endian* lignes 62 et 63 :

```
52: void ecrire_donnees_normalisees_WAV(FILE *fichier) {
53:     unsigned int i ;
54:     double maxi = 1e-16 ;
55:
56:     for (i = 0 ; i < taille ; i=i+1) {
57:         if (fabs(gauche[i]) > maxi)  maxi = fabs(gauche[i]) ;
58:         if (fabs(droite[i]) > maxi)  maxi = fabs(droite[i]) ;
59:     }
60:
61:     for (i = 0 ; i < taille ; i=i+1) {
62:         ecrire_little_endian((int)(gauche[i]/maxi*AMPLITUDE_
MAXI), 2, fichier) ;
63:         ecrire_little_endian((int)(droite[i]/maxi*AMPLITUDE_
MAXI), 2, fichier) ;
64:     }
65: }
```

L'espace disque utilisé par un fichier WAV stéréo à 44100 Hz et 16 bits (qualité d'un CD) sera donc de 44100*2*(16/8)=176400 octets par seconde, soit 10,6 Mio/min. Vous pouvez compresser vos fichiers WAV en MP3 (ou en Ogg Vorbis) avec la commande **avconv** ou **ffmpeg** :

```
$ avconv -i monfichier.wav monfichier.mp3
```

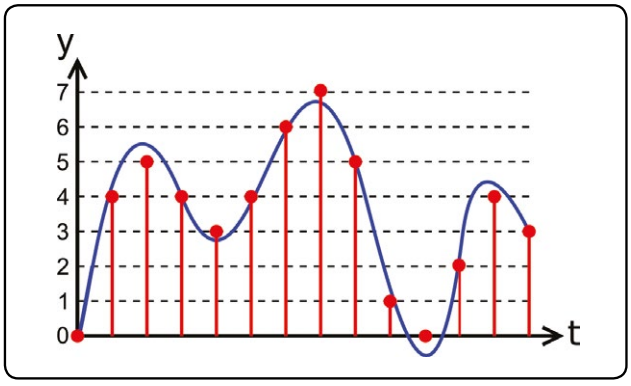


Fig. 1 : Échantillonnage d'un signal.

2 Synthèse d'un son sinusoïdal

Après écriture de l'en-tête du fichier WAV, le programme principal appelle ligne 111 la procédure **mon_signal()** qui va remplir les tableaux de réels **gauche[]** et **droite[]** avec de simples signaux sinusoïdaux **Amp * sin(omega*t + phi)**, avec ici une phase **phi** nulle, commençant à l'instant **t1** et se terminant à l'instant **t2**, ces deux tableaux constituant en fait une piste dont la durée est indiquée dans la variable globale **duree**. Vous pouvez ainsi disposer d'autant de sons que vous voulez sur cette piste et même les superposer, car la procédure **mon_signal()** ajoute en fait le signal aux données déjà présentes dans les tableaux.

```
67: void mon_signal(double t1, double t2, double f, double Amp) {
68:     unsigned int i;
69:     double omega = 2 * PI * f ;
70:     double t = 0 ;
71:     double dt = 1.0 / taux ;
72:     double phi = 0 ;
73:
74:     for (i=(unsigned int)(t1*taux) ; i<(unsigned int)(t2*taux) ;
i=i+1) {
75:         gauche[i] = gauche[i] + Amp * sin(omega*t + phi) ;
76:         droite[i] = droite[i] + Amp * sin(omega*t + phi) ;
77:         t = t + dt ;
78:     }
79: }
```

Rappelons que **omega=2*PI*f** est la pulsation, exprimée en radians par seconde.

Remarquez dans le résultat de la commande **hexdump** présentée plus haut, que les quatre premiers octets de données, juste après l'en-tête du fichier, valent zéro puisque notre programme commence la sinusoïde en un nœud. Et les octets suivants se répètent deux par deux puisqu'ici nous générons deux canaux stéréo identiques.

Vous pouvez écouter le fichier résultat **synthe.wav** avec votre lecteur habituel ou en utilisant la commande **play** après installation du paquet de **SoX** :

```
$ play synthe.wav

synthe.wav:

File Size: 10.6M     Bit Rate: 1.41M
 Encoding: Signed PCM
 Channels: 2 @ 16-bit
Samplerate: 44100Hz
Replaygain: off
 Duration: 00:01:00.00

In:1.55% 00:00:00.93 [00:00:59.07] Out:41.0k [-=====|=====-]
Hd:5.0 Clip:0
```

Vous obtenez un son très pur et vous pourrez visualiser la sinusoïde en zoomant dans Audacity. Rien de très excitant pour l'instant, mais sachez que la sinusoïde est un peu à la musique ce que l'alphabet est à la littérature !

Conclusion

Nous avons fait le tour du programme, à part la procédure **generateur_enveloppe()** que nous aborderons la prochaine fois, l'enveloppe étant un concept important de la synthèse sonore.

Nous avons découvert comment étaient codés les fichiers WAV les plus courants, mais sachez qu'ils peuvent également contenir plusieurs pistes ainsi que des métadonnées. Ils sont même utilisés par certains logiciels d'électronique pour stocker des ondes échantillonnées non acoustiques. Bien que le standard WAV présente quelques défauts tels que des ambiguïtés ou une certaine redondance des données dans l'en-tête, il reste très utilisé dans les logiciels de manipulation du son et il est assez facile d'écrire un programme écrivant ou lisant ce format.

Les pionniers de la musique électronique des années 50 travaillaient bien sûr avec des circuits analogiques. Quant à nous, avec un court programme en C nous pouvons partir sur leurs traces et générer des sons numériques dont nous maîtrisons parfaitement chaque paramètre. Et tel était l'objectif de Stockhausen : pouvoir travailler avec des sons parfaitement maîtrisés, puisque synthétisés **[9]**. Nous verrons dans un prochain article que la synthèse sonore permet effectivement de mieux comprendre ce qu'est un harmonique, un timbre, un accord, etc. Et que l'on peut créer des sons étonnants avec finalement une grande économie de moyens ! ■

Références

[1] « *Langage C, le guide pour mieux développer en C sous Linux* », *GNU/Linux Magazine HS n°70.*

[2] CHAZALLET S., « *Langage C, le guide pour apprendre à programmer en C en 5 jours* », *GNU/Linux Magazine HS n°80.*

[3] Codage Little Endian : https://fr.wikipedia.org/wiki/Endianness

[4] Le format RIFF : https://en.wikipedia.org/wiki/Resource_Interchange_File_Format

[5] Le format WAV : https://en.wikipedia.org/wiki/WAV

[6] Spécifications du format WAV : http://www-mmsp.ece.mcgill.ca/Documents/AudioFormats/WAVE/WAVE.html

[7] Collectif, « *Le son numérique* », *Linux Pratique HS n°29*, p. 8 à 15.

[8] Codage PCM : https://en.wikipedia.org/wiki/Pulse-code_modulation

[9] FICHET L., *Les théories scientifiques de la musique aux XIXe et XXe siècles*,Vrin, 1996, ISBN 978-2-7116-4284-7.

Pour aller plus loin

Si vous n'avez pas la patience d'attendre le second article, vous pouvez consulter les documents suivants :

- BENSON D., « *Music: A Mathematical Offering* », 2008, livre téléchargeable gratuitement en PDF sur son site : http://homepages.abdn.ac.uk/mth192/pages/html/maths-music.html
- RISSET J.-C., « *COMPUTER MUSIC: WHY ?* » : https://www.utexas.edu/cola/france-ut/_files/pdf/resources/risset_2.pdf

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 :	

LINUX MAGAZINE / FRANCE

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

Vos remarques :

.....

.....

☐ 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.

FORMAT WAV : DES SONS DE PLUS EN PLUS COMPLEXES

Vincent Magnin [Maître de conférences à Polytech Lille et l'IEMN, administrateur du projet gtk-fortran]

Dans le précédent article, nous avons autopsié le format WAV et écrit un programme en C permettant de créer un son sinusoïdal pur. Comme promis, partons maintenant sur les pas des pionniers de la synthèse sonore et de la musique électronique !

Mots-clés : Format WAV : des sons de plus en plus complexes

Résumé

À l'aide du programme en C décrit dans le premier article, nous générons des sons de plus en plus complexes avec une grande économie de moyens. Nous commençons par étudier les harmoniques en synthèse additive, puis l'importance de l'enveloppe du signal pour produire différents effets. Nous montrons enfin l'apport de la synthèse par modulation de fréquence avant d'évoquer d'autres méthodes et d'ouvrir quelques perspectives.

Dans l'article précédent, nous nous étions arrêtés à la synthèse d'un son sinusoïdal pur à l'aide du programme `synthe.c`. Nous allons maintenant voir comment nous pouvons créer des sons plus intéressants et explorer différents aspects des théories acoustiques et musicales, en tapant finalement très peu de code supplémentaire. Les sons étudiés dans cet article sont disponibles sur la page **[1]** au format audio Ogg Vorbis.

Nous allons aborder la synthèse additive en sommant des harmoniques puis, nous verrons comment synthétiser un accord majeur. Nous nous intéresserons ensuite à l'enveloppe des sons et à son influence sur le timbre des instruments et nous verrons également comment spatialiser le son. Nous aborderons la modulation de fréquence, qui permet d'obtenir très facilement des sons très complexes. Enfin, nous évoquerons d'autres méthodes de synthèse sonore.

1 Synthèse additive

Rappelons que tout signal périodique peut être décomposé en une somme de sinusoïdes (séries de Fourier) **[2]** : la première correspond à la fréquence fondamentale `f`, ou harmonique de rang 1, les suivantes correspondent aux harmoniques dont les fréquences sont des multiples de `f`. Lorsque l'on joue une note sur un instrument réel, un grand nombre d'harmoniques sont générés et ils vont avoir un rôle important dans la formation du timbre de l'instrument. Enfin, en règle générale, plus les harmoniques sont de rang élevé plus ils s'amortissent rapidement.

Dans la procédure `mon_signal()`, remplacez le signal sinusoïdal par cette boucle qui somme un fondamental et ses six premiers harmoniques, avec de plus des facteurs d'amortissement dépendant de l'ordre `j` de l'harmonique (la fonction mathématique `pow(t,j)` renvoie `t` à la puissance `j`) :

```
for(int j=1 ; j<=7 ; j=j+1){
    gauche[i] = gauche[i] + Amp/
(j*(1+pow(t,j)))*sin(j*omega*t) ;
    droite[i] = gauche[i] ;
}
```

Dans le programme principal, réglez la fréquence à 110 Hz. Après compilation et exécution, ouvrez le fichier **synthe.wav** dans Audacity. Vous obtiendrez la forme d'onde, avec le temps en abscisse et l'amplitude en ordonnée (figure 1), pour les deux voix qui sont ici identiques. À l'échelle de la seconde, vous ne voyez que l'enveloppe du signal. Si vous zoomez à l'échelle de la dizaine de millisecondes, vous voyez le signal lui-même, qui est ici proche d'un signal en dents de scie descendantes (augmentez le nombre d'harmoniques et vous vous en approcherez encore plus). Mais cette forme de visualisation nous apprend finalement peu de choses sur le son que nous percevons.

Le sonagramme (figure 2), ou sonogramme, va nous en apprendre beaucoup plus **[3]**. L'abscisse représente toujours le temps, mais l'ordonnée est ici la fréquence du signal et son intensité est codée par une palette de couleurs. Dans Audacity, il suffit de cliquer sur le petit menu déroulant présent en haut à gauche de la fenêtre contenant la piste sonore et de choisir ***Spectre logarithmique décimal*** au lieu de ***Forme d'onde***. À noter que dans la version 2.0.6 que j'ai utilisée, il semble y avoir un bug au niveau de l'échelle des fréquences, qui est fausse quand on passe dans ce mode. Mais il suffit de zoomer puis de dézoomer sur cette échelle pour obtenir une échelle correcte. Dans ce sonagramme, nous voyons clairement les six harmoniques et le fondamental à 110 Hz, et leur décroissance d'autant plus rapide qu'ils sont de rang élevé.

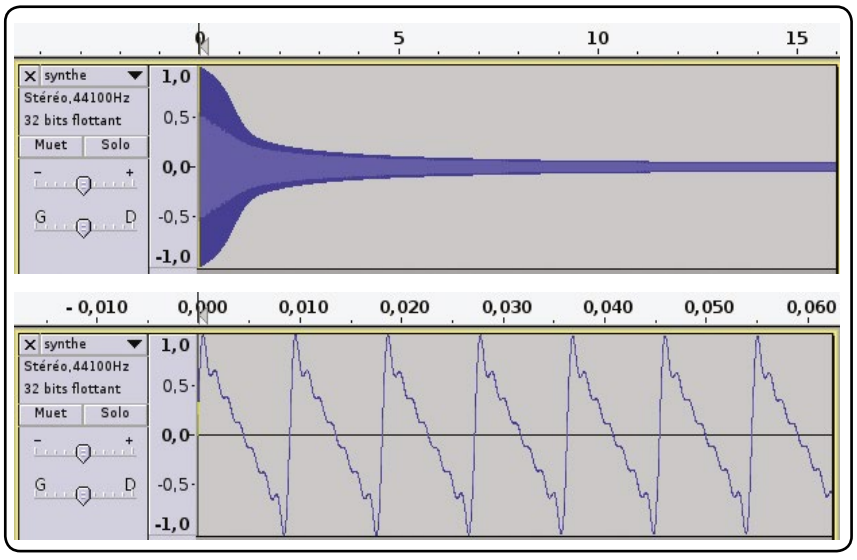


Fig. 1 : En haut, canal gauche à l'échelle de la seconde et en bas, zoom sur les premières dizaines de millisecondes.

Note

Par défaut, la résolution fréquentielle du sonagramme est peu élevée (256) dans Audacity. Vous pouvez l'améliorer en allant dans le menu ***Édition > Préférences... > Spectrogrammes > Définition d'affichage*** et la fixer par exemple à 8192.

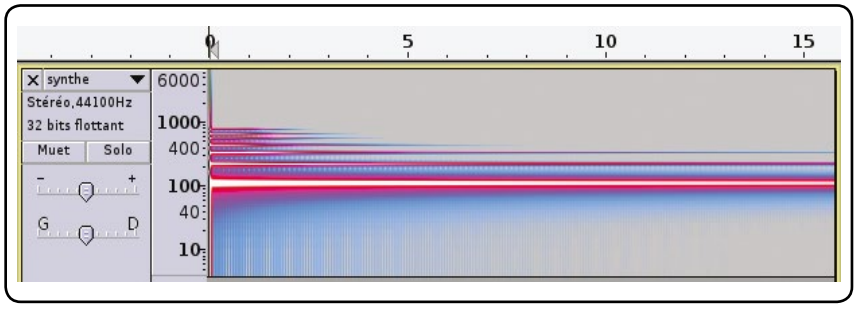


Fig. 2 : Sonagramme du même son (15 secondes, canal gauche) faisant apparaître le fondamental et six harmoniques.

Audacity permet également d'afficher un simple spectrogramme, grâce au menu ***Analyse > Tracer le spectre...***, où vous lisez cette fois l'intensité en décibels associée à chaque fréquence, sans la dimension temporelle.

Si vous vous limitez aux harmoniques impairs, en incrémentant **j** de deux unités à chaque itération, vous obtiendrez un signal carré. Vous pouvez bien sûr écrire des procédures générant directement des formes d'onde de types dents de scie (montantes ou descendantes), triangle, carré, rectangle, impulsion, bruit, etc. Ce sont les principaux générateurs d'ondes dont disposent les synthétiseurs.

2 Un accord majeur

Note

Si vous voulez créer des mélodies ou des accords avec votre programme, il suffit de connaître la gamme chromatique : *do, do#, ré, ré#, mi, fa, fa#, sol, sol#, la, la#, si, do.* Passer d'une octave à l'octave supérieure revient simplement à doubler la fréquence de la note : par exemple le *la 3* de référence (troisième octave) a une fréquence de 440 Hz et le *la 4* de 880 Hz. Dans la gamme chromatique tempérée, pour passer d'une note à la suivante (on parle de demi-ton), il suffit de multiplier la fréquence par $2^{1/12}$. Par exemple, le *la# 3* a une fréquence de $440*2^{1/12}$=466,16 Hz. Finalement, un clavier musical est en quelque sorte une échelle logarithmique !

Si vous jouez un accord de plusieurs notes, d'après la théorie de Helmholtz **[2]** le fait que certains de leurs harmoniques soient très proches va être déterminant pour la qualité de l'accord (majeur, mineur, etc.). Dans le cas d'un accord *do-mi-sol,* on voit par exemple dans le tableau suivant que l'harmonique 3 du *do* est à moins de 1 Hz de l'harmonique 2 du *sol* et que l'harmonique 4 du *sol* est à moins de 2 Hz de l'harmonique 6 du *do* (tableau ci-dessous).

Quand les harmoniques sont trop proches, ils sont d'ailleurs confondus dans le sonagramme d'Audacity. Par contre, votre oreille sera capable de percevoir certains battements résultants.

Voici le bout de code qui vous permettra, en s'arrêtant ici à l'harmonique de rang 7, de créer un bel accord parfait majeur, sachant que le *mi* est quatre demi-tons au-dessus du *do* et le *sol* sept demi-tons au-dessus :

```
for(j=1 ; j<=7 ; j=j+1){
    gauche[i] = gauche[i] + Amp/(pow(j, 2)*(1+pow(t,j)))*(sin(j*omega*t)
+ sin(j*omega*pow(2.0, 4/12.0)*t) + sin(j*omega*pow(2.0, 7/12.0)*t)) ;
}
```

Veillez bien à écrire `4/12.0` ou `4/(double)12` pour éviter que le compilateur ne code une division euclidienne, dont le résultat serait bien sûr nul !

3 Enveloppe

3.1 Percussions

Les sons que nous venons d'aborder commencent brutalement pour décroître ensuite en suivant une loi mathématique du type `1/(j+pow(t,j))`, où `j` est le rang de l'harmonique. Vous pouvez essayer d'autres formules comme `t/(j+pow(t,j))` ou `exp(-j*t)`. Le sinus cardinal donne également un résultat intéressant, ressemblant à une percussion :

```
phi = -2*PI ;
if (omega*t+phi != 0.0) {
    gauche[i] = gauche[i] + Amp * sin(omega * t + phi) /
(omega*t+phi) ;}
else {
    gauche[i] = gauche[i] + Amp ; // sinc(0)=1
}
```

Voilà déjà de quoi programmer une petite boîte à rythmes à l'aide de quelques boucles ! Si vous descendez en dessous de 150 Hz, vous remarquerez néanmoins qu'il vaut mieux avoir des enceintes correctes ou un casque. Le sonagramme montrerait que durant les trois premiers dixièmes de seconde le spectre est très large et continu, ce qui est caractéristique des bruits.

Notez que le son perçu différera sensiblement selon la valeur de la phase `phi` choisie. Pour `phi=0`, on entend un clic désagréable au tout début. Vous pouvez essayer différents multiples de `PI`, positifs ou négatifs.

3.2 Enveloppe ADSR

Pierre Schaeffer avait dans ses expérimentations mis en évidence l'importance de l'attaque dans le timbre de l'instrument et sa reconnaissance par l'auditeur. Une attaque brutale correspond bien aux instruments frappés ou pincés

	1	2	3	4	5	6	7	8	9
do	261,6	523,3	784,9	1046,5	1308,1	1569,8	1831,4	2093,0	2354,6
mi	329,6	659,3	988,9	1318,5	1648,1	1977,8	2307,4	2637,0	2966,6
sol	392,0	784,0	1176,0	1568,0	1960,0	2352,0	2744,0	3136,0	3528,0

Tableau 1 : Fréquences des harmoniques dans un accord majeur do-mi-sol.

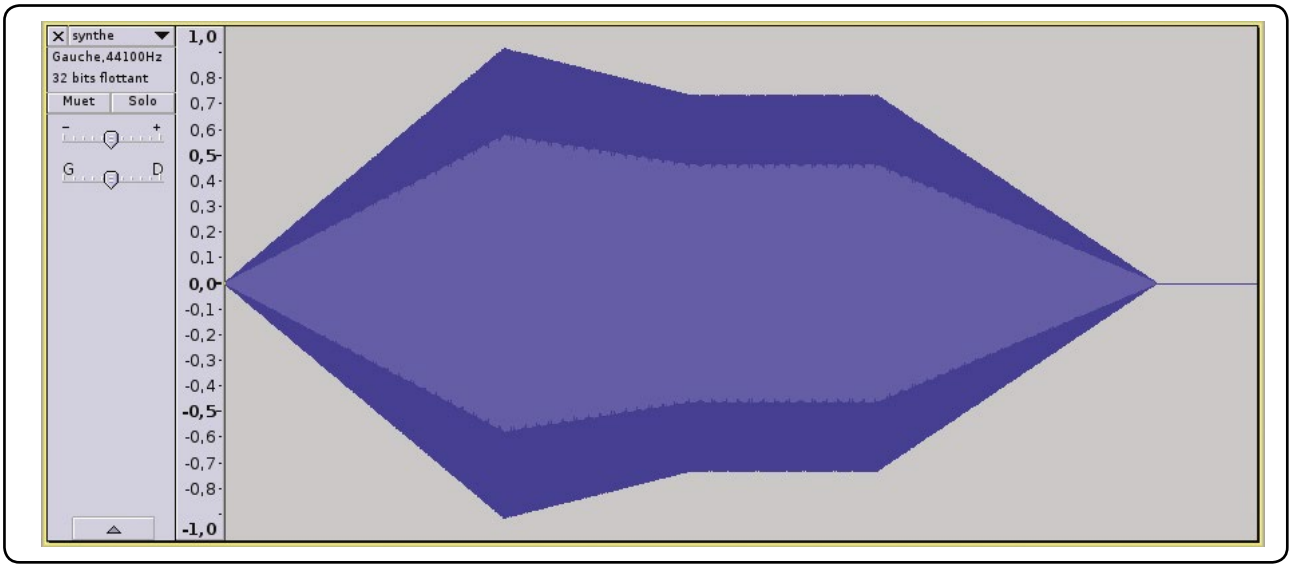


Fig. 3 : Enveloppe ADSR appliquée à un son.

(percussions, guitare, piano, etc.). Pour les instruments à vent ou à cordes frottées, une attaque longue est nécessaire. Il est très courant en synthèse sonore d'utiliser le modèle ADSR (figure 3), implémenté dans notre programme par la procédure **generateur_enveloppe()**. L'enveloppe est appliquée à la piste sonore entre les instants **t1** et **t2** et comprend quatre paramètres :

- l'attaque (**attack**) donnée en pourcentage de la durée **t2-t1** ;
- le déclin (**decay**) donné en pourcentage de la durée **t2-t1** ;
- le maintien (**sustain**) donné en pourcentage par rapport à la valeur pic de l'amplitude (la durée du maintien peut être déduite des trois autres paramètres) ;
- le relâchement (**release**) donné en pourcentage de la durée **t2-t1**.

Chaque partie de l'enveloppe ADSR est traitée par une boucle, après calcul des indices nécessaires :

```
81: void generateur_enveloppe(double t1, double t2, double attack, double
decay, double sustain, double release) {
82:     unsigned int i ;
83:     unsigned int i1 = t1 * taux ;
84:     unsigned int i2 = (t1 + (t2-t1) * attack / 100) * taux ;
85:     unsigned int i3 = (t1 + (t2-t1) * (attack+decay) / 100) * taux ;
86:     unsigned int i4 = (t2 - (t2-t1) * release / 100) * taux ;
87:     unsigned int i5 = t2 * taux ;
88:
89:     for (i=i1 ; i<i2 ; i=i+1) {
90:         gauche[i] = gauche[i] * (i-i1)/(double)(i2-i1) ;
91:         droite[i] = droite[i] * (i-i1)/(double)(i2-i1) ;
92:     }
93:     for (i=i2 ; i<i3 ; i=i+1) {
94:         gauche[i] = gauche[i] * (100 - (i-i2)/(double)(i3-
i2)*(100-sustain))/100 ;
95:         droite[i] = droite[i] * (100 - (i-i2)/(double)(i3-
i2)*(100-sustain))/100 ;
96:     }
97:     for (i=i3 ; i<i4 ; i=i+1) {
98:         gauche[i] = gauche[i] * (sustain / 100) ;
99:         droite[i] = droite[i] * (sustain / 100) ;
100:     }
101:    for (i=i4 ; i<i5 ; i=i+1) {
102:        gauche[i] = gauche[i] * (sustain - (i-i4)/(double)
(i5-i4)*sustain) / 100 ;
103:        droite[i] = droite[i] * (sustain - (i-i4)/(double)
(i5-i4)*sustain) / 100 ;
104:    }
105: }
```

Essayez par exemple dans la procédure **mon_signal()**, de remplacer votre sinusoïde par ce son ne contenant que des harmoniques impairs :

```
for(j=1 ; j<=11 ; j=j+2){
    gauche[i] = gauche[i] + Amp/pow(j, 0.7)*sin(j*omega*t) ;
}
```

Puis dans le programme principal, appelez cette procédure pour créer un son de trois secondes à 220 Hz et appliquez ensuite une enveloppe :

```
mon_signal(0.0, 3.0, 220.0, 1.0) ;
generateur_enveloppe(0.0, 3.0, 30.0, 20.0, 80.0, 30.0) ;
```

Le résultat évoque une clarinette. Si au lieu d'appliquer cette enveloppe ADSR, vous faites décroître au cours du temps le niveau en multipliant le signal par un facteur

`exp(-sqrt(j)*t)`, vous remarquerez que votre son n'est plus perçu comme ressemblant à une clarinette.

À noter également que notre procédure applique l'enveloppe à l'ensemble du signal, alors que les pionniers de la synthèse sonore s'étaient rendu compte qu'il pouvait être nécessaire d'avoir une enveloppe différente pour chaque harmonique afin de synthétiser correctement en particulier les instruments à vent ou à cordes frottées (Figure 3, page précédente).

3.3 Trémolo

On utilise beaucoup dans les synthétiseurs des oscillateurs basses fréquences (LFO, pour *Low Frequency Oscillator*) pour faire varier certaines grandeurs. Faire varier par exemple l'amplitude du son avec une fréquence de 5 à 10 Hz produira un trémolo :

```
gauche[i] = gauche[i] *
(1.0-AmpLFO*sin(omegaLFO*t)) ;
```

Ici, on module l'amplitude du signal déjà existant dans le canal gauche avec une pulsation lente `omegaLFO` et une profondeur `AmpLFO` (par exemple 0,1). En théorie du signal, c'est de la modulation d'amplitude (AM), le son de départ jouant le rôle de porteuse dont on module l'enveloppe.

Attention, si la fréquence de modulation est beaucoup plus élevée, cette modulation sera perçue par l'oreille d'une façon très différente, avec l'apparition de signaux aux deux fréquences éloignées du fondamental de cette valeur, conformément à la formule de trigonométrie `sin(a)*sin(b)=(cos(a-b)-cos(a+b))/2`.

3.4 Spatialisation du son

Dans tous les exemples donnés jusqu'ici, nous nous sommes contentés d'assigner au canal droit les mêmes valeurs qu'au canal gauche. Mais le programme est conçu pour que vous puissiez aussi jouer de la stéréo. Faire par exemple tournoyer le son n'est pas plus difficile que de moduler le signal à l'aide d'un cosinus d'un côté et d'un sinus de l'autre, afin que le signal soit maximum d'un côté quand il est minimum de l'autre :

```
gauche[i] = gauche[i] * (1.0-AmpLFO*sin(omegaLFO*t + phi)) ;
droite[i] = droite[i] * (1.0-AmpLFO*cos(omegaLFO*t + phi)) ;
```

Ici `omegaLFO` est la pulsation du tournoiement, par exemple `2*PI*f` avec f=0,5 Hz, et `AmpLFO` la profondeur de modulation entre 0 et 1. La phase `phi` permet de régler le point de départ.

En améliorant le programme pour travailler avec plus de canaux dans le fichier WAV, par exemple en 5.1, on pourrait facilement déplacer le son dans l'espace et étudier nous aussi, comme Stockhausen, la spatialisation du son.

4 Modulation de fréquence ou de phase

Quand on parle de modulation de fréquence (FM), il s'agit souvent en fait d'une modulation de phase (PM) **[2]**. En modulant la phase de la sinusoïde à une fréquence d'environ 4 Hz, vous obtiendrez un effet de vibrato :

```
gauche[i] = gauche[i] + Amp * sin(omega * t + phi*(1.0+0.5*sin(2*PI*4.0*t))) ;
```

Mais si la modulation se fait à des fréquences similaires à celles des signaux acoustiques, vous rentrerez dans le domaine de la synthèse dite FM. Voici par exemple un son de type bourdonnement :

```
gauche[i] = gauche[i] + Amp * sin(omega*t + phi*sin(omega*0.2*t)) ;
```

La synthèse FM a pour effet de faire apparaître un large spectre de fréquences (figure 4, ci-contre).

À vous d'imaginer des formules mathématiques générant des sons intéressants, comme (à faire durer au moins 50 secondes) :

```
gauche[i] = gauche[i] + Amp * sin(omega*t + phi*cos(75*t)/(1+log(t/100+0.01))) ;
```

Vous pouvez aussi moduler véritablement la fréquence :

```
gauche[i] = gauche[i] + Amp * sin(omega*(1.0+0.001*sin(t*50)) * t + phi) ;
```

L'effet est souvent contre-intuitif : dans cet exemple, la profondeur de modulation n'est que d'un millième, mais l'effet sur le son devient au fil des secondes de plus en plus énorme ! Et en remplaçant le facteur 50 par 500, vous obtenez un son très intéressant, qui mériterait une enveloppe ADSR bien ficelée pour le mettre en valeur. Passez maintenant d'un facteur 500 à un facteur 5000 et vous voilà avec un son dont les premières secondes évoquent un gong.

N'hésitez pas à moduler la modulation (et pourquoi pas la modulation de la modulation !) :

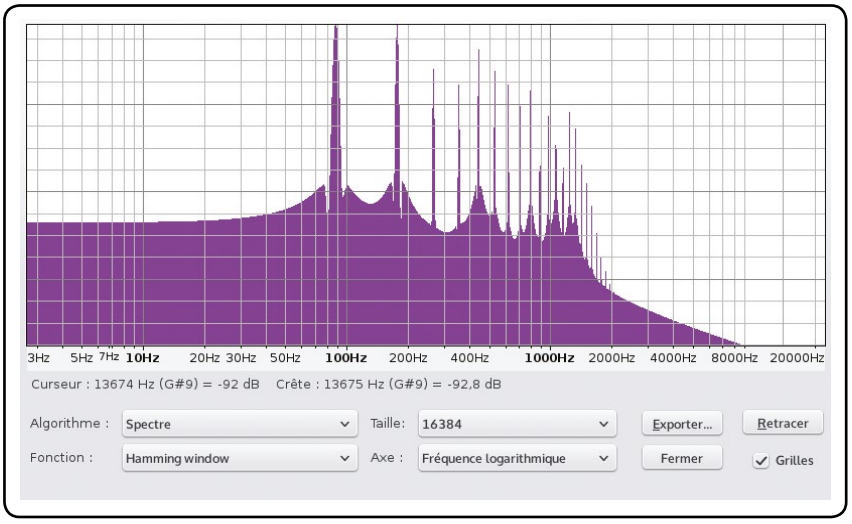


Fig. 4 : Spectrogramme d'un son obtenu par modulation de phase.

```
gauche[i] = gauche[i] + Amp * sin(omega*(1.0+0.001*sin(t*500*(1.0
+0.0002*cos(t*50)))) * t + phi) ;
```

Vous comprenez que le terrain de jeu est infini ! La synthèse FM a été la grande révolution du début des années 80 avec le synthétiseur Yamaha DX7 qui a connu un énorme succès sur la scène pop de l'époque **[2]**.

5 Autres méthodes de synthèse sonore

La synthèse soustractive consiste à sculpter le son en filtrant un signal riche en harmoniques à l'aide par exemple de filtres passe-bas, passe-haut, passe-bande, etc. Mais ces filtres nécessitent des méthodes mathématiques particulières telles que les transformées de Fourier et dépassent le cadre de cet article.

La synthèse physique consiste à modéliser les vibrations d'un objet à l'aide d'équations physiques, par exemple une corde vibrante ou même un instrument complet comme une guitare ou un piano.

Concernant les instruments à cordes pincées, vous pouvez plus simplement vous intéresser à l'étonnant algorithme de Karplus et Strong (1983) qui en à peu près huit lignes de code génère des sons de guitare très convaincants **[2]**. Il consiste à générer un signal aléatoire au début, la suite étant calculée à l'aide d'une relation de récurrence ayant pour effet de faire disparaître progressivement les fréquences élevées.

Conclusion

Nous avons vu comment, à partir d'un court programme et d'une simple sinusoïde, il est possible de créer des sons plus ou moins complexes, en sommant des sinusoïdes, en modifiant l'enveloppe des sons, en modulant certains paramètres de l'onde, et comment l'on pouvait même jouer sur sa spatialisation.

Vous pouvez maintenant vous lancer sur les pas d'Éliane Radigue **[4]**, grande exploratrice de la musique drone, musique qui se caractérise par l'usage de bourdons (*drone* en anglais), sons continus évoluant très lentement au cours de morceaux souvent assez longs (typiquement une heure). Ou tenter de reproduire des illusions acoustiques telles que le glissando de Shepard-Risset **[5, 6]**, série de glissandos entrant et sortant continûment du spectre audible pour donner l'illusion d'une ascension ou d'une descente éternelle, sorte d'équivalent acoustique des escaliers d'Escher.

Pour les plus enthousiastes, l'étape suivante pourrait être de s'attaquer au langage de création sonore Csound **[7]**, logiciel libre initialement développé au MIT... ■

Références

[1] Écoutez les sons de l'article : http://magnin.plil.net/spip.php?article131

[2] BENSON D., « *Music: A Mathematical Offering* », 2008, livre téléchargeable en PDF sur son site http://www.maths.abdn.ac.uk/¤bensondj/html/maths-music.html

[3] Sonagramme : https://fr.wikipedia.org/wiki/Sonagramme

[4] Émission sur Eliane Radigue : http://www.francemusique.fr/emission/le-jour-d-avant/2013-2014/eliane-radigue-ou-l-experience-electronique-11-10-2013-00-00

[5] Glissando de Shepard-Risset : https://en.wikipedia.org/wiki/Shepard_tone

[6] RISSET J.-C., « *COMPUTER MUSIC: WHY ?* », https://www.utexas.edu/cola/france-ut/_files/pdf/resources/risset_2.pdf

[7] Langage de création musicale Csound : http://csound.github.io/


INTERFACES UTILISATEUR EN PYTHON : LE MODE CLI

Tristan Colombo

Lorsque l'on développe un programme ou même un simple script, l'objectif est qu'il soit utilisé le plus possible et par le plus de monde possible. Malheureusement (ou peut-être heureusement), nous n'avons pas tous les mêmes notions d'ergonomie et d'accessibilité...

Mots-clés : Interfaces, CLI, Python, Paramètres, Argparse

Résumé

Cet article présente un petit projet fonctionnant depuis des appels en ligne de commandes. L'objectif est de construire le programme de manière à ce qu'il puisse être appelé depuis différentes IHM (développées par la suite) de façon à s'adapter aux habitudes des utilisateurs.

Cela vous est sans doute déjà arrivé : quelqu'un dans votre entourage ou au travail vous parle d'un problème récurrent qui vous semble très simple à traiter à l'aide d'un petit programme. Vous vous attelez à la tâche et vous obtenez une application fonctionnelle que vous vous empressez de transmettre à ladite personne. Tout ce travail pour vous rendre compte quelque temps plus tard qu'elle ne l'utilise pas du tout et continue de traiter son problème manuellement en y consacrant un temps assez conséquent. Vous sentez monter en vous un profond sentiment de colère... mais en y réfléchissant bien, est-ce vraiment justifié ? N'êtes-vous pas finalement fautif en ayant proposé une application devant être appelée exclusivement en ligne de commandes ?

Nous sommes tous différents : certains préféreront la ligne de commandes, d'autres une interface en mode texte et enfin les derniers ne jureront que par les interfaces graphiques. En tant que développeurs, pouvons-nous vraiment juger l'utilisateur sur ses préférences en matière d'interface ? Ne sommes-nous pas plutôt tenus de nous adapter à ses désirs ?

L'extrémisme et l'intégrisme n'ont jamais apporté de solution à quelque problème que ce soit. Lorsque vous ouvrez une application sous Windows vous allez râler parce qu'il faut tout faire à la souris dans une interface graphique... alors, pourquoi suivre le même travers, sous prétexte que vous utilisez Linux, et imposer une utilisation de la console ? N'oubliez pas que dans logiciel libre, il y a le mot « libre ». J'aime bien me dire que cette liberté va au-delà des quatre libertés essentielles **[1]** et que l'utilisateur doit aussi être libre de choisir l'interface qui lui convient. Bien sûr, cela implique une charge de travail supplémentaire pour le développeur, mais, si l'on y réfléchit un peu plus, vaut-il mieux développer rapidement un programme que peu de monde (voire personne) n'utilisera ou vaut-il mieux y passer un peu plus de temps, mais s'assurer ainsi que la plupart des gens seront en capacité de l'utiliser ?

Dans cet article, nous allons partir d'un projet relativement simple et nous verrons les différentes étapes qui nous conduiront à son achèvement tout en gardant à l'esprit que l'on doit pouvoir lui adjoindre trois modes d'appel : en ligne de commandes, avec une inter-

face en mode texte et enfin avec une interface graphique. Comme il serait trop long de traiter les trois types d'interfaces en un seul article, nous nous attacherons ici à l'interface CLI (*Command-Line Interface*).

1 Le projet

Je vous ai annoncé un projet simple, mais nous allons quand même faire en sorte que celui-ci ait un minimum de fonctionnalités. Il s'agira de créer un petit annuaire dans lequel nous stockerons les nom, prénom et adresse mail des contacts. Notre application permettra d'envoyer un mail à un contact en donnant uniquement son prénom et son nom. Ce mail contiendra un petit message et un fichier PDF qui sera généré automatiquement et qui contiendra :

```
Bonjour <Prénom> <Nom>,

Ceci est un message généré automatiquement.
Aujourd'hui il <fait beau>/<pleut> !
```

La fin de la dernière phrase sera déterminée par l'utilisateur lors de l'appel du programme.

2 Choix techniques

Pour répondre au problème, il va falloir effectuer quelques choix techniques :

- nous développerons notre programme en Python 3.4 (ou plus) ;
- pour stocker les informations des contacts, nous allons utiliser une base de données SQLite3 et le module de même nom. Cette base ne contiendra qu'une seule table dont le schéma est donné en figure 1 ;
- pour envoyer les mails, nous utiliserons les modules **`smtplib`** et **`email`** ;
- nous générerons les fichiers PDF à partir de fichiers LaTeX à l'aide de la commande **`pdflatex`** ;
- pour modifier le contenu du mail (et donc du fichier LaTeX), nous utiliserons le système de gabarits *(templates)* Mustache **[2]** et le module **`pystache`**.

Et comme nous avons dit que nous développerons trois niveaux d'interfaces utilisateur, il va falloir là encore faire des choix. Pour l'instant, nous nous concentrons sur le mode CLI, donc nous utiliserons le traditionnel **`argparse`**.

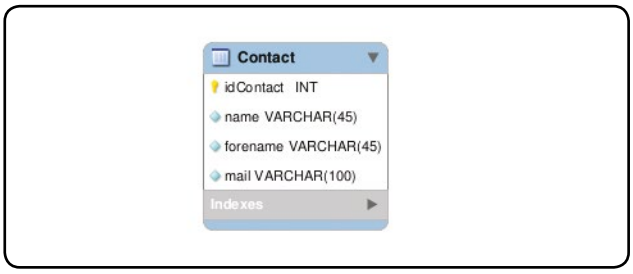


Fig. 1 : Schéma de la base de données (réalisé avec MySQL Workbench).

Nous allons créer un environnement virtuel pour notre projet de manière à pouvoir développer en toute tranquillité :

```
$ pyvenv contacts
$ source contacts/bin/activate
(contacts) $
```

3 La ligne de commandes

Bien évidemment, nous allons commencer par développer le programme en version ligne de commandes. C'est le cœur de notre application et une fois que cela fonctionnera il ne nous restera plus qu'à y greffer une interface en mode texte ou une interface graphique. Et d'ailleurs, si l'on voit plus loin, si l'interface graphique que nous proposons ne convient pas aux utilisateurs, si le programme a été bien pensé, n'importe quel développeur pourra y adjoindre une nouvelle interface.

Pour lancer le programme en mode graphique ou avec une interface textuelle, il faudra également appeler la ligne de commandes. Gérons donc les différents paramètres que nous pourrons lui fournir.

3.1 La gestion des paramètres

Comme nous aurons trois types d'appels, nous allons créer trois sous-types de commandes. Notre programme principal s'appellera **`contacts.py`** et il pourra être appelé par :

- **`python contacts.py cli "Prénom Nom" -s`** : appel direct en ligne de commandes. Il faut indiquer le nom du contact et éventuellement le temps (**`-s`** ou **`--soleil`** pour dire qu'il fait beau et sinon il pleuvra) ;
- **`python contacts.py text`** : interface textuelle donc pas de paramètre à fournir ;
- **`python contacts.py gui`** : interface graphique et là encore aucun paramètre à indiquer.

Partant de ces contraintes, voici le code que nous obtenons :

```
...
14: import argparse
15:
16:
17: def cli_mode(args):
18:     print('Mode CLI')
19:     print('Contact:', args.name)
20:     print('Temps: ', end='')
21:     if args.sun:
22:         print('soleil')
23:     else:
24:         print('pluie')
25:
26:
27: def text_mode(args):
28:     print('Interface textuelle')
29:
30:
31: def gui_mode(args):
32:     print('Interface graphique')
33:
34:
35: if __name__ == '__main__':
36:     """
37:         Création du parser d'arguments pour la ligne
38:         de commandes
39:     """
40:     parser = argparse.ArgumentParser()
41:     subparsers = parser.add_subparsers(help='Commandes')
42:
43:     # Mode CLI
44:     cli_parser = subparsers.add_parser('cli', help='Mode
ligne de commandes')
45:     cli_parser.add_argument('name', help='Prénom et nom du
contact')
46:     cli_parser.add_argument('-s', '--soleil', action='store_
true',
47:                             default=False, dest='sun',
48:                             help='Temps à insérer dans le
fichier pdf')
49:     cli_parser.set_defaults(func=cli_mode)
50:
51:     # Mode texte
52:     text_parser = subparsers.add_parser('text',
53:                                         help="Mode
interface textuelle")
54:     text_parser.set_defaults(func=text_mode)
55:
56:     # Mode graphique
57:     gui_parser = subparsers.add_parser('gui', help="Mode
interface graphique")
58:     gui_parser.set_defaults(func=gui_mode)
59:
60:     args = parser.parse_args()
61:     args.func(args)
```

Les lignes 1 à 12 (supprimées ici) constituent le bloc de commentaires indiquant ce que fait le programme, qui contacter en cas de problème (il ne s'agit pas d'un ego surdimensionné, mais bien d'une aide au développeur qui voudra modifier le code), quand a été modifié le code (nous aurions pu également indiquer sa version) et sous quelle licence il est distribué (sans oublier de fournir avec le code un fichier contenant ladite licence au format texte). Pour pouvoir gérer les paramètres transmis dans la ligne de commandes, nous importons le module **argparse** en ligne 14. Les trois fonctions des lignes 17 à 32 correspondent aux trois modes qui pourront être lancés. J'ai utilisé ici un simple affichage à l'aide d'appels à la fonction **print()** de manière à ne pas alourdir le code, mais pour que ce soit plus propre, il aurait fallu utiliser le module **logging** et créer un fichier de log. Le programme principal se contente d'indiquer quels sont les paramètres admis dans la ligne de commandes en créant différents *parsers*. Par exemple, dans les lignes 43 à 49, il s'agit de la définition de l'appel de la commande **cli** :

- en ligne 44, j'indique que la commande cli sera valide et qu'elle pourra éventuellement contenir des paramètres. Pour cela, je crée le *parser* **cli_parser** ;
- en ligne 45, j'ajoute le paramètre **name** qui sera obligatoire (contient une chaîne de caractères) ;
- les lignes 46 à 48 permettent l'ajout d'une option **-s** (ou **--soleil** dans sa version longue) qui est un booléen (**action='store_true'**) dont la valeur par défaut est **False** ;
- enfin, en ligne 49, j'indique que lorsque cette commande est appelée, il faut exécuter la fonction **cli_mode()** (définie dans les lignes 17 à 24).

Le principe est le même pour les modes texte (lignes 51 à 54) et graphique (lignes 56 à 58). Ces modes sont plus simples puisqu'ils n'admettent pas de paramètres. Enfin, en ligne 60 nous lançons l'analyse de la ligne de commandes et nous appelons la fonction adéquate en lui transmettant les paramètres en ligne 61 (notez que ces paramètres sont des attributs de l'objet **args** comme on peut le voir dans les lignes 19 et 21).

Ce morceau de code est fonctionnel : vous pouvez l'exécuter et, en fonction des paramètres donnés, vous déclencherez l'une des trois fonctions **cli_mode()**, **text_mode()** ou **gui_mode()**.

Comme nous développons ici le mode CLI, nous nous intéresserons désormais à la fonction **cli_mode()**.

3.2 La base de données SQLite3

Comme nous connaissons le nom et le prénom du contact auquel l'utilisateur souhaite envoyer un mail, nous pouvons le rechercher dans la base de données. Mais il y aura deux cas à traiter :

- si la base n'existe pas, il faut la créer ;
- si le contact n'existe pas, il faudra demander à l'utilisateur de saisir l'adresse mail pour pouvoir créer un nouveau contact dans la base.

3.2.1 Ouverture et fermeture de la base

L'ouverture et la fermeture de la base sont des opérations qui seront communes aux trois modes. Nous allons donc les externaliser dans des fonctions que nous insérerons après la ligne 34 :

```
36: def create_db():
37:     base = sqlite3.connect('base.db')
38:     cursor = base.cursor()
39:
40:     try:
41:         cursor.execute("""create table Contact(
42:                         idContact integer primary key,
43:                         name text,
44:                         forename text,
45:                         mail text)""")
46:         base.commit()
47:     except sqlite3.OperationalError:
48:         pass
49:
50:     return (base, cursor)
51:
52:
53: def close_db(args):
54:     args.cursor.close()
55:     args.base.close()
```

Vous aurez sans doute remarqué qu'en ligne 48 je ne traite pas l'exception et vous vous demandez quelle est cette horreur. Ici nous ne pouvons pas afficher de message pour indiquer l'existence de la base : imaginez le résultat avec les autres interfaces... En fait, si nous avions implémenté le fichier de log nous pourrions y inscrire que la base a été ouverte correctement. Quoi qu'il en soit, notez bien qu'il s'agit d'un cas très spécial et qu'il vaut mieux appliquer la règle consistant à traiter obligatoirement toute exception interceptée !

Ces fonctions doivent être appelées dans le programme principal. Comme l'ouverture nous renvoie deux objets (la base et le curseur), nous allons profiter de l'objet **args** de l'analyse des paramètres pour les insérer en tant qu'attributs. Ainsi, ils seront automatiquement transmis à nos fonctions d'interface. Pour la fermeture de la base, il faudra l'exécuter en fin de programme :

```
83:     args = parser.parse_args()
84:     args.base, args.cursor = create_db()
85:     args.func(args)
86:     close_db(args)
```

3.2.2 Ajouter et rechercher un contact

L'utilisateur va nous fournir un nom et un prénom et il faudra aller chercher dans la base si une occurrence existe. Mais il ne faut surtout pas oublier que le code devra pouvoir être réutilisable quelle que soit l'interface choisie ! Pour cela, nous allons créer un objet **Contact** qui aura pour attributs **name**, **forename**, **mail** et les accès à la base de données. Voici le code du fichier **Contact.py** :

```
001: """
002:     Objet Contact représentant un contact de l'Annuaire
...
012: """
013:
014:
015: import sqlite3
016:
017:
018: class Contact:
019:
020:     base = None
021:     cursor = None
022:
023:     def __init__(self, name, db_args):
024:         name = name.split(' ')
025:
026:         if len(name) != 2:
027:             print('Vous devez saisir un prénom suivi du nom')
028:             print('Syntaxe: Prénom Nom')
029:             exit(1)
030:
031:         if Contact.base is None:
032:             Contact.base = db_args[0]
033:         if Contact.cursor is None:
034:             Contact.cursor = db_args[1]
035:         self.forename = Contact.capitalizeName(name[0])
036:         self.name = Contact.capitalizeName(name[1])
037:         self.mail = self.searchMail()
```

Pour l'objet **Contact**, nous définissons trois attributs « classiques » **forename**, **name** et **mail** ainsi que deux

attributs statiques **base** et **cursor** (ces attributs seront partagés par l'ensemble des instances de **Contact** : ils ne seront pas dupliqués en mémoire). Dans les lignes 26 à 29, nous traitons le cas où la chaîne de caractères passée en paramètre ne contient pas de séparateur espace entre le prénom et le nom. Ici encore il faudrait que le message d'erreur passe par un fichier de log, mais nous nous contenterons d'un simple affichage dans le terminal. En ligne 37, nous faisons appel à la méthode **searchMail()**, définie plus loin, pour rechercher dans la base de données le mail du contact. Vous remarquerez dans les lignes 35 et 36 l'appel à une méthode statique **capitalizeName** (car préfixée par le nom de la classe) :

```
039:     @staticmethod
040:     def capitalizeName(name):
041:         capitalized = ''
042:         pred = ''
043:         for letter in name:
044:             if pred in ('', ' ', '-'):
045:                 if letter != ' ':
046:                     capitalized += letter.upper()
047:             else:
048:                 capitalized += letter
049:             pred = letter
050:
051:         return capitalized
```

Cette fonction permet de nettoyer une chaîne transmise par l'utilisateur. En effet, lorsque l'on recherche « jean-marc truc », « Jean-marc Truc » ou « Jean- Marc truc », il s'agit de la même personne. Ici toutes les lettres sont mises en minuscules puis, la première lettre et toute lettre se situant après un caractère espace ou tiret sera mise en majuscule (ligne 46). De même, si plusieurs espaces se suivent, un seul sera conservé (test de la ligne 45).

Il faut ensuite être capable de savoir si un contact est déjà présent dans la base de données, pouvoir l'ajouter et récupérer son adresse mail :

```
053:     def isInDB(self):
054:         result = Contact.cursor.execute("""select name from
Contact
055:                                 where forename = ?
056:                                 and name = ?""", (self.
forename,
057:                                                   self.name))
058:         result = result.fetchone()
059:         return result is not None
060:
061:     def addInDB(self):
062:         try:
063:             Contact.cursor.execute("""insert into Contact
064:                                       values(NULL, ?, ?, ?)""",
065:                                       (self.name, self.
forename, self.mail))
066:             Contact.base.commit()
067:         except sqlite3.OperationalError:
068:             print('Erreur d\'écriture dans la base')
069:             exit(2)
070:
071:     def searchMail(self):
072:         mail = Contact.cursor.execute("""select mail from
Contact
073:                                 where forename = ?
074:                                 and name = ?""", (self.
forename,
075:                                                   self.name))
076:         mail = mail.fetchone()
077:         if mail is None:
078:             return None
079:         else:
080:             return mail[0]
```

Les trois fonctions présentées ici sont surtout basées sur des exécutions de requêtes dans la base de données (lignes 54 à 58, 63 à 66 et 72 à 76). Dans l'écriture des requêtes, il faut penser à utiliser le symbole **?** pour des insertions de texte protégées contre les injections SQL (les données remplaçant le point d'interrogation sont alors transmises en second paramètre sous la forme d'un tuple).

La méthode **fetchone()** employée dans les lignes 58 et 76 permet de récupérer la première ligne des résultats (en l'appelant une deuxième fois on obtient la deuxième ligne et ainsi de suite). Lorsque la requête ne renvoie aucun enregistrement, **fetchone()** vaut **None** et sinon il s'agit d'une liste contenant les éléments récupérés. Dans le cadre de la recherche du mail d'un contact (fonction **searchMail()** des lignes 71 à 80), si le contact n'est pas trouvé, la valeur **None** sera renvoyée.

Il faut ensuite définir les méthodes d'accès et de modifications aux attributs. Pour le mail, ce sera un peu plus complexe que pour le nom et le prénom :

```
082:     def getMail(self):
083:         if self.__mail is None:
084:             return self.searchMail()
085:         else:
086:             return self.__mail
087:     def setMail(self, mail):
088:         if mail is None:
089:             self.__mail = None
```

```
090:        else:
091:            self.__mail = mail.lower()
092:            if self.isInDB():
093:                try:
094:                    Contact.cursor.execute("""update Contact
095:                                set mail = ?
096:                                where name = ?
097:                                and forename = ?""",
098:                                (self.mail, self.name,
self.forename))
099:                    Contact.base.commit()
100:                except sqlite3.OperationalError:
101:                    print('Erreur d\'écriture dans la base')
102:                    exit(2)
103:            else:
104:                self.addInDB()
105:    mail = property(getMail, setMail)
```

Pour obtenir le mail (méthode **getMail()** des lignes 82 à 86), il n'y a rien de bien compliqué : si le mail n'est pas encore connu, on le recherche dans la base (ligne 84) et sinon on renvoie directement sa valeur.

En ce qui concerne la modification du mail, nous distinguerons le cas où le contact est déjà référencé dans la base et où il s'agit effectivement d'une modification d'un champ de l'enregistrement (lignes 93 à 102), du cas où le contact est nouveau et qu'il faut en fait l'ajouter à la base (ligne 104).

Enfin, j'ai utilisé une propriété en ligne 105 de manière à rendre l'appel à ces fonctions entièrement transparent.

Pour les accès et modifications aux attributs **__name** et **__forename**, nous resterons dans du très classique :

```
107:    def getName(self):
108:        return self.__name
109:    def setName(self, name):
110:        self.__name = name
111:    name = property(getName, setName)
112:
113:    def getForename(self):
114:        return self.__forename
115:    def setForename(self, forename):
116:        self.__forename = forename
117:    forename = property(getForename, setForename)
```

Pour utiliser cette classe dans le fichier **contacts.py**, il faut modifier quelques lignes :

```
...
14: import argparse
15: import sqlite3
16: from Contact import Contact
17:
18:
19: def cli_mode(args):
20:     person = Contact(args.name, (args.base, args.cursor))
21:     forename = person.forename
22:     name = person.name
23:     mail = person.mail
24:
25:     if mail == None:
26:         print('Nouveau contact')
27:         mail = input('Veuillez saisir le mail: ')
28:         person.mail = mail
29:
30:     print('Mode CLI')
31:     print('Contact:', forename, name)
32:     print('Mail:', mail)
...
```

Vous voyez que le plus gros du travail a été effectué dans la classe **Contact**. Ici nous nous contentons de créer une instance de cette classe (ligne 20) puis de récupérer les prénom, nom et mail (lignes 21 à 23). Si le mail n'est pas trouvé dans la base, c'est que le contact n'y est pas enregistré et qu'il faut donc saisir son mail en vue de sa sauvegarde (lignes 25 à 28).

Dans sa partie base de données, le code est fonctionnel. Générons maintenant le document PDF.

3.3 La génération du PDF

Avant de pouvoir générer notre document, il faut s'occuper de son gabarit. En effet, nous avions dit que nous utiliserions le module **pystache** héritant de la syntaxe de Mustache pour indiquer dans un fichier LaTeX les zones dans lesquelles insérer des chaînes de caractère. Ce module n'est pas présent par défaut dans Python et il faut donc l'installer :

```
$ pip install pystache
```

3.3.1 Rappels de syntaxe Mustache

La syntaxe employée pour insérer des données dans le document est très simple. Voici les marqueurs que nous emploierons :

- **{{variable}}** : remplace le marqueur par la valeur de **variable** ;
- **{{{variable}}}** : remplace le marqueur par la valeur de **variable** sans interprétation du contenu ;

- **{{^cle}}...{{/cle}}** : si la variable booléenne **cle** vaut **False**, le bloc (**...**) sera conservé ;
- **{{#cle}}...{{/cle}}** : si la variable **cle** vaut **True**, alors le bloc (**...**) sera conservé.

3.3.2 Le gabarit

Notre gabarit, **doc.tex** sera très simple puisqu'il ne contiendra que trois zones d'insertions pour le prénom, le nom et le temps :

```
01: \documentclass[a4paper,12pt]{article}
02:
03: \usepackage[french]{babel}
04: \usepackage[utf8]{inputenc}
05:
06:
07: \begin{document}
08:
09: \indent Bonjour {{{forename}}} {{{name}}},\\
10:
11: \vspace{2cm}
12:
13: \noindent Ceci est un message généré automatiquement.\\
14: \\
15: Aujourd'hui il {{#sun}}fait beau{{/sun}}{{^sun}}pleut{{/sun}} !
16:
17: \end{document}
```

Il s'agit d'un fichier LaTeX de base où les zones d'insertions sont notées en rouge. Si la variable **sun** vaut **True**, nous afficherons les mots « fait beau » et sinon nous afficherons le mot « pleut ».

Nous générerons le fichier PDF depuis ce fichier en utilisant la commande **pdflatex**, disponible avec toute distribution LaTeX. Si vous n'avez pas de version de LaTeX d'installée sur votre système, vous pouvez par exemple installer le paquet **texlive** :

```
# aptitude install texlive
```

Attention : en l'état, ce fichier LaTeX ne compile pas (il faudrait supprimer les zones d'insertions).

3.3.3 La création du document

Nous allons rattacher le gabarit à l'objet **Contact** (nous utilisons le même gabarit quel que soit le contact) :

```
15: import sqlite3
16: import pystache
17: import os
18:
19:
20: class Contact:
21:
22:     base = None
23:     cursor = None
24:     template = None
25:
26:     def __init__(self, name, db_args):
...
39:         if Contact.template is None :
40:             try:
41:                 with open('doc.tex', 'r') as fic:
42:                     Contact.template = fic.read()
43:             except:
44:                 print('Accès impossible au gabarit')
45:                 exit(3)
46:
47:         self.forename = Contact.capitalizeName(name[0])
...
```

Cette façon de procéder nous permet de nous assurer que nous ne lirons le fichier **doc.tex** qu'une seule fois pour le stocker dans l'attribut statique **Contact.template**. La génération du fichier PDF se fera en utilisant cet attribut et en remplaçant les zones d'insertions par les bonnes valeurs :

```
131:     def generateDoc(self, sun=False):
132:         try:
133:             with open('doc_contact.tex', 'w') as fic:
134:                 fic.write(pystache.render(Contact.template,
135:                                 {'name': self.name,
136:                                  'forename': self.forename,
137:                                  'sun': sun}))
138:
139:             # Génération du PDF
140:             os.system('pdflatex doc_contact.tex 1>/dev/null')
141:         except:
142:             print('Erreur lors de la génération du PDF')
143:             exit(4)
```

Nous créons ici un fichier **doc_contact.tex** (ligne 133) dans lequel nous écrivons le contenu du gabarit (ligne 134) où les zones d'insertions sont remplacées par les valeurs des variables transmises dans le dictionnaire des lignes 135 à 137. La génération du PDF se fait à l'aide d'un appel système (ligne 140) et nous perdons donc l'aspect multiplateforme de notre application. Notez l'utilisation de **1>/dev/null** en fin de commande : ceci permet de désactiver les sorties-écran de pdflatex.

Il ne reste plus qu'à envoyer un mail à l'utilisateur.

3.4 L'envoi du mail

Pour pouvoir envoyer un mail, nous allons avoir besoin de quelques modules. Il faudra également indiquer qui envoie le mail et quel est le serveur :

```
015: import sqlite3
016: import pystache
017: import os
018: import smtplib
019: from email.mime.multipart import MIMEMultipart
020: from email.mime.base import MIMEBase
021: from email.mime.text import MIMEText
022: from email.utils import COMMASPACE, formatdate
023: from email import encoders
024:
025:
026: class Contact:
027:
028:     base = None
029:     cursor = None
030:     template = None
031:     send_from = 'moi@mon_adresse.com'
032:     server = 'localhost'
```

Dans le paramètre **server**, pour une utilisation locale avec une box quelconque vous devrez sans doute spécifier quelque chose du genre **smtp.mabox.com**.

Nous définissons ensuite une méthode **sendMail()** qui va permettre d'envoyer le mail et la pièce jointe :

```
153:     def sendMail(self):
154:         try:
155:             filename = 'doc_contact.pdf'
156:             msg = MIMEMultipart()
157:             msg['From'] = Contact.send_from
158:             msg['To'] = self.mail
159:             msg['Date'] = formatdate(localtime=True)
160:             msg['Subject'] = 'Un petit message'
161:
162:             msg.attach(MIMEText("""
163:                 Hey!\n
164:                 \n
165:                 Ce n'est pas du SPAM, tu peux lire le fichier
attaché!
166:             """))
167:
168:             part = MIMEBase('application', 'octet-stream')
169:             part.set_payload(open(filename,'rb').read() )
170:             encoders.encode_base64(part)
171:             part.add_header('Content-Disposition',
172:                             'attachment; filename="{}"'.format(
173:                                 os.path.basename(filename)))
174:              msg.attach(part)
175:
176:             smtp = smtplib.SMTP(Contact.server)
177:             smtp.sendmail(Contact.send_from, self.mail,
178:                           msg.as_string().encode('utf-8'))
179:             smtp.close()
180:         except:
181:             print('Erreur lors de l\'envoi du mail')
182:             exit(4)
```

Nous commençons par indiquer quel sera le fichier à joindre au mail (ligne 155) puis, en ligne 156, par définir notre message comme ayant le type **MIME** (*Multipurpose Internet Mail Extensions*). Les lignes 157 à 160 permettent de définir les traditionnels champs d'un mail (destinataire, expéditeur, date et sujet). Nous définissons ensuite le texte du mail (lignes 162 à 166), avant d'attacher le fichier à joindre (lignes 168 à 174). Enfin, nous envoyons le mail dans les lignes 176 à 179.

Pour que cette fonctionnalité soit activée, il faut spécifier dans le fichier **contacts.py** qu'après avoir généré le document PDF nous l'envoyons :

```
19: def cli_mode(args):
20:     person = Contact(args.name, (args.base, args.cursor))
21:     forename = person.forename
22:     name = person.name
23:     mail = person.mail
24:
25:     if mail == None
26:         print('Nouveau contact')
27:         mail = input('Veuillez saisir le mail: ')
28:         person.mail = mail
29:
30:     person.generateDoc(args.sun)
31:     person.sendMail()
32:
33:     print('Le mail a été envoyé à', forename, name)
```

Conclusion

Le cœur du programme est fonctionnel. Vous pouvez l'utiliser en ligne de commandes, mais vos utilisateurs ne seront pas encore satisfaits, il reste encore deux types d'interfaces à développer... ce que nous pourrons voir dans de prochains articles. ■

Références

[1] Définition du logiciel libre : https://www.gnu.org/philosophy/free-sw.fr.html

[2] Site du projet Mustache : http://mustache.github.io/

ASCIIDOC ET ASCIIDOCTOR POUR SOIGNER VOTRE DOCUMENTATION

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

La rédaction de documentation pour un projet est souvent reléguée au second plan, « on s'en occupera plus tard », voire carrément ignorée. Ce qui est très malheureux, surtout quand on est soi-même prompt à pester contre ce manque de documentation lorsque l'on récupère le projet d'un collègue ou que l'on arrive sur un nouveau poste. Plus qu'un manque de temps, c'est souvent les outils qui font défaut, et nous rebutent à nous lancer dans le processus de rédaction. Vous vous êtes d'ailleurs peut-être tourné vers Markdown, attiré par sa simplicité, mais limité, car trop simpliste.

Mots-clés : Asciidoc, Asciidoctor, Documentation, Editeur, HTML, Ebook.

Résumé

Dans cet article, nous allons découvrir Asciidoc et Asciidoctor : comment simplement convertir un fichier source au format Asciidoc en un document HTML et PDF.

Asciidoc [1] est un langage de balisage léger qui vous permet de rédiger vos documents, en les structurant à l'aide de caractères ASCII (*, _, =, ...), pour les générer en différents formats : pages web, e-books, etc.

Pourquoi Asciidoc et pas un éditeur WISYWIG comme **LibreOffice** ou **OpenOffice** ? Parce qu'avec Asciidoc vous n'avez pas besoin d'apprendre, ni de vous perdre dans les menus d'un éditeur lourd. Parce qu'avec Asciidoc vous travaillez au format texte, l'édition de documents Asciidoc s'intègre donc de manière transparente dans votre flux de travail à l'aide de vos outils courants, surtout si vous êtes habitué à travailler en mode console. Le format texte vous permet aussi de versionner votre travail dans un SCM et de collaborer facilement sur les documents.

Pourquoi Asciidoc et pas **Markdown** ? Asciidoc est moins limité que Markdown, tout en étant aussi simple et concis : on minimise la syntaxe à écrire, sans limiter la syntaxe que l'on peut écrire.

Pourquoi Asciidoc et pas DocBook ? Parce que la syntaxe d'Asciidoc permet de générer des documents aussi complexes qu'avec Docbook, le XML en moins de ce dernier : on garde la même sémantique, la verbosité en moins.

Comme ses alternatives, Asciidoc possède de nombreuses options, et en offre des supplémentaires :

- export vers HTML, DocBook, PDF, e-book, présentation, pages de manuel man ;
- système de thèmes ;
- création de syntaxe personnalisée ;
- inclusion de code, coloration syntaxique.

Qui utilise Asciidoc pour gérer sa documentation ? Asciidoc a été adopté par de larges projets pour leur documentation comme **Openshift**, **Elastic**, **Groovy** ou **Red Hat**. Git l'utilise pour gérer sa documentation officielle **Pro Git [2]** avec rendu au format HTML, PDF et e-book (vous trouverez les sources sur GitHub **[3]**). Oreilly l'utilise sur sa plateforme Atlas, mise à disposition pour ses auteurs **[4]**.

Et **Asciidoctor** dans tout ça ? À la base, Asciidoc, sorti début 2000, est à la fois la syntaxe et l'outil **`asciidoc`**, implémentation écrite en Python, qui traite les documents écrits en Asciidoc et génère le document final au format HTML ou Docbook (un autre outil du projet Asciidoc, **`a2x`**, permet la génération dans d'autres formats).

En 2013, Dan Allen de Red Hat débute le projet Asciidoctor **[5]**, et écrit l'outil **`asciidoctor`**, en Ruby, comme alternative au préprocesseur Python du projet natif Asciidoc. Implémentation moderne, elle est plus rapide que l'outil original, et offre plus d'extensions et fonctionnalités (émoticônes, balises de marquage supplémentaires, etc.). Elle est aussi polyglotte : l'API d'Asciidoctor est accessible depuis l'écosystème Java grâce à JRuby, et depuis JavaScript grâce à **Opal**, un transcompilateur Ruby vers JavaScript qui permet de générer une implémentation JavaScript presque complète de l'API à partir des sources Ruby. L'outil en ligne de commandes original **`asciidoc`** est toujours disponible est installable sur toutes distributions Linux, mais l'outil **`asciidoctor`** est devenu le standard de fait comme chaîne de compilation de documents au format Asciidoc.

Le projet Asciiidoctor s'est beaucoup développé, de nombreux sous-projets annexes ont vu le jour, sa communauté s'est fortement développée, et fait beaucoup la promotion de la syntaxe Asciidoc : c'est d'ailleurs grâce aux membres du projet Asciidoctor et leur action d'évangélisation que l'on peut profiter du support d'Asciidoc sur GitHub, permettant une stylisation du rendu au même titre que Markdown.

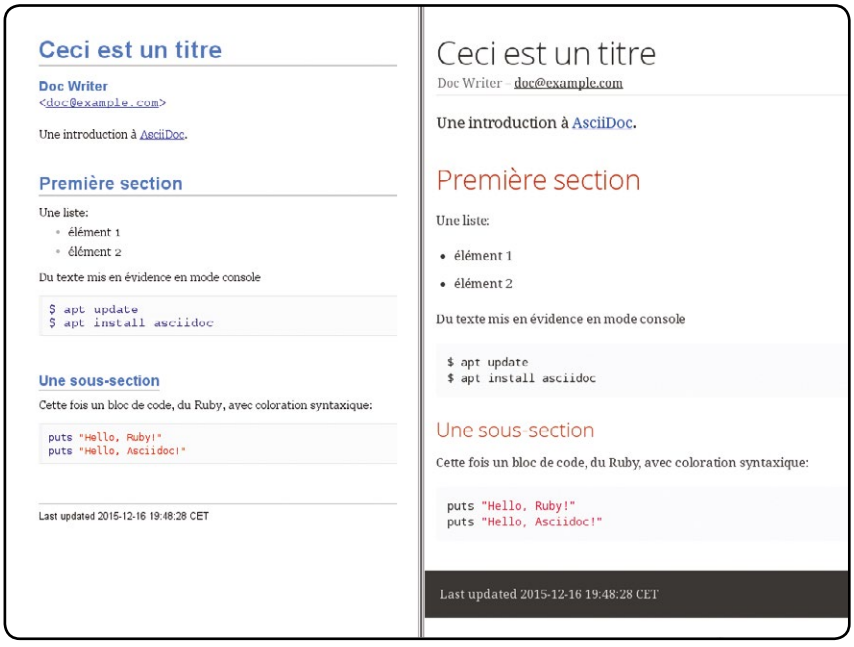


Fig. 1 : Rendu d'une page avec Asciidoc à gauche et Asciidoctor à droite.

Toute cette forte activité du projet Asciidoctor prête à confusion, entraîne une tendance à croire que les deux projets sont en compétition, voire qu'Asciidoc a disparu au profit de Asciidoctor. Pour conclure cette partie, et clarifier la situation : le projet Asciidoc existe toujours et définit la syntaxe, mais ses outils sont moins utilisés et je ne les utiliserai pas dans la suite de l'article ; Asciidoctor est le projet englobant différents outils qui sont désormais les plus couramment utilisés pour traiter le format Asciidoc, outils que je privilégierai dans la suite de l'article.

1 La syntaxe Asciidoc

Voici un exemple succinct de document au format Asciidoc :

```
= Ceci est un titre
Doc Writer <doc@example.com>

Une introduction à http://asciidoc.
org[AsciiDoc].

== Première section

Une liste:

* élément 1
* élément 2

Du texte mis en évidence en mode console

----
$ apt update
$ apt install asciidoc
----

=== Une sous-section

Cette fois un bloc de code, du Ruby, avec
coloration syntaxique:

[source,ruby]
----
puts "Hello, Ruby!"
puts "Hello, Asciidoc!"
----
```

À la lecture de ce fichier, on s'aperçoit de la concision de la syntaxe. Même si j'ai mis du texte explicatif en regard de certaines balises pour être explicite, cette syntaxe serait tout aussi simple

et évidente à comprendre et utiliser, même si on n'a pas lu sa spécification.

Dans la figure 1, page précédente, vous pouvez voir un exemple de rendu, à gauche avec l'outil original **asciidoc**, à droite avec l'outil **asciidoctor**. Dans les deux cas, l'export par défaut se fait au format HTML. On peut voir que le rendu par défaut d'**asciidoctor** est un peu plus sympathique. Dans la suite, nous allons voir comment obtenir et générer ce rendu de différentes manières via plusieurs outils du projet Asciidoctor.

2 Génération avec Asciidoctor

2.1 L'outil en ligne de commandes

L'outil par défaut à utiliser est l'outil en ligne de commandes d'Asciidoctor écrit en Ruby, **asciidoctor**. Pour l'installer, rien de plus simple, il est à coup sûr disponible sur votre distribution. Sous Debian, cela nous donne :

```
$ sudo apt install asciidoctor
```

Cette méthode est simple, mais présente l'inconvénient d'installer une version ancienne : sous Debian, la version 0.1.4-3. Pour installer une version plus récente, comme la dernière version stable 1.5.3, il faudrait directement installer la gemme Ruby (les versions 0.1.4 et 1.5, malgré leur numérotation, sont bien des versions consécutives, la différence de numérotation entre les deux étant due à une remise à plat des numérotations des différents projets internes de Asciidoctor) :

```
$ sudo gem install asciidoctor
```

Quelle que soit la méthode utilisée, vous aurez l'exécutable **asciidoctor** à disposition. Si vous avez enregistré le listing d'exemple Asciidoc précédent dans un fichier **exemple.adoc** (*adoc* est l'extension par convention d'un fichier au format Asciidoc), pour le générer au format HTML, on exécute la commande :

```
$ asciidoctor -a source-highlighter=coderay exemple.adoc
```

Le drapeau **-a** permet de préciser un ou plusieurs attributs, des paramètres qui gèrent le rendu de votre document : ici, j'ai indiqué d'utiliser **Coderay** pour la coloration syntaxique. Si vous ne voulez pas préciser d'attribut sur la ligne de commandes, vous pouvez les rajouter au début, en entête, de votre fichier Asciidoc en les rajoutant, un par ligne, selon la syntaxe suivante :

```
:source-highlighter: coderay
```

Note

Vous aurez peut-être à installer le paquet **coderay** (respectivement la gemme **coderay**, suivant si vous avez installé Asciidoctor en tant que gemme ou via le gestionnaire de paquets Debian).

Il existe de nombreux attributs : pour contrôler la numérotation des sections, des lignes des listings, la table des matières, etc.

Si vous êtes plutôt JavaScript, l'outil en ligne de commandes **asciidoctorjs**, installable via NPM, vous permet d'utiliser Asciidoctor via l'implémentation JavaScript du projet.

2.2 Extensions navigateurs

Vous n'avez envie d'installer ni une gemme Ruby, ni un paquet NPM? Pas de problème, via une extension navigateur, vous pouvez à la volée convertir du contenu Asciidoc en HTML directement depuis votre navigateur. De telles extensions sont disponibles pour **Chrome [6]** et **Firefox [7]**. Une fois installée et activée, tout fichier **.adoc** ouvert depuis votre navigateur, que ce soit un fichier local, ou une URL, sera automatiquement converti et rendu en HTML.

2.3 DocGist

Vous ne voulez vraiment rien installer sur votre ordinateur pour tester Asciidoc ? Il vous suffit d'utiliser **DocGist**, un service mis en place par Asciidoctor, qui se veut une alternative en ligne au rendu à la volée des extensions navigateurs. Alternative en ligne, tout d'abord, car c'est un service accessible à l'adresse http://gist.asciidoctor.org/, et aussi parce que le document source *adoc* à convertir doit être accessible via une URL. Différents services d'hébergement sont également supportés comme source en ligne de vos documents **.adoc** : dépôt Github, Gist Github, Google Doc, Etherpad, Dropbox, etc.

DocGist est un projet libre de Asciidoctor, vous pouvez récupérer ses sources **[8]** si vous souhaitez l'auto-héberger sur votre serveur.

2.4 Éditeurs

De nombreux éditeurs incluent un support avancé d'Asciidoc, et fournissent en particulier le *live preview* pour pouvoir afficher en temps réel le rendu de votre document Asciidoc tout en étant en train de le modifier : **IntelliJ**, **Eclipse**, **Atom**, **Bracket**, **Sublime**, etc.

3 Personnalisations

Nous avons vu comment utiliser le rendu par défaut HTML d'Asciidoctor. Mais l'utilisation de l'outil en ligne de commandes présente de nombreuses possibilités de thèmes et de *backends*.

3.1 Thèmes

Si le thème par défaut HTML ne vous satisfait pas, Asciidoctor propose un framework de création de thèmes, basé sur **Sass** et **Foundation**, et vous pourrez aussi utiliser des langages de *template* HTML comme **HAML** ou **Slim** pour accélérer vos développements. Une documentation complète, pas à pas, est disponible sur le site officiel **[9]**.

3.2 Backends

Jusqu'ici, nous n'avons vu que l'un des *backends* par défaut : le *backend* HTML. Il existe d'autres *backends* disponibles par défaut (Dockbook et pages de manuel) ainsi que des *backends* supplémentaires à installer en tant qu'extension : PDF, e-book, présentation (**Reveal.js**, **Deck.js**, etc.).

Les *backends* ne sont pas inclus par défaut lorsque vous installez l'outil, et sont à installer soit sous la forme de gemmes, soit sous la forme de dépôts source sur GitHub, voire les deux. Par exemple, pour utiliser le *backend* PDF, on doit installer la gemme :

```
$ sudo gem install --pre asciidoctor-pdf
```

Vous pouvez ensuite utiliser la même commande `asciidoctor` que précédemment, avec toujours notre fichier d'exemple Asciidoc comme source, mais cette fois-ci en précisant d'utiliser le *backend* PDF :

```
$ asciidoctor -a source-highlighter=coderay -r asciidoctor-pdf -b
pdf exemple.adoc
```

Dans la figure 2, vous pouvez voir la deuxième page du rendu PDF. Le titre principal du document n'est pas visible, car il a été automatiquement utilisé pour la page de garde du PDF.

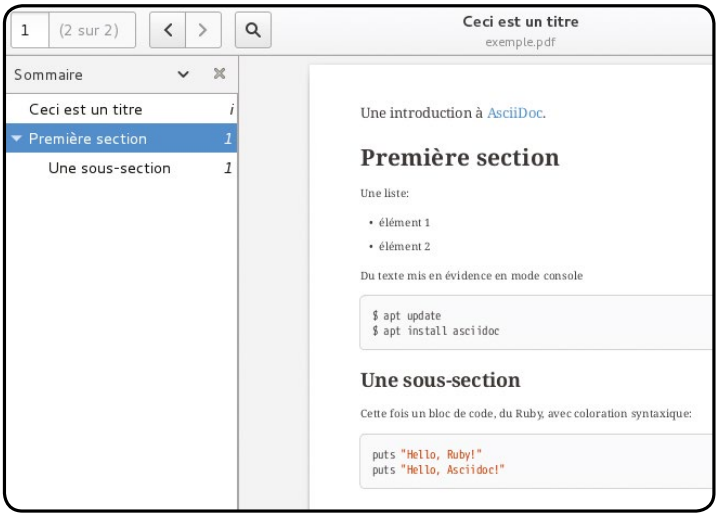


Fig. 2 : Rendu PDF de la page d'exemple.

Conclusion

J'espère que ce rapide tour d'Asciidoc et Asciidoctor vous a permis d'apprécier leurs qualités pour rédiger et générer vos documents. Dans un prochain article, nous verrons comment utiliser les *backends* et possibilités de templates d'Asciidoctor pour générer un document au format ODT. ■

Références

[1] Site officiel de Asciidoc : http://www.methods.co.nz/asciidoc/

[2] Livre Pro Git : https://git-scm.com/book/en/v2

[3] Source du livre Pro Git : https://github.com/progit/progit2

[4] Plateforme Atlas d'O'Reilly : https://atlas.oreilly.com/

[5] Site officiel d'Asciidoctor : http://asciidoctor.org/

[6] Extension Chrome : https://github.com/asciidoctor/asciidoctor-chrome-extension

[7] Extension Firefox : https://github.com/asciidoctor/asciidoctor-firefox-addon

[8] Dépôt sources du projet DocGist : https://github.com/asciidoctor/docgist

[9] Framework pour créer ses thèmes sous Asciidoctor : http://asciidoctor.org/docs/produce-custom-themes-using-asciidoctor-stylesheet-factory/

PHARO : GÉNÉRER DES DOCUMENTS PDF AVEC ARTEFACT

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]

Pharo dispose du framework Artefact pour la génération de documents PDF. Extensible sur la base de composants réutilisables, il propose un mécanisme de feuilles de style autorisant un contrôle aisé de l'apparence des documents.

Mots-clés : Pharo, PDF, Composant, Document, Unit, Roassal

Résumé

Artefact est un puissant *framework* permettant la génération de documents PDF avec Pharo. Dans cet article, vous allez découvrir ses principales fonctionnalités en produisant un document. Vous allez découvrir comment construire des composants réutilisables et gérer les styles de présentation.

Pour Artefact, l'intégralité des éléments composant un fichier PDF se résume à une arborescence d'objets indépendants les uns des autres. L'ordre dans lequel vous les positionnez dans le document n'influe pas sur leur apparence. Contrairement à de nombreux *frameworks* PDF qui exploitent une notion de flux dans la définition des styles, Artefact considère que chaque élément intègre sa propre feuille de style. Si un attribut n'est pas défini dans l'élément, Artefact exploite alors une feuille de style par défaut définie au niveau du document. Cette indépendance des éléments et cette gestion des styles permettent de produire rapidement un document et de le personnaliser pour une exploitation particulière.

Artefact peut générer des fichiers avec ou sans compression. Il permet la définition des métadonnées (titre, auteur, etc.) et gère les options d'affichage lors de l'ouverture du document dans un lecteur compatible avec cette fonctionnalité. Chaque page d'un document PDF peut se voir attribuer un format et une orientation particulière. Une page est composée de composants simples ou complexes (*pattern composite*) et n'obtient sa position définitive dans le document que lorsque celle-ci lui a été ajoutée. Les composants sont indépendants les uns des autres et se positionnent par rapport au coin supérieur gauche d'une page. Artefact offre une gestion des niveaux de gris, l'utilisation de couleurs (modèle RVB) et de la transparence. Les fontes de caractères sont celles définies dans la spécification PDF. Enfin, il est possible d'insérer des images JPEG ou PNG dans un document PDF.

1 Prise en main d'Artefact

Pour installer Pharo, rendez-vous sur le site du projet Pharo **[1]** et téléchargez la version 4.0 adaptée à votre ordinateur. Il vous faut ensuite ajouter le *framework* Artefact **[2]** qui est

disponible sur le dépôt Smalltalkhub.com. Il s'installe à l'aide du *Configuration Browser* de Pharo. Le *framework* **Unit**, destiné à la manipulation des différentes unités de mesure, sera installé automatiquement.

Une documentation électronique est accessible via le *Help Browser*. Le paquet **Artefact-Demos** contient de nombreux exemples d'utilisation. Ces petites démonstrations seront stockées dans un répertoire nommé **pdf** placé dans l'arborescence de Pharo. Pour générer ces exemples, vous devez exécuter dans le *Playground* le code suivant :

```
PDFDemos runAllDemos
```

Dans **Artefact-Elements-Basic** et **Artefact-Elements-Composites**, vous trouvez les différents composants (texte, formes géométriques, images, etc.) proposés de base par le *framework*. Les fontes sont définies dans le paquet **Artefact-Fonts** et les formats de documents (A4, A3, ebook, etc.) dans le paquet **Artefact-Formats**.

2 "Hello World!"

Pour créer votre premier document PDF, vous devez définir une instance d'un document PDF.

```
PDFDocument new exportTo: 'helloworld.pdf' asFileReference
writeStream
```

L'instance de **PDFDocument** est exportée à l'aide d'un flux vers un fichier nommé **helloworld.pdf**. Par défaut, le document PDF est placé dans le même répertoire que Pharo. Pour le moment, le fichier PDF est vide. Enrichissons maintenant le code précédent en ajoutant une page au document.

```
PDFDocument new add: PDFPage new; exportTo: 'helloworld.pdf'
asFileReference writeStream
```

Votre document contient maintenant une page vierge. Artefact suit la philosophie « *Convention over configuration* ».

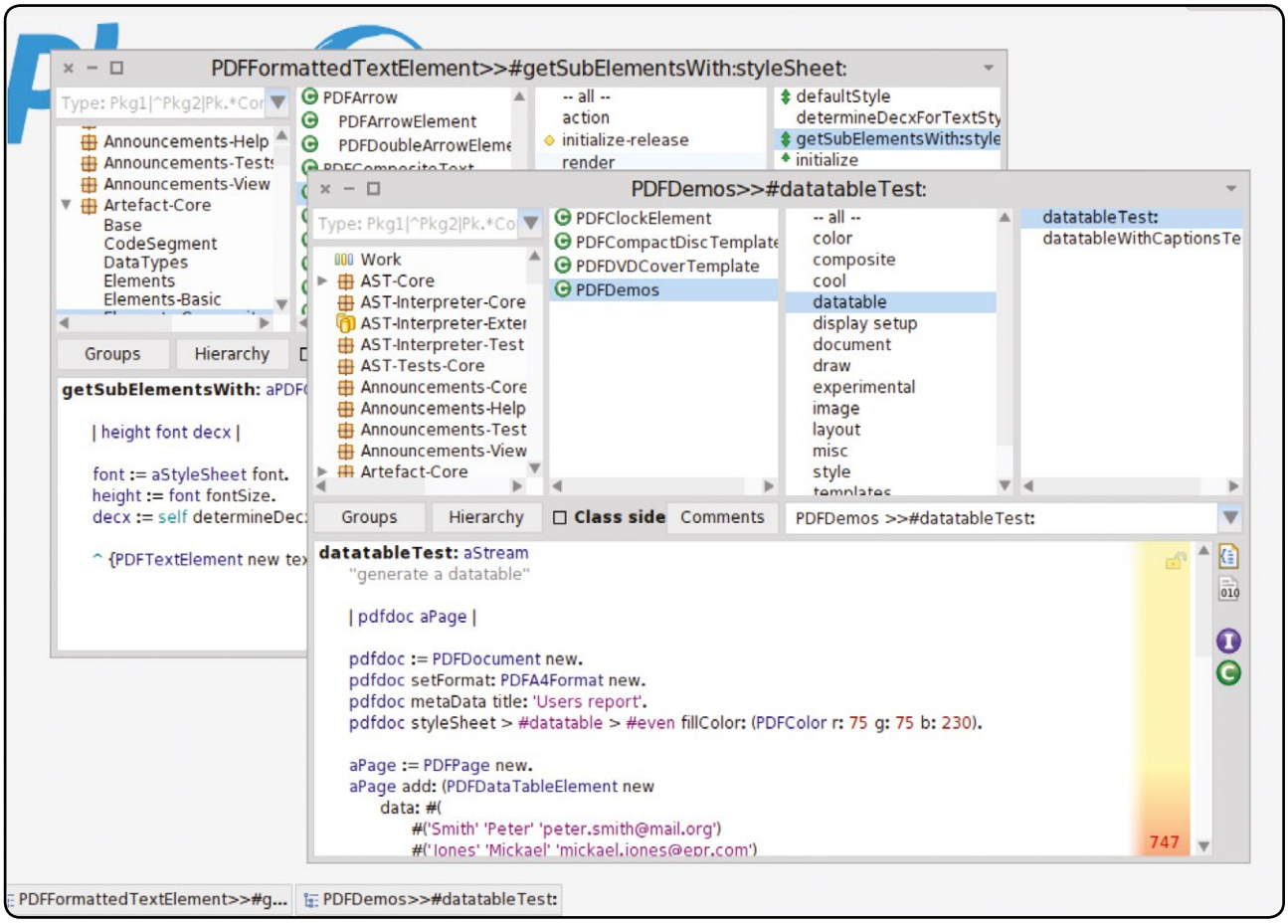


Fig. 1 : Faire du PDF avec Pharo.

Seuls les paramètres spécifiques doivent être définis. Artefact a donc attribué un format A4 à votre document, fixé l'orientation en portrait et initialisé les métadonnées avec des valeurs par défaut. Nous terminons notre premier document en ajoutant un composant texte sur la page.

```
PDFDocument new add:
  (PDFPage new add:(PDFTextElement new text: 'Hello World!';
from: 10mm @ 10mm));
  exportTo: 'helloworld.pdf' asFileReference writeStream
```

L'utilisation du *framework* Unit autorise l'utilisation d'unités métriques. Ces coordonnées sont définies à partir de l'angle supérieur gauche de la page (Figure 1, page précédente).

3 Travaux pratiques

L'exemple sera composé d'une page au format A4 en mode paysage. Dans la partie supérieure, vous allez positionner une image. Ensuite, vous allez disposer un paragraphe de texte, un tableau de données, un bas de page et un histogramme. Ces deux derniers éléments seront l'occasion pour vous de créer des composants Artefact réutilisables.

3.1 Conception du document et de la page

Dans Pharo, ajoutez un paquet nommé **LinuxMagArtefact** qui contiendra les classes du projet. Deux approches sont possibles pour la définition du document et des pages. La première consiste à utiliser des variables d'instance pour manipuler les instances du document et des pages. La seconde consiste à ajouter de nouvelles classes à Pharo en se basant bien évidemment sur l'héritage. C'est cette solution que vous allez utiliser. Vous devez donc définir la classe **LMPDFDocument** qui hérite de **PDFDocument** :

```
PDFDocument subclass: #LMPDFDocument
  instanceVariableNames: ''
  classVariableNames: ''
  category: 'LinuxMagArtefact'
```

Dans cette classe, ajoutez un protocole **initialize-release** et définissez la méthode **initialize** afin d'initialiser les attributs du document :

```
LPDFDocument >> initialize
  super initialize.
  self metaData title: 'Artefact'; subject: 'Exemple pour
LinuxMagazine';
  author: 'Olivier Auverlot'; keywords: 'pdf pharo'; creator:
'Artefact - Pharo'.
```

Ces métadonnées sont importantes, car elles permettent de documenter votre document et sont utilisées par les outils d'indexation de votre ordinateur ou des solutions de GED (Gestion Électronique de Documents).

La seconde étape consiste à créer la page de votre document. Pour cela, définissez la classe **LMPDFPage** qui hérite de **PDFPage** :

```
PDFPage subclass: #LMPDFPage
  instanceVariableNames: ''
  classVariableNames: ''
  category: 'LinuxMagArtefact'
```

Ajoutez un protocole **initialize-release** contenant une méthode **initialize**. Celle-ci sélectionne le format de la page via la variable d'instance **format**. Dans Artefact, chaque format de document est défini par une classe qui hérite de **PDFFormat**.

```
LMPDFPage >> initialize
  super initialize.
  self format: PDFA4Format new setLandscape.
```

Le code destiné à placer les composants Artefact sur la page est placé dans une méthode nommée **renderContent**, classée dans le protocole **render**. Pour l'instant, créez simplement ce protocole et cette méthode, mais laissez-la vide de tout contenu. Pour générer le document, une solution simple est de construire le fichier PDF à chaque instanciation de **LMPDFDocument**. Il vous faut rédiger une méthode de classe **render** qui instancie **LMPDFPage** et appelle la méthode d'instance **renderContent**.

```
LMPDFPage class >> render
  ^self new renderContent
```

Ajoutez ensuite le code suivant dans la méthode **initialize** de la classe **LMPDFDocument** :

```
self add: LMPDFPage renderContent; exportTo:'/tmp/exemple.pdf'
asFileReference writeStream
```

Ainsi, à chaque fois qu'une instance de **LMPDFDocument** est produite, le rendu de la page est déclenché. La page est ajoutée au document et exportée dans un fichier via un flux. Si vous ouvrez le fichier généré à l'aide d'un lecteur PDF,

vous obtenez un document contenant une unique page vide. Seuls les attributs du document sont renseignés. Bon, c'est pas mal, mais maintenant, il va falloir remplir.

3.2 Ajouter une image

L'image que vous allez placer dans la partie supérieure de la page provient du blog du magazine *GNU/Linux Magazine*. Pour stocker l'image, vous devez définir une variable d'instance **logoLinuxMag** dans la classe **LMPDFPage**. Dans la méthode **initialize** de cette classe, ajoutez le code suivant pour la télécharger et l'affecter à la variable d'instance :

```
logoLinuxMag := ZnClient new
 url: 'http://www.gnulinuxmag.com/wp-content/uploads/2014/10/
blog_glmf_banniere.png';get.
```

La méthode **renderContent** instancie un composant **PDFPngElement** qui contient l'image PNG et fixe sa position et sa taille. L'image est redimensionnée proportionnellement avant d'être ajoutée à la page :

```
self add: ((PDFPngElement fromStream: (logoLinuxMag readStream))
 from: 10mm@10mm; dimension: 277mm@56mm).
```

3.3 Ajouter un paragraphe de texte

Artefact dispose du composant **PDFParagraphElement**. Le texte est inséré dans un cadre ayant une position de départ, une largeur et une hauteur. Le composant gère la césure du texte en fonction de ses dimensions ainsi que de l'alignement. Le document produit étant totalement fictif, vous allez générer une chaîne de caractères en guise de texte. Vous avez besoin de deux méthodes qui sont regroupées dans un protocole **private** puisqu'elles ne sont pas susceptibles d'être appelées en dehors de **LMPDFPage**.

```
LMPDFPage >> randomString: anInteger
 ^(String new: anInteger) collect: [ :x |
'abcdefghijklmnopqrstuvwxyz' atRandom ]

LMPDFPage >> randomText: anInteger
 | text |
 ^String streamContents: [ :text | anInteger timesRepeat: [
   text nextPutAll: (self randomString: (8 atRandom)); nextPut:
Character space ] ]
```

Vous pouvez maintenant modifier la méthode **renderContent** en ajoutant le code nécessaire pour construire le paragraphe contenant deux cents chaînes de caractères.

```
self add: (
 PDFParagraphElement new from: 10mm@70mm; dimension: 60mm@120mm;
text: (self randomText: 200)).
```

3.4 Utiliser un tableau de données

Artefact dispose d'un composant tableau de données qui autorise la production rapide de rapports. Dans le paquet **Elements-Composite**, se trouvent la classe **PDFDataTableElement** et ses classes héritées.

Pour cet exemple, vous allez générer dynamiquement un tableau de deux colonnes composé d'un nom aléatoire et d'une valeur numérique également aléatoire. Ajoutez une méthode **randomDataGrid:** dans le protocole **private** de la classe **LMPDFPage**. L'argument de la méthode est le nombre de lignes du tableau.

```
LMPDFPage >> randomDataGrid: anInteger
 | datagrid |
 datagrid := OrderedCollection new.
 anInteger timesRepeat: [
  datagrid add: (OrderedCollection with: (self randomString:10)
with: (255 atRandom)) ].
 ^datagrid
```

Vous pouvez maintenant ajouter le code suivant à la méthode **renderContent** afin d'insérer un tableau comportant quinze lignes :

```
self add: (PDFDataTableElement new data: (self randomDataGrid:
15); from: 80 mm @ 70 mm; dimension: 60 mm @ 120 mm).
```

Améliorez un peu le tableau en ajoutant des noms de colonne pour le rendre plus expressif. Artefact le fait très simplement grâce à la sous-classe **PDFDataTableWithColumnsCaptionElement**.

```
self add: (PDFDataTableWithColumnsCaptionElement new data: (self
randomDataGrid: 15);
  from: 80 mm @ 70 mm; dimension: 60 mm @ 120 mm; captions:
#('Nom' 'Valeur')).
```

La tableau de données autorise de nombreuses personnalisations. Vous pouvez modifier les fontes de caractères, les couleurs, l'épaisseur des lignes, etc. Les feuilles de style que vous découvrirez un peu plus loin dans cet article, sont l'outil à privilégier pour définir l'apparence de composants complexes tels que ceux hérités de **PDFDataTableElement**.

3.5 Créer un composant pour le bas de page

Votre document PDF doit avoir un bas de page constitué d'un texte et d'une ligne séparant ce texte du corps du document. Ce besoin simple est l'occasion de créer votre premier composant. Celui-ci étant le résultat de l'assemblage de différents éléments, vous devez le faire hériter de la classe **PDFComposite**.

```
PDFComposite subclass: #LMPDFFooterElement
  instanceVariableNames: 'text'
  classVariableNames: ''
  category: 'LinuxMagArtefact'
```

Votre composant **LMPDFFooterElement** dispose d'une variable d'instance **text** destinée à contenir le texte et accessible via des accesseurs.

```
LMPDFFooterElement >> text
  ^ text
LMPDFFooterElement >> text: aString
  text := aString
```

Les composants ont l'obligation de déclarer leur style par défaut. C'est ce style qui permettra par la suite de définir intelligemment l'apparence du composant. Pour le fixer, créez un protocole **style** et ajoutez une méthode nommée **defaultStyle** :

```
LMPDFFooterElement >> defaultStyle
  ^ #lmFooter
```

Pour spécifier le rendu de votre composant, ajoutez un protocole **render** et ajoutez-lui une méthode **getSubElementsWith:styleSheet:**. Dans un premier temps, le composant dessine une simple ligne.

```
LMPDFFooterElement >> getSubElementsWith: aPDFGenerator
styleSheet: aStyleSheet
  ^{ PDFLineElement from: (self from) to: (self dimension x @
self from y) }
```

Le paramètre **aPDFGenerator** permet de fournir au composant le contexte dans lequel celui-ci est rendu. La méthode retourne un tableau d'instances de composants. La ligne est dessinée avec le composant **PDFLineElement** et les dimensions s'adaptent à la taille du composant qui la contient.

Vous pouvez maintenant ajouter un texte sous la ligne. Celui-ci devant être centré sur la page, il est placé dans un **PDFFormattedTextElement** qui a les dimensions de son conteneur.

```
LMPDFFooterElement >> getSubElementsWith: aPDFGenerator styleSheet: aStyleSheet
  ^{ PDFLineElement from: (self from) to: (self dimension x @ self from y).
    PDFFormattedTextElement new from: (self from); dimension: (self dimension x @ 5);
    text: self text; alignment: (PDFAlignment center). }
```

Il ne vous reste plus qu'à utiliser ce nouveau composant dans votre document PDF. Pour cela, ajoutez une nouvelle instance de **LMPDFFooterElement** dans la méthode **renderContent** de la classe **LMPDFPage** :

```
self add: (LMPDFFooterElement new text: 'Une démonstration d''Artefact'; from: 10 mm @
195 mm;
 dimension: 277 mm @ 10 mm; font: (PDFHelveticaFont new fontSize: 16 pt)).
```

Bien évidemment, il est possible de concevoir des composants bien plus évolués tels qu'un histogramme.

3.6 Dessiner un histogramme à l'aide de Roassal

Un besoin courant est de générer des graphiques adaptés à la représentation de données. Pour cela, Pharo dispose du *framework* **Roassal [3]**. Pour produire un histogramme composé de barres verticales dans votre document PDF, l'approche idéale avec Artefact consiste donc à définir un nouveau composant qui aura la responsabilité de demander à Roassal de construire l'histogramme avant de l'insérer dans

Fig. 2 : Les différents composants ont été placés sur la page.

le document PDF sous la forme d'une image. Votre nouveau composant **VerticalBarDiagramElement** sera ainsi réutilisable.

Pour installer Roassal dans votre image Pharo, ouvrez un *Playground* et exécutez le code suivant :

```
Gofer it
  smalltalkhubUser: 'objectProfile' project: 'Roassal2';
  configurationOf: 'Roassal2';
  loadStable
```

Dans votre *package* **LinuxMagArtefact**, créez maintenant la classe **VerticalBarDiagramElement**. Elle hérite de **PDFComposite** et sa variable d'instance **data** contient les données à représenter.

```
PDFComposite subclass: #VerticalBarDiagramElement
  instanceVariableNames: 'data'
  classVariableNames: ''
  category: 'LinuxMagArtefact'
```

Pour être accessible à l'extérieur du composant, la variable d'instance **data** doit disposer d'accesseurs.

```
data
  ^data

data: aCollectionOfValues
  data := aCollectionOfValues
```

Ajoutez maintenant une méthode **defaultStyle** permettant à Artefact d'appliquer son mécanisme de feuille de style sur le contenu de ce composant. C'est une bonne pratique de le faire systématiquement.

```
defaultStyle
  ^ #verticalBarDiagram
```

La méthode **getSubElementsWith:styleSheet:** est responsable de la génération du graphique. La première étape consiste à calculer un ratio entre la largeur et la hauteur du graphique sur la page afin de respecter les proportions et éviter ainsi que le graphique soit déformé. La méthode génère une instance de la classe **GET2DiagramBuilder** fournie par Roassal. Elle lui transmet les données à représenter ainsi que les dimensions de la surface sur laquelle le graphique doit être dessiné.

```
getSubElementsWith: aPDFGenerator styleSheet: aStyleSheet
 | builder dataSet ratio view chartmorph |
 ratio := self dimension y / self dimension x.
 builder := RTGrapherBuilder new.
 builder extent: 300 @ (300 * ratio).
 dataSet := RTStackedDataSet new.
 dataSet points: self data.
 dataSet barShape color: Color gray; width: 20.
 builder add: dataSet.
 builder axisX.
 builder axisYWithNumberOfTicks: 4.
 view := builder build.
 builder view setUpCanvas.
```

Roassal dessine l'histogramme sur une instance de la classe **TRCanvas**, convertie ensuite en image PNG avant son insertion dans le document PDF. Vous devez donc produire une instance de **TRMorph** à partir du contenu du **TRCanvas**. La surface obtenue est plus grande que celle du graphique afin d'obtenir des marges autour du dessin.

```
chartmorph := view canvas buildMorph width: 400; height: (400 * ratio).
```

La dernière étape consiste à positionner l'image sur la page et à l'entourer par un simple rectangle. La méthode **getSubElementsWith:styleSheet:** doit retourner une collection contenant une instance de **PDFRectElement** ainsi qu'une instance de **PDFPngElement**.

```
^{ (PDFRectElement from: (self from) dimension: (self dimension)).
  ((PDFPngElement fromMorph: chartmorph)from: self from;
dimension: self dimension ) }
```

Votre composant **VerticalBarDiagramElement** est prêt. Pour l'utiliser, ajoutez le code suivant dans la méthode **renderContent** de la classe **LMPDFPage** :

```
self add: (VerticalBarDiagramElement new from: 150 mm @ 70 mm;
  dimension: 130 mm @ 120 mm; data: ((1 to: 10) collect: [ :v |
10 atRandom ])).
```

Un magnifique histogramme représentant dix valeurs aléatoires est ajouté à votre page. Chaque composant étant autonome, vous pouvez le réutiliser et placer plusieurs histogrammes dans votre document.

4 Modifier l'esthétique d'un document avec les feuilles de styles

Les feuilles de styles constituent un mécanisme incontournable en contrôlant l'apparence d'un document PDF.

La classe **PDFStyleSheet** est un dictionnaire qui regroupe les propriétés de rendu. Une feuille de styles par défaut est attachée à chaque instance de la classe **PDFDocument** et fournie des styles par défaut aux composants. Le modèle est hiérarchique : une feuille de styles définie au niveau le plus bas surchargera les propriétés héritées.

Dans notre document, l'ensemble des textes est écrit en noir. En modifiant sa feuille de style, vous allez spécifier une couleur différente, utiliser une fonte différente et des lettres plus épaisses. La définition de la feuille de style du document est placée dans la méthode **defineDocumentStyleSheet** de la classe **LMPDFDocument**. Ajoutez l'appel de cette méthode dans la méthode **initialize** de la classe **LMPDFDocument**.

Les composants d'Artefact disposent d'un identifiant défini par la méthode **defaultStyle**. Par défaut, les composants manipulant du texte ont l'identifiant **#text**. Le code suivant fixe la fonte et la couleur du texte pour chacun des composants utilisant ce style.

```
defineDocumentStyleSheet
  self styleSheet > #text at: #font put: ((PDFTimesFont size: 10 pt) bold);
    at: #textColor put: (PDFColor r: 128 g: 128 b: 128).
```

Fig. 3 : L'apparence des composants est gérée par une feuille de styles.

Pour modifier la couleur et la fonte de l'instance du composant **LMPDFFooterElement**, vous devez ajouter à la méthode le code suivant :

```
self styleSheet > #lmFooter
  font: ((PDFTimesFont size: 10 pt) bold); at: #textColor put:
(PDFColor r: 128 g: 128 b: 128).
```

Malheureusement, cela ne fonctionne pas correctement, car la fonte est restée la même. Ceci est dû au fait que vous avez préalablement forcé l'usage d'une fonte dans votre code. Cette valeur est la dernière prise en compte par Artefact lors de la résolution des styles. Vous ne devez donc jamais préciser l'apparence d'un composant sans passer par sa définition via une feuille de styles. Pour corriger votre erreur, il vous suffit de modifier la méthode **renderContent** de la classe **LMPDFPage**, à l'endroit où vous insérez une instance de **LMPDFFooterElement** :

```
self add: (LMPDFFooterElement new text: 'Une démonstration
d''Artefact';
  from: 10 mm @ 195 mm; dimension: 277 mm @ 10 mm).
```

Les styles permettent de modifier l'ensemble des attributs de présentation dans Artefact tels que les fontes, les couleurs, l'épaisseur des lignes, les pointillés, la transparence, etc. Profitez-en pour améliorer l'apparence du tableau de données !

L'identifiant de cette famille de composant est **#datatable**. Commencez par modifier la fonte et la couleur afin de rendre l'apparence du tableau cohérente avec le reste du document :

```
self styleSheet > #datatable
  font: ((PDFTimesFont size: 10 pt) bold); textColor: (PDFColor
r: 128 g: 128 b: 128).
```

Modifiez maintenant la couleur de tracé des contours du tableau et ajoutez une couleur de fond :

```
self styleSheet > #datatable
  font: ((PDFTimesFont size: 10 pt) bold); textColor: (PDFColor
r: 128 g: 128 b: 128);
  drawColor: (PDFColor r: 128 g: 128 b: 128); fillColor:
(PDFColor r: 238 g: 234 b: 236);
```

À l'aide des attributs **thickness** et **dotted**, vous modifiez l'épaisseur des lignes et utiliser des pointillés :

```
self styleSheet > #datatable
  font: ((PDFTimesFont size: 10 pt) bold); textColor: (PDFColor
r: 128 g: 128 b: 128);
  drawColor: (PDFColor r: 128 g: 128 b: 128); fillColor:
(PDFColor r: 238 g: 234 b: 236);
  thickness:  0.2 mm; dotted: (PDFDotted new length: 0.4mm;
space: 0.4mm).
```

Avec la classe **PDFDataTableWithColumnsCaptionElement**, vous avez la possibilité de modifier l'apparence des titres de colonnes. Pour cela, vous devez modifier le style **#caption** :

```
self styleSheet > #datatable > #caption
  font: ((PDFTimesFont size: 10 pt) bold);
  textColor: (PDFColor r: 238 g: 234 b: 236); fillColor:
(PDFColor r: 128 g: 128 b: 128).
```

L'apparence de votre tableau de données a radicalement changé, et cela sans toucher à l'instance du composant placée sur la page.

Conclusion

Vous êtes maintenant capable de produire des documents PDF avec Artefact. C'est un *framework* simple d'utilisation, mais extrêmement riche en termes de fonctionnalités. Il suit la logique de Pharo en vous permettant de réaliser des opérations complexes simplement, efficacement et en permettant de manière intuitive la réutilisation de votre travail. N'hésitez pas à télécharger et à modifier cet exemple qui est disponible sur le site de dépôt Smalltalkhub **[4]**. ■

Références

[1] Site officiel du projet Pharo : http://www.pharo.org

[2] Site officiel du projet Artefact : https://sites.google.com/site/artefactpdf

[3] Site officiel du projet Roassal : http://agilevisualization.com

[4] Télécharger l'exemple : http://smalltalkhub.com/#!/~olivierauverlot/LinuxMagPDF

ACTUELLEMENT DISPONIBLE

MISC N°83 !

IPv6 : 10 ANS APRÈS !

NE LE MANQUEZ PAS

CHEZ VOTRE MARCHAND DE JOURNAUX ET SUR :

www.ed-diamond.com

CRÉATION D'UN MENU EN 5 MINUTES AVEC POLYMER

Tristan Colombo

Polymer permet de créer et d'utiliser des Web components pour un développement plus rapide. Nous allons voir que grâce à ce framework le développement d'un menu pour une application Web ne prend pas plus de cinq minutes.

Mots-clés : Polymer, Javascript, Web components, Ergonomie, Menu, HTML

Résumé

Nous avons toujours besoin d'une barre de menu dans une application web... mais il ne s'agit « que » d'un élément ergonomique, ce n'est pas lui qui va porter l' « intelligence » de l'application et on a donc souvent tendance à le négliger, à le coder rapidement. Pourtant, pour l'utilisateur, il s'agit d'un élément essentiel lui permettant de dialoguer avec l'application. Polymer permet de développer un menu ergonomique en seulement quelques minutes et c'est ce que nous réalisons dans cet article.

Après vous avoir présenté Polymer le mois dernier **[1]**, je vous propose de développer un *Web component* utile dans la majeure partie des développements d'applications web : le menu. Bien entendu, en cinq minutes nous n'allons pas créer une application complète, mais simplement une page assez classique comportant uniquement un menu permettant de naviguer sur plusieurs pages et un bouton de connexion. Malgré un titre aguicheur, je tiens à préciser que vous pourrez réaliser votre menu en cinq minutes certes, mais cela grâce au présent article qui a nécessité plus de cinq minutes de rédaction : la documentation de Polymer est très bien faite, mais malheureusement incomplète et de nombreux exemples sont disponibles uniquement en version 0.5 (qui n'est plus compatible avec la version 1.x). L'objectif de ce mini projet est donc de vous montrer la souplesse d'utilisation de ce *framework* en l'appliquant à un cas pratique.

1 Le démarrage du projet

Je vais supposer que vous avez déjà installé **Bower**. Pour initialiser notre projet, il suffit donc de créer un répertoire et d'y intégrer Polymer :

```
$ mkdir glmf
$ cd glmf
$ bower init
$ bower install --save Polymer/polymer
```

Pour notre projet, nous allons utiliser l'élément **`paper-toolbar`** qui crée une barre d'outils pouvant servir de menu. Dans cette barre, nous allons pouvoir placer des icônes grâce à **`iron-icons`** (ensemble d'icônes) et **`paper-icon-button`** (le nom parle de lui-même) :

```
$ bower install --save polymerelements/paper-toolbar
$ bower install --save polymerelements/iron-icons
$ bower install --save polymerelements/paper-icon-button
```

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-
toolbar/paper-toolbar.html" />
11:         <link rel="import" href="bower_components/iron-icons/
iron-icons.html" />
12:         <link rel="import" href="bower_components/paper-icon-
button/paper-icon-button.html" />
13:     </head>
14:
15:     <body unresolved>
16:         <paper-toolbar>
17:             <paper-icon-button icon="menu" on-
tap="menuAction"></paper-icon-button>
18:             <span class="title">GLMF Polymer</span>
19:         </paper-toolbar>
20:     </body>
21: </html>
```

On retrouve le squelette d'un document Polymer classique avec le chargement de la bibliothèque **webcomponentsjs** (ligne 8) et des éléments Polymer que l'on va utiliser (lignes 10 à 12).

Pour rappel, l'attribut **unresolved** associé au tag **<body>** permet de prévenir les FOUC (*Flash Of Unstyled Content*).

Dans les lignes 16 à 19, nous créons la barre d'outils par **<paper-toolbar>** et nous ajoutons une entrée sous forme d'icône à l'aide de **<paper-icon-menu>**. Pour l'instant, l'action **menuAction** n'est pas encore définie (rappelez-vous que **on-tap** est un événement Polymer et que nous pourrons le définir en créant le *Web component* qui sera associé à notre menu).

Pour voir le résultat de notre page, il faut lancer un serveur (**python3 -m http.server**), puis vous rendre à l'adresse localhost:8000 pour voir la barre d'outils telle que présentée en figure 1.

Fig. 1 : Barre d'outils paper-toolbar contenant une icône de menu et un titre.

Si nous voulons faire fonctionner notre bouton de menu, il faut compléter notre page.

2 Ajout des entrées du menu

Maintenant que nous avons créé notre « brouillon » de menu, nous allons le modifier pour rester conforme à la structure de Polymer. Pour cela, nous allons créer de nouveaux *Web components* dans des fichiers distincts et nous les utiliserons ensuite dans la page **index.html**.

Pour pouvoir réellement afficher un menu (et non simplement son icône), il va nous falloir de nouveaux éléments :

```
$ bower install --save polymerelements/paper-menu-button
$ bower install --save polymerelements/paper-menu
$ bower install --save polymerelements/paper-item
$ bower install --save polymerelements/iron-pages
$ bower install --save polymerelements/paper-material
$ bower install --save polymerelements/paper-header-panel
```

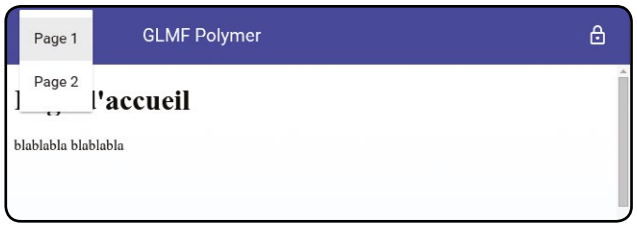


Fig. 2 : Aperçu du projet fini.

Je vais partir du projet fini (voir figure 2) et vous expliquer les différents éléments utilisés. Commençons par le 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="bower_components/paper-
toolbar/paper-toolbar.html" />
11:         <link rel="import" href="bower_components/paper-
header-panel/paper-header-panel.html" />
12:
```

```
13:         <link rel="import" href="glmf_components/glmf-menu/
glmf-menu.html" />
14:         <link rel="import" href="glmf_components/glmf-login-
button/glmf-login-button.html" />
15:         <link rel="import" href="glmf_components/glmf-pages/
glmf-pages.html" />
```

Dans les lignes 10 et 11, vous pouvez voir les imports des éléments Polymer **paper-toolbar** (la barre d'outils) et **paper-header-panel** (page avec en-tête qui va permettre de conserver le menu en haut de page). Dans les lignes 13 à 15, nous importons des *Web components* que nous avons créés et dont nous verrons le code par la suite.

```
17:         <style is="custom-style">
18:             .content {
19:                 height: 2000px;
20:                 padding: 0 10px;
21:             }
22:
23:             .blue-gradient {
24:                 background: linear-gradient(white, #b3e5fc);
25:             }
26:         </style>
27:     </head>
```

Le bloc de style des lignes 17 à 26 permet de modifier le style par défaut. Ici la classe **.content** définit la taille d'une page et la classe **.blue-gradient** sera utilisée pour son fond : en faisant défiler la page, celle-ci deviendra progressivement bleue.

```
29:     <body unresolved class="fullbleed layout vertical">
30:         <paper-header-panel mode="waterfall" class="flex">
31:             <paper-toolbar>
32:                 <glmf-menu display="page"></glmf-menu>
33:
34:                 <span class="title">GLMF Polymer</span>
35:
36:                 <glmf-login-button display="page">
</glmf-login-button>
37:             </paper-toolbar>
38:
39:             <div class="content fit blue-gradient">
40:                 <glmf-pages display="page" selected="home">
</glmf-pages>
41:             </div>
42:         </paper-header-panel>
43:     </body>
44: </html>
```

En ligne 29, nous retrouvons le tag **<body>** avec l'attribut **unresolved** pour éviter les FOUC. Les classes employées sont prédéfinies dans Polymer (ici pour un affichage plein écran). En ligne 30, nous définissons la page à l'aide d'un **paper-header-panel** en mode **waterfall** : l'en-tête, qui est ici la barre d'outils des lignes 31 à 37, restera fixe. À propos de barre d'outils, celle-ci est définie en employant deux *Web components* que nous avons définis : **glmf-menu** pour le menu déroulant et **glmf-login-button** pour le bouton de connexion. Le contenu des pages sera géré par **glmf-pages** (ligne 40).

Pour comprendre le fonctionnement de la page, il nous faut maintenant nous pencher sur les éléments que nous avons créés. Commençons par le menu se trouvant dans le fichier **glmf_components/glmf-menu/glmf-menu.html** (je pense qu'il est toujours préférable d'adopter la même nomenclature pour les fichiers du projet que celle utilisée dans Polymer) :

```
01: <link rel="import" href="../../bower_components/polymer/
polymer.html">
02: <link rel="import" href="../../bower_components/paper-menu-
button/paper-menu-button.html" />
03: <link rel="import" href="../../bower_components/paper-menu/
paper-menu.html" />
04: <link rel="import" href="../../bower_components/paper-item/
paper-item.html" />
05: <link rel="import" href="../../bower_components/iron-icons/
iron-icons.html" />
06: <link rel="import" href="../../bower_components/paper-icon-
button/paper-icon-button.html" />
07:
08: <dom-module id="glmf-menu">
09:
10:     <template>
11:         <paper-menu-button>
12:             <paper-icon-button icon="menu" class="dropdown-
trigger"></paper-icon-button>
13:             <paper-menu class="dropdown-content">
14:                 <paper-item on-tap="menuSite">Page 1
</paper-item>
15:                 <paper-item on-tap="menuConnect">Page 2
</paper-item>
16:             </paper-menu>
17:         </paper-menu-button>
18:     </template>
19:
20:     <script>
21:         Polymer({
22:             is: 'glmf-menu',
23:             properties: {
24:                 display: String
25:             },
26:             menuSite: function () {
27:                 document.getElementById(this.display).
select('page1');
```

```
28:             },
29:             menuConnect: function () {
30:                 document.getElementById(this.display).
select('page2');
31:             },
32:         })();
33:     </script>
34:
35: </dom-module>
```

Un nouvel élément est défini par **<dom-module>** et l'attribut **id** donne son nom. On trouve ensuite la partie **template** qui reprend le menu que nous avions créé dans notre « brouillon » d'application. En ligne 22, on enregistre le *Web component* en faisant appel à **Polymer()()** et en indiquant dans la clé **is** le nom du **dom-module**. Dans les lignes 23 à 25, la clé **properties** permet de définir un attribut **display** contenant une chaîne de caractères. Cet attribut désigne l'**id** de l'élément servant à afficher les pages.

La suite du code définit les fonctions appelées sur les actions **on-tap** des lignes 14 et 15. **this.display** permet de récupérer la valeur fournie en attribut et la méthode **select()** change la page (qui sera définie par la suite).

Revenons au bouton de connexion de la barre d'outils se trouvant dans **glmf_components/glmf-login-button/glmf-login-button.html** :

```
01: <link rel="import" href="../../bower_components/polymer/
polymer.html">
02: <link rel="import" href="../../bower_components/paper-icon-
button/paper-icon-button.html" />
03: <link rel="import" href="../../bower_components/iron-icons/
iron-icons.html" />
04:
05: <dom-module id="glmf-login-button">
06:
07:     <template>
08:         <paper-icon-button name="test" icon="lock-outline"
on-tap="menuLogin"></paper-icon-button>
09:     </template>
10:
11:     <script>
12:         Polymer({
13:             is: 'glmf-login-button',
14:             properties: {
15:                 display: String
16:             },
17:             menuLogin: function () {
18:                 document.getElementById(this.display).
select('login');
19:             },
20:         })();
21:     </script>
22:
23: </dom-module>
```

Ici, on retrouve la même structure que dans le *Web component* précédent en plus simple puisque nous n'affichons qu'un bouton.

La gestion de l'affichage des pages se fait à l'aide du *Web component* **glmf-pages** défini dans **glmf_components/glmf-pages/glmf-pages.html** :

```
01: <link rel="import" href="../../bower_components/polymer/
polymer.html">
02: <link rel="import" href="../../bower_components/iron-pages/
iron-pages.html" />
03: <link rel="import" href="../../bower_components/paper-
material/paper-material.html" />
04:
05: <link rel="import" href="pages/glmf-page-home.html" />
```

```
06: <link rel="import" href="pages/glmf-page-page1.html" />
07: <link rel="import" href="pages/glmf-page-page2.html" />
08: <link rel="import" href="pages/glmf-page-login.html" />
09:
10: <dom-module id="glmf-pages">
11:
12:     <template>
13:         <iron-pages id={{display}} selected={{selected}} attr-for-selected="data-
route">
14:                     <paper-material elevation="0" data-route="home">
15:                         <glmf-page-home></glmf-page-home>
16:                     </paper-material>
17:
18:                     <paper-material elevation="0" data-route="page1">
19:                         <glmf-page-page1></glmf-page-page1>
20:                     </paper-material>
21:
22:                     <paper-material elevation="0" data-route="page2">
23:                         <glmf-page-page2></glmf-page-page2>
24:                     </paper-material>
25:
26:                     <paper-material elevation="0" data-route="login">
27:                         <glmf-page-login></glmf-page-login>
28:                     </paper-material>
29:         </iron-pages>
30:     </template>
```

C'est l'élément **iron-pages** qui va nous permettre d'afficher nos pages. Notez les termes entre accolades sur la ligne 13 : ce sont les valeurs des attributs de même nom qui sont fournies à **glmf-pages** qui sont réutilisées. Chaque page est ensuite définie dans un bloc **paper-material** et le contenu se trouve dans un *Web component* que nous créerons spécialement pour cela.

```
32:     <script>
33:         Polymer({
34:             is: 'glmf-pages',
35:             properties: {
36:                 display: String,
37:                 selected: String,
38:             },
39:         })();
40:     </script>
41:
42: </dom-module>
```

Ici il n'y a pas d'action spéciale à définir, seulement les attributs à indiquer en ligne 36 et 37.

Enfin chaque page est définie dans le répertoire **glmf_components/glmf-pages/pages** dans un fichier dédié. Voici l'exemple de **glmf-page-home.html** :

```
01: <link rel="import" href="../../../bower_components/polymer/polymer.html">
02:
03: <dom-module id="glmf-page-home">
04:
05:     <template>
06:         <h1>Page d'accueil</h1>
07:         blablabla blablabla
08:     </template>
09:
10:     <script>
11:         Polymer({
12:             is: 'glmf-page-home',
13:         })();
14:     </script>
15:
16:
17: </dom-module>
```

Conclusion

Finalement c'est assez simple... quand on est le concepteur de l'élément. Avouez qu'il est difficile de lire ce code qui fait appel à de multiples fichiers. Je voulais profiter de cet article pour montrer que Polymer était certes redoutable d'efficacité, mais qu'il fallait prendre garde de ne pas abuser de la création de *Web components* au risque de rendre votre application impossible à maintenir. Dans le cas du menu que j'ai présenté, on pourrait incriminer le découpage, car il aurait été bien plus pratique de n'avoir qu'un seul élément « menu » dans **index.html** quitte à ce que cet élément soit ensuite la composition d'une multitude d'autres éléments... Oui, mais le problème c'est qu'au moment où j'ai effectué mes tests certains éléments ne supportent pas d'être enfouis dans d'autres. Il faut donc faire très attention à la façon dont vous allez structurer vos pages et vos éléments, mais il serait dommage d'abandonner un si bel outil à cause de cela ! ■

Référence

[1] Colombo T., « *Faisons un peu de chimie html avec Polymer* », *GNU/Linux Magazine n°189*, janvier 2016.

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

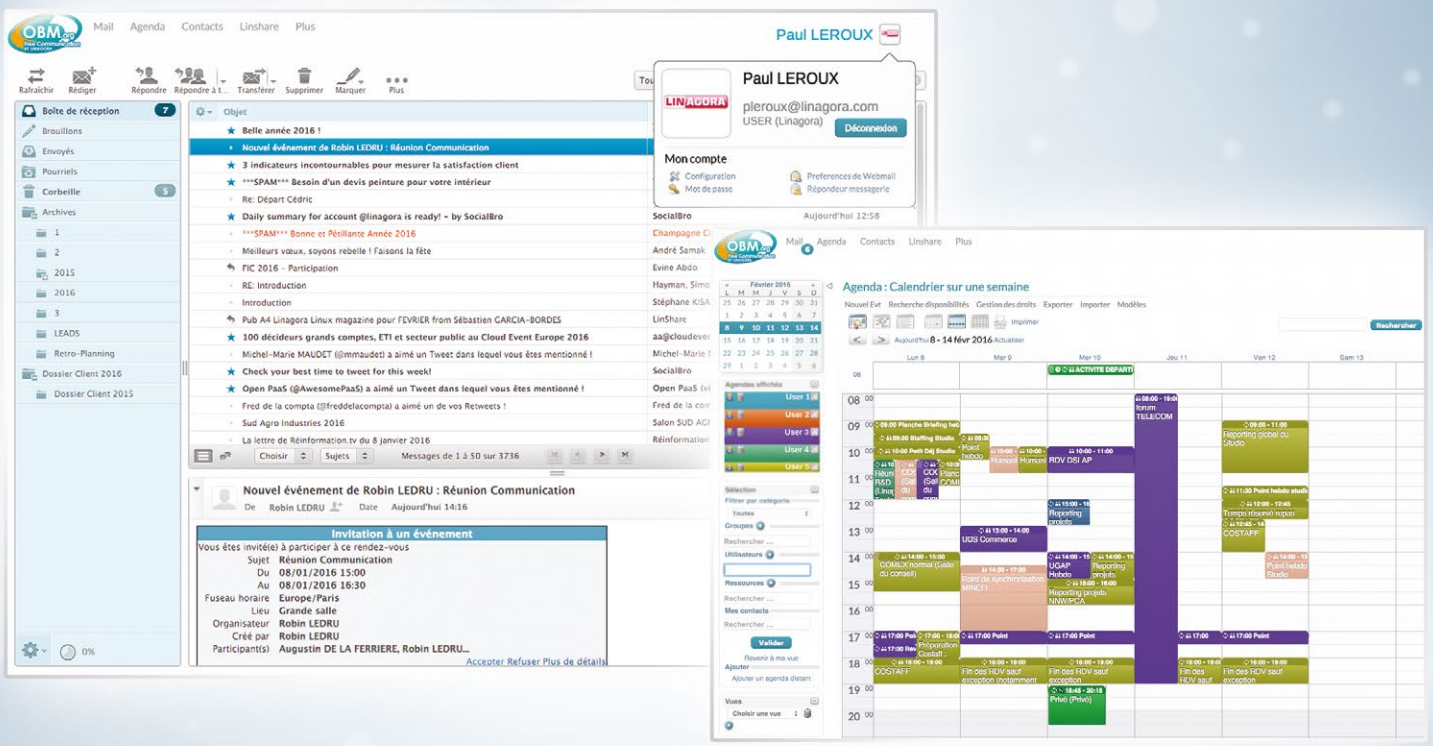
EX10

Cloud Hybride
Exchange
Lync
Sharepoint
Plateforme Collaborative

sales@ex10.biz
01 84 01 02 53
www.ex10.biz

LINAGORA

VOUS AUSSI ADOPTEZ
OBM.ORG !



LA SOLUTION LIBRE DE MESSAGERIE COLLABORATIVE LA PLUS UTILISÉE EN FRANCE

OBM.org

EMAIL · CALENDRIER · RESSOURCES · CONTACTS · TELEPHONE

www.linagora.fr/OBM

DÉCOUVREZ NOS AUTRES SOLUTIONS LIBRES !

LinShare — Partage et transfert sécurisé de fichiers

LinID — Gestion et fédération des identités

LinPKI — PKI et signature électronique

petals ESB — ESB et orchestration

OpenPaaS — Plateforme collaborative

www.linagora.com

@Linagora