PROGRAMMEZ !

#209 - juillet/août 2017

le magazine des développeurs

Spécial été !

Je monte et je code mon robot

Le coin Android

- Je crée mon bloqueur de pubs pour mon téléphone
- J'apprends le nouveau langage Kotlin

Tu veux programmer des apps mobiles ?

Avec Xamarin, c'est facile !

Vieilles machines

Je découvre l'Oric 1, l'Apple II, le Commodore 64, l'Atari ST

M 04319 - 209 - F: 6,50 € - RD

Printed in EU - Imprimé en UE - BELGIQUE 7 € - Canada 9,80 $ CAN - SUISSE 13,10 FS - DOM Surf 7,50 € - TOM 1020 XPF - MAROC 55 DH

OPÉRATION POUR 1 EURO DE PLUS

Pour bénéficier de cette offre exceptionnelle, il suffit de commander WINDEV Mobile 22 (ou WINDEV 22, ou WEBDEV 22) chez PC SOFT au tarif catalogue avant le 13 juillet 2017. Pour 1 Euro de plus, vous recevrez alors le ou les magnifiques matériels que vous aurez choisis. Offre réservée aux sociétés, administrations, mairies, GIE et professions libérales, en France métropolitaine. L'offre s'applique sur le tarif catalogue uniquement. Voir tous les détails sur : **WWW.PCSOFT.FR** ou appelez-nous au **04.67.032.032**
Le Logiciel et le matériel peuvent être acquis séparément. Tarif du Logiciel au prix catalogue de 1.650 Euros HT (1.980,00 TTC). Merci de vous connecter au site www.pcsoft.fr pour consulter la liste des prix des matériels. Tarifs modifiables sans préavis.

PROLONGATION JUSQU'AU 13 JUILLET

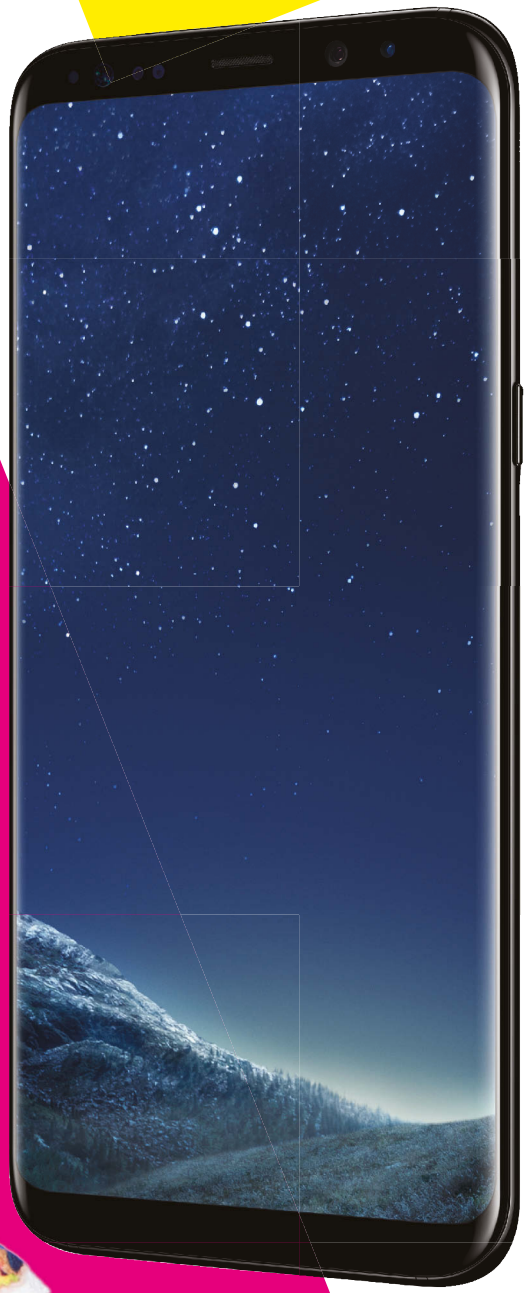
COMMANDEZ WINDEV MOBILE 22

OU WEBDEV 22 OU WINDEV 22

ET RECEVEZ LE NOUVEAU

Galaxy S8 | S8+

Choix de la couleur sur le site



CHOISISSEZ:

- Galaxy S8
  ou
- Galaxy S8+
  ou
- Smart TV 4K 140 cm
  ou
- Tablette Galaxy Tab S3+clavier
  ou
- 2xTablettes Galaxy Tab S2 9,7"

(Détails et autres matériels sur www.pcsoft.fr)

Atelier de Génie Logiciel Professionnel

DAS Galaxy S8 = 0,315 W/kg – DAS Galaxy S8+ = 0,260 W/kg Le DAS (débit d'absorption spécifique des appareils mobiles) quantifie le niveau d'exposition maximal de l'utilisateur aux ondes électromagnétiques. La réglementation française impose que le DAS ne dépasse pas 2W/Kg. L'utilisation d'un kit mains libres est recommandée.

Tél Paris: 01 48 01 48 88
Tél Montpellier: 04 67 032 032

WWW.PCSOFT.FR

Summer coding*

Ce n'est pas parce que c'est l'été qu'il ne faut pas coder, ni jouer du fer à sonder ou de son imprimante 3D. A Programmez!, il est de tradition de faire un numéro estival un peu plus léger mais très dense. Nous n'allons pas déroger à cette bonne pratique.

Nous allons nous replonger dans le passé et remonter le temps de 30 – 35 ans. Nous parlerons des ordinateurs mythiques des années 80 et 90 : Commodore, Oric, ZX, CPC, Atari, Amiga, Apple II. Nous ferons un peu de programmation old school** et verrons qu'avec quelques Ko, on faisait des miracles !

Pour le côté ludique et programmation, notre dossier maker / IoT va vous plaire. Nous parlerons robotique. Aymeric propose un dossier sur comment monter et coder son premier robot de A à Z : impression des pièces, la partie électrique et bien entendu, le code qui permet de faire vivre notre robot.

Vous ne développez pas encore des apps mobiles ou vous voulez vous lancer sur la plateforme Xamarin ? Notre dossier Xamarin est fait pour vous ; vous y découvrirez les services et outils disponibles qui vous aident à créer de belles apps mobiles.

En mai dernier, Google dévoilait l'intégration d'un nouveau langage de programmation dans Android : Kotlin. Nous vous proposons de découvrir le langage, sa syntaxe, les outils compatibles.

TROLL DU MOIS

Quelle est la différence entre un écran bleu et un kernel panic ? Le premier est une espèce commune, le second est si rare qu'on ne le voit qu'une fois par an…

La rédaction vous souhaite un excellent été et rendez-vous le 1er septembre avec le numéro 210 !

François Tonic
ftonic@programmez.com

** programmation d'été*
*** vieille école*

Dans le prochain numéro !
Programmez! #210, dès le 1er septembre 2017

Cobol, mainframe : toujours là !
Certaines technologies résistent à tout. Le mainframe et le cobol en sont la parfaite illustration. Ils restent au coeur de nombreuses administrations et grandes entreprises. Faut-il les garder, les migrer ou les adapter ?

Les bases de données
On en parle peu et pourtant la base de données est partout. Pour stocker, traiter, analyser des données, le SGBD est incontournable. SQL ou NoSQL ? Base locale ou dans le cloud ?

Softbank : la robotique japonaise n'a jamais été aussi puissante !

Le géant japonais Softbank devient un acteur robotique mondial. Après le rachat d'Aldebaran (Nao / Pepper), c'est au tour de Boston Dynamics d'être racheté. Boston Dynamics se spécialise dans les robots de grandes tailles et capable d'agir sur tous les terrains. Google l'avait acheté en 2013 mais depuis plusieurs mois, il cherchait un acheteur. Le Japonais a aussi racheté le constructeur robotique Schaft, détenu aussi par Google.

Si Google avait de grandes ambitions dans la robotique, il avait bien du mal à le concrétiser. Et surtout, la branche robotique gardait une grande autonomie et était éloignée de l'esprit Google. Surtout, le budget nécessaire ne cessait d'augmenter avec un robot vendable peu certain. Google a coupé d'autres projets ambitieux.
Pour Softbank, comme l'avenir de l'Homme passe par la robotique pour son patron, ces rachats sont d'une logique implacable et ils ne vont pas s'arrêter là : que ce soit dans des rachats ou la recherche. Le Japonais est désormais un acteur incontournable de la robotique et il investit aussi dans les sociétés technologiques comme Nvidia au printemps dernier. Il y a un an, il rachetait l'Anglais ARM pour 31 milliards $ avant de céder 25 % à Vision Fund, un fond d'investissement créé par SoftBank. A terme, Vision Fund pèsera 100 millards $.

LE TOP 10 DES ENTREPRISES QUI FONT RÊVER LES JEUNES INFORMATICIENS FRANÇAIS

1. **Google**
2. **Microsoft**
3. **Ubisoft**
4. **Apple**
5. **Thales**
6. **IBM**
7. **Amazon**
8. **Electronic Arts**
9. **Orange**
10. **Airbus**

Il est intéressant de constater que seules 2 entreprises viennent de l'industie (Thales et Airbus), 5 sont des fournisseurs de technologies et 2 sont des leaders du jeu…

Sur le secteur d'activité favori des étudiants du secteur, la conception de logiciels arrive largement en tête suivie des technologies de l'information, des télécoms, de l'aérospatiale et de la défense. Audit, conseil, banques arrivent loin derrière.

(source : universum)

Vers un reboot de **Windows Mobile ?** Microsoft serait en train de le faire en interne et sortira, ou pas, courant 2018.

Waymo – Google arrête la voiture Firefly qui avait servi à démontrer et à tester les technologies autonomes de conduite. R.I.P.

Intel veut démocratiser la connectique Thunderbolt 3. Il serait temps d'y penser.

Atari va-t-elle sortir une nouvelle console ? La réponse est oui.

Apple va tuer définivement les apps 32-bit. iOS 11 ne sera pas compatible (apps et matériel) et surtout, et c'est la surprise de la conférence développeur Apple, macOS aussi va les supprimer. A partir de janvier 2018 : les nouvelles apps seront forcément 64-bit et en juin 2018, toutes les applications et les mises à jour !

Google fait une belle prise de guerre en débauchant un des meilleurs experts en puces d'Apple. Apple a montré la voie en désignant les puces mobiles ARM. Google a l'ambition de faire la même chose même si c'est un travail énorme et qui prend plusieurs années.

INTEL PAS CONTENT !

Début juin, Intel publiait un long post de blog sur 40 ans d'innovations et d'évolution des processeurs x86 : puissance sécurité, mémoire, etc. Cette publication n'est pas totalement innocente.
Aujourd'hui, Intel est bousculé sur la partie mobile où le fondeur est quasi absent au niveau processeur. Les processeurs ARM règnent en maître et ils commencent à atteindre l'ordinateur et les serveurs. Ce n'est pas la première que ARM arrive sur ces machines mais là, le marché va connaître une bousculade sans précédent avec l'annonce des premiers PC ARM sous Windows 10. Mais pour assurer la compatibilité et ne pas perdre les utilisateurs, l'architecture x86 sera émulée pour exécuter les logiciels… En attendant le très long travail de recompilation et d'adaptation des codes actuels.

Cette perspective ne semble pas être très appréciée d'Intel même si ARM n'est pas cité… Intel veut protéger les innovations du x86 et le fondeur veille sur cette protection. Si certaines sociétés concurrentes respectent les droits d'Intel sur la compatibilté x86, d'autres non.

Intel lance-t-il ici un avertissement sans frais à Microsoft et aux constructeurs de PC ? On peut le penser. Mais entre l'avertissement et aller au clash, il y a un énorme fossé. Pour Intel, quel avenir du x86 ? Car les processeurs ARM continuent à monter en puissance.
Si aujourd'hui, ils ne peuvent pas prétendre rivaliser avec les plus puissants Xeon, dans 2 ou 3 ans, la situation aura évolué.

Intel sera-t-il condamné à laisser le x86 et à se tourner vers d'autres types de processeurs ?

OPÉRATION POUR 1 EURO DE PLUS

Pour bénéficier de cette offre exceptionnelle, il suffit de commander WINDEV Mobile 22 (ou WINDEV 22, ou WEBDEV 22) chez PC SOFT au tarif catalogue avant le 13 juillet 2017. Pour 1 Euro de **plus, vous recevrez alors le ou les magnifiques matériels que vous** aurez choisis. Offre réservée aux sociétés, administrations, mairies, **GIE et professions libérales, en France métropolitaine. L'offre s'ap-plique sur le tarif catalogue uniquement**. Voir tous les détails sur : **WWW.PCSOFT.FR** ou appelez-nous au 04.67.032.032
Le Logiciel et le matériel peuvent être acquis séparément. Tarif du Logiciel au prix catalogue de 1.650 Euros HT (1.980,00 TTC). Merci de vous connecter au site www.pcsoft.fr pour consulter la liste des prix des matériels. Tarifs modifiables sans préavis.

PROLONGATION JUSQU'AU 13 JUILLET

COMMANDEZ WINDEV MOBILE 22

OU WEBDEV 22 OU WINDEV 22

ET RECEVEZ LE NOUVEL iPhone 7

Choix de la couleur sur le site

iPhone 7 Plus

CHOISISSEZ :

- **iPhone 7 Plus** 128GB
  ou
- **iPhone 7** 256GB
  ou
- **2x iPad New** 9,7" 128GB
  ou
- **MacBook Air** 13,3" 128GB

(Détails et autres matériels sur www.pcsoft.fr)

Atelier de Génie Logiciel Professionnel

Tél Paris: 01 48 01 48 88
Tél Montpellier: 04 67 032 032

WWW.PCSOFT.FR

septembre

UBUCON EUROPE :

8, 9 &10 septembre / Paris

La communauté Ubuntu se réunira à Paris début septembre à l'UbuCon. Il s'agit de la 2e édition de la conférence européenne, la 1ere édition s'était déroulée en Allemagne. L'évènement s'annonce important : des dizaines de conférences et d'ateliers, 16 salles, 2000 m2 dédiés. Les communautés seront naturellement très présentes avec des espaces pros, communautaires et de nombreux thèmes seront abordés : système, smart city, infrastructure, etc.

Pour en savoir plus : https://ubucon.paris

DEVFEST TOULOUSE 2017 :

28 septembre / Toulouse

Le DevFest est une conférence technique destinée aux développeur.se.s. Elle s'adresse aussi bien aux étudiants, aux professionnels ou tout simplement aux curieux technophiles. Durant toute la journée, des orateurs & oratrices reconnus viendront présenter des sujets variés : autour du développement mobile, du Web, de la data, des objets connectés, du Cloud, du devops etc., ainsi que des bonnes pratiques de développement. Le DevFest Toulouse est organisé par les communautés de développeurs de Toulouse, et porté administrativement par le GDG Toulouse.

La première édition en 2016, a été un véritable succès et nous remettons donc le couvert cette année en augmentant le nombre de talks et de participants. L'édition 2017 aura lieu le 28 Septembre au Mega CGR de Blagnac et accueillera 450 participant.e.s.

Si vous avez un ou plusieurs sujets à nous soumettre, il n'y a pas de soucis, le CFP (Call For Papers) est ouvert jusqu'au 14 Juillet : https://devfest-toulouse.cfp.io/

La billetterie est également ouverte : http://bit.ly/devfesttoulouse-2017-billetterie

Plus d'infos sur : https://devfesttoulouse.fr

RGC 2017 :

30 septembre & 1er octobre / Meaux

Pour tous les amoureux de jeux et d'ordinateurs anciens, la RGC est une convention qui réunit plus de 300 personnes en automne. L'édition 2017 promet d'être très riche et active sur l'ensemble des machines : Atari, Amiga, MSX, CPC, etc. La RGC est réservée aux inscrits.

octobre

MICROSOFT EXPERIENCES'17 :

4 & 5 octobre / Paris

Microsoft experiences'17

Cette année, les MS Experiences parleront de 3 grands thèmes : l'intelligence artificielle, les nouvelles méthodes et pratiques de travail et la confiance numérique (blockchain, identité, etc.). L'évènement parlera bien entendu aux responsables IT et aux développeurs. Le format reste grosso modo identique à l'édition 2016 : plénières, ateliers, sessions.

DEVCON #4 :

26 octobre / Paris

Programmez ! organisera sa prochaine conférence technique le 26 octobre prochain. Le thème sera : 100 % Raspberry Pi & Co en production. On parlera maker / IoT, desktop, serveurs, etc. Comment ces cartes sont utilisées dans des projets et environnements de production ?

FORUM PHP 2017 :

26 & 27 octobre / Paris

FORUMPHP PARIS2017

L'édition 2017 se transporte de nouveau à Paris, à 2 pas de Denfert-Rochereau. 190 propositions de sessions ont été déposées. Un des focus sera le langage proprement dit : PHP 7 et l'avenir du langage.

Pour en savoir plus : event.afup.org

novembre

NI DAYS PARIS :

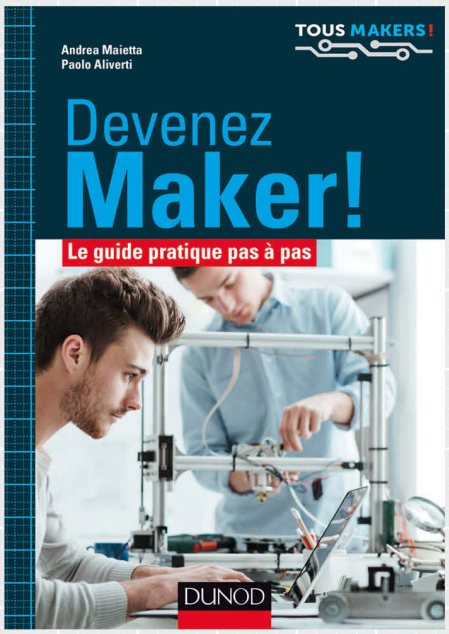
7 novembre / Paris

National Instruments est un constructeur / éditeur phare du monde de l'embarqué et de l'industrie. Chaque année, la journée NI Days permet de découvrir les dernières versions des outils d'instrumentation, l'écosystème. De nombreuses sessions sont jouées durant la journée.

Pour en savoir plus : https://www.ni.com/nidays/

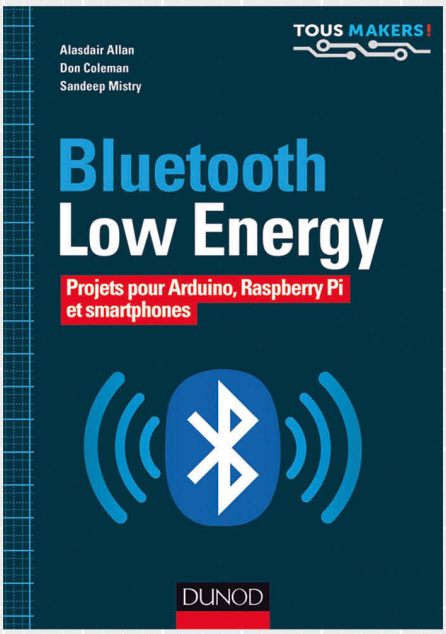
TOUS MAKERS !

RÉVÉLEZ VOTRE POTENTIEL PAR LA CRÉATION



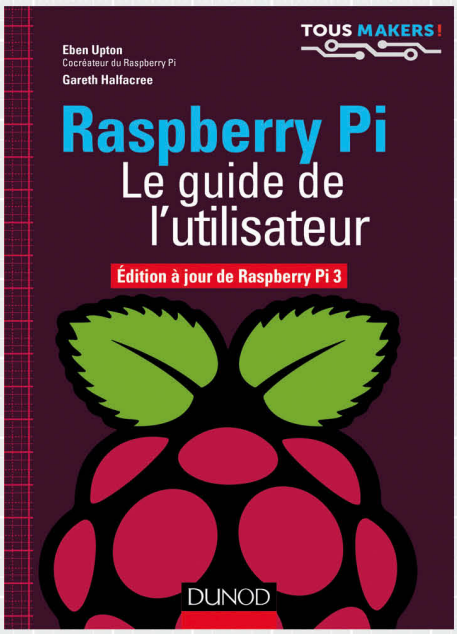
9782100762934, 304 pages, 24,90 €
ANDREA MAIETTA, PAOLO ALIVERTI
Comment transformer vos idées en projets concrets ? Ce livre vous accompagne dans la réalisation de vos premières créations.

9782100758487, 240 pages, 27 €
ALEX ELLIOTT
Le compagnon idéal pour vous guider pas à pas tout au long de la construction de votre drone.

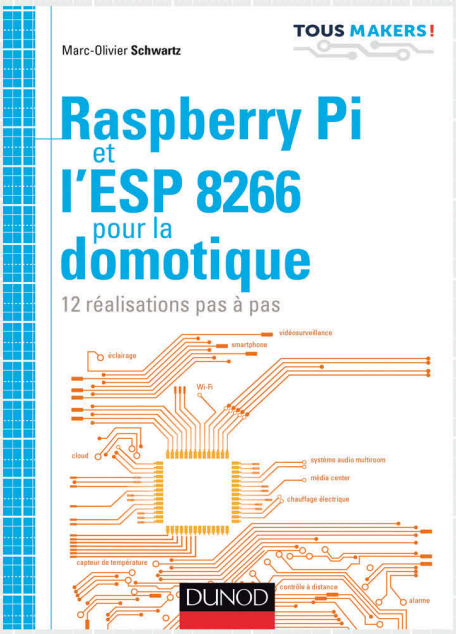


9782100760855, 272 pages, 29 €
ALASDAIR ALLAN ET AL.
Maîtrisez la nouvelle technologie Bluetooth Low Energy en réalisant les différents projets détaillés dans cet ouvrage.

9782100738106, 176 pages, 23 €
PATRICE OGUIC
Un guide pour construire votre machine CNC, idéale pour la gravure et le perçage des circuits imprimés, les maquettistes et les modélistes.



9782100762262, 288 pages, 26,90 €
EBEN UPTON, GARETH HALFACREE
Écrit par le co-créateur du Raspberry Pi, cet ouvrage donne toutes les clés pour tirer le meilleur parti du nano-ordinateur révolutionnaire.



9782100746835, 200 pages, 24,90 €
MARC-OLIVIER SCHWARTZ
Initiez-vous à la domotique avec le Raspberry Pi associé à la puce Wi-Fi ESP 8266 : 12 projets concrets pour rendre votre maison plus « intelligente ».

TOUT LE CATALOGUE SUR **WWW.DUNOD.COM**

DUNOD
ÉDITEUR DE SAVOIRS

Apple investit le marché de la **réalité augmentée**

François Tonic

Les concurrents l'attendaient. Les analyses étaient impatients. Les développeurs aussi, surtout pour les machines et les utilisateurs, on ne sait pas trop. Quoi qu'il en soit, avec la présentation officielle lors de la WWDC, Apple est désormais un acteur en puissance de la réalité augmentée, en attendant la réalité mixte et surtout le matériel adapté.

Apple travaille sur deux fronts :

- Les terminaux mobiles sous iOS ;
- Les machines capables de supporter l'usage et le développement de contenus de réalité virtuelle / augmentée.

Sur la partie desktop

Jusqu'à présent pour faire tourner des outils et des contenus Vive ou Occulus sur un Mac, il n'y avait pas grand chose, même rien. Apple a décidé de se bouger un peu et de proposer enfin du matériel capable de supporter en exécution et en développement les exigences de ces outils et casques. Pour compléter la partie purement matérielle avec les nouveaux iMac et

MacBook Pro, et le futur iMac Pro, Apple complète avec un kit spécial eGPU composé d'un carte Radeon RX 580 au prix de 599 $. Il supporte macOS High Sierra, le prochain macOS et les applications utilisant Metal, OpenCL, OpenGL. Il nécessite un Thunderbolt 3. Il nécessite Xcode 9, l'environnement de développement d'Apple.

Pour compléter ce tableau, Apple a travaillé avec les principaux fournisseurs de moteurs et d'outils VA / VR : Unreal, Unity, SteamVR. On notera aussi que le HTC Vive est désormais supporté officiellement par macOS.

Apple a aussi développé une nouvelle version de Metal : Metal 2. Metal est un framework pour le rendu graphique, la 3D, les traitements parallèles de données, il permet d'optimiser les traitements et les accès GPU et maintenant les GPU externes. Metal 2 sera optimisé pour la VR / VA (metal for VR). Le constructeur a révélé avoir travaillé avec Valve, Unity et Epic Games. Reste à voir comment Oculus réagit à ces annonces.

iOS mise pour le moment sur la réalité augmentée

Pour la partie iOS, Apple va déployer les premières technologies de réalité augmentée pour iPhone et iPad, en exploitant les capacités matérielles des terminaux mobiles et avec le tout nouveau ARKit.

En gros, ARKit doit fournir un motion tracking stable et rapide, une estimation de l'espace plat disponible, les limites (annoncées comme basique pour le moment), une estimation de l'échelle de son environnement, une analyse et prise en compte de la lumière, le support de Unity, Unreal et SceneKit et de nouveaux templates Xcode.

Les démos live ont montré les possibilités actuelles d'ARKit : on dépose des outils dans son environnement réel, manipulation d'objets complexes sur une table, le sol, etc. Plus complexe, la démo assez impressionante de Wingnut AR, un jeu issu du studio de Peter Jackson.

ARKit repose sur le principe de l'odométrie visuelle c'est à dire un procédé permettant de localiser dans l'espace un objet grâce à une caméra fixée sur un terminal / matériel. Il mappe donc le réel en utilisant les données fournies par la caméra et les données du CoreMotion. Cette double source de données doit permettre une précision de placement et de positionnement. Il détecte les surfaces planes horizontales dans une pièce et traque les objets virtuels / réels. Cela nécessite une puissance de calcul suffisante, ARKit nécessite minimum un processeur A9.

ARKit exige iOS 11 et Xcode 9.

Pour conclure

Soyons clairs, Apple a uniquement posté les premières briques techniques (outils, fonctions, matériels). Hormis la partie matérielle sur les gammes desktop et portables, rien n'a été annoncé sur les iPhone, iPad. Il faudra sans doute attendre le prochain iPhone en septembre pour en savoir plus.

Doit-on s'attendre à un casque ou des lunettes ? Ce serait la suite logique de ces annonces surtout que la concurrence est déjà là et que les casques compatibles Windows Holographic peuvent bousculer les acteurs actuels (Facebook, HTC, Sony). Google a fait de même. Cela peut laisser une large ouverture pour un autre acteur. •

DELL ANNONCE DES TOUT-EN-UN POUR LA VR

Le constructeur a annoncé les modèles Gaming Desktop, Inspiron 27 700 AIO et 24 500 AIO taillés pour la VR et la 4K. La disponibilité de ces modèles varie de début juillet à fin août. Il sera possible d'utiliser les Vive et Oculus.

La VR pour lutter contre la douleur

L'usage de la réalité virtuelle dans la médecine et certains traitements se développe. La fondation APICIL a récemment évoqué l'usage de la VR pour réduire la douleur notamment pour les jeunes enfants ayant une paralysie cérébrale. L'idée est ici de proposer une expérience ludique immersive pour réduire la douleur durant les traitements.

De **Geek à DSI** accompli grâce à mon Executive MBA

Jean-Philippe Hallot, @jphi_hallot
Directeur Technique chez **BK Consulting** (www.bk-consulting.com)
Executive MBA Epitech Promo 2017

Bon, j'exagère un peu dans le titre parce qu'en vrai j'étais déjà allé jusqu'à un bon niveau sans Executive MBA, mais en 2013 j'ai commencé à sentir un plafond de verre et des difficultés pour aller plus loin seul. J'étais alors directeur technique chez BK Consulting, ils m'avaient donné ma chance 3 ans plus tôt. J'avais mis en place plusieurs équipes chez nos clients et je gérais un département de 40 personnes.

Mes premières difficultés : comprendre et me faire comprendre

Quand on est directeur technique, on assiste aux comités de direction. J'avais le secret espoir de pouvoir mieux faire entendre ma fibre technologique pour faire avancer certains sujets. J'ai rapidement compris que les échanges seraient compliqués, un peu comme la tour de Babel : je parlais technique quand ils parlaient compta, commerce, stratégie, juridique.

Une première solution : l'autoformation

Quand un geek ne sait pas gérer, il cherche, j'ai cherché : à moi BFM Business, les sites spécialisés sur la finance, les Business Plans, la RH… J'ai pu apprendre un peu, mais sans guide, sans cadre, je n'ai pu acquérir qu'un vernis.

La solution : l'Executive MBA « Management IT » by Epitech

Cyril Pierre de Geyer (un geek à la base qui est également professeur affilié à HEC) m'a présenté l'Executive MBA Epitech qu'il a créé et qu'il dirige toujours. L'objectif de cette formation est de permettre à des profils comme moi, des geeks devenus managers, d'avoir une vision 360 ° de l'entreprise. L'idée n'est pas de devenir Directeur Marketing, mais d'être suffisamment à l'aise pour en challenger un. Idem pour la finance, le commerce, la stratégie…Ce que j'ai aimé c'est l'approche hors du temps de travail possible. La formation a été étalée sur 2 ans et son rythme est adapté pour rester salarié : 1/3 MOOC, 1/3 travaux de groupes et 1/3 ensemble, un samedi par mois.

Les MOOC

Les MOOC nous permettent de travailler à notre rythme, tout en restant en poste. Ils sont fournis sous forme de vidéos et de divers supports écrits associés. Tout est fait pour faciliter la compréhension. Nous avons un accès illimité aux cours, à n'importe quelle heure, online ou offline. Avantage non négligeable, nous pouvons revoir la formation autant de fois que nécessaire.

Le travail en groupe pour assimiler le cours

Nous étudions un sujet sur 1 mois. Les cours sont associés à un projet à rendre. Pas de panique, ce devoir se fait par groupe de 5 à 6 personnes. Cela rend l'exercice particulièrement instructif. En effet, chacun comprend le cours à sa manière et amène sa « pierre » au devoir à rendre. Peu à peu, à force d'échanges et de questionnement, nous comprenons le cours plus profondément. C'est presque magique. Cela fonctionne très bien. Ce qui est agréable aussi c'est de travailler dans les boites des uns, des autres. J'ai eu l'occasion d'aller chez BlaBlaCar, Criteo, Deezer…

Le présentiel : 1 samedi par mois

Un module se termine par un samedi en présentiel. C'est l'occasion de nous rencontrer physiquement, de rencontrer le professeur et de lui poser toutes questions utiles. La journée est généralement organisée en 3 temps. Tout d'abord, la revue des points essentiels du cours. Cela permet d'enlever nos dernières incompréhensions et de l'assimiler un peu plus. Ensuite, la présentation orale de nos devoirs. C'est l'occasion de progresser sur l'expression orale (No pain, no gain !). Enfin, nous terminons sur la correction du devoir puis une master class sur la thématique du présentiel.

Les précieux compléments

Vous l'avez compris, tout est organisé pour optimiser la compréhension et l'apprentissage en un minimum de temps en présentiel. Cela fonctionne merveilleusement bien. En plus de ce dispositif, nous avons de précieux compléments qui aident à ouvrir l'esprit, à imaginer de nouveaux lendemains et à mieux nous changer de l'intérieur :

- **Le coaching.** Pendant le cursus, nous sommes suivis par un coach certifié. C'est un travail très personnel sur notre fonctionnement individuel et nos relations de travail avec les autres, notre mode de management. Ces 20 entretiens d'1 heure m'ont permis de mieux comprendre mes motivations personnelles, mes objectifs et le chemin à suivre pour les atteindre. Cela permet de prendre de la hauteur, de l'autonomie et de rapidement gagner en efficacité. C'est une vraie plus-value de l'Executive MBA Epitech !
- **Le stage leadership au GIGN.** Ce fut passionnant, à la recherche de nos limites pour pouvoir les dépasser. Je n'en dis pas plus, je vous laisse découvrir. ;)
- **Les « master class »** en fin de présentiel. C'est très intéressant d'écouter un professionnel sur un sujet en lien avec le cours. Nous découvrons un nouvel angle de vue et de bonnes pratiques.
- **Les voyages d'études :**

La première année, nous sommes partis à San Francisco découvrir la Silicon Valley. Nous avons rendu visite à tous types de sociétés. Des plus petites aux plus grandes (Google, Facebook et Netflix par exemple), avec un passage chez Y Combinator qui a aidé à lancer des succès comme Airbnb, Dropbox, Docker. Chaque visite a été riche d'enseignement. L'écosystème technique à SF est vraiment très en avance. Nous sommes rentrés avec de nombreuses bonnes idées à mettre en place. La seconde année : Singapour. Là aussi, beaucoup de visites très variées (Google, MyRepublic, Business France...). Nous avons découvert l'état d'esprit asiatique et comment une entreprise française peut travailler localement.

Les changements

Pour revenir à mon cas personnel, oui, j'ai largement fluidifié et amélioré ma communication avec les autres directeurs chez BK Consulting. Je sais maintenant comment adapter mon message à chaque interlocuteur. Pour moi, c'est le savant dosage de tous ces éléments qui font que l'expérience Executive MBA Epitech a été une très grande réussite. J'ai changé. Ma vision du monde a changé. Tous mes collègues de promotion ont changé, en mieux ! Nous sommes maintenant tous optimistes dans notre avenir, tous convaincus que nous pouvons dépasser les limites du possible. •

Abonnez-vous à programmez!

le magazine des développeurs

Nos classiques

1 an — **49€***
11 numéros

2 ans — **79€***
22 numéros

Etudiant — **39€***
1 an - 11 numéros

** Tarifs France métropolitaine*

Abonnement numérique

PDF — **35€**
1 an - 11 numéros

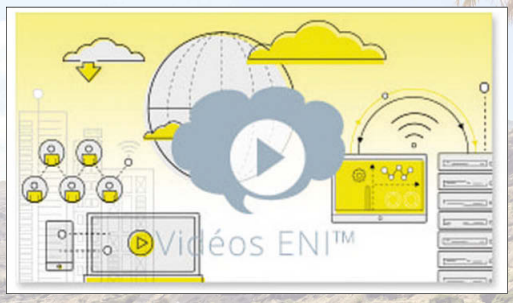
Souscription uniquement sur
www.programmez.com

Option :
accès aux archives 10€

Nos offres spéciales été 2017

1 an — **59€**
11 numéros + 1 vidéo ENI au choix :
- Big Data avec Hadoop
- Framework Spring

(Valeur de la vidéo de 29,99 à 59,99 €)



2 ans — **89€**
22 numéros + 1 vidéo ENI au choix :
- Big Data avec Hadoop
- Framework Spring

(Valeur de la vidéo de 29,99 à 59,99 €)

Offre limitée à la France métropolitaine

Toutes nos offres sur www.programmez.com

Oui, je m'abonne

ABONNEMENT à retourner avec votre règlement à :
Service Abonnements PROGRAMMEZ, 4 Rue de Mouchy, 60438 Noailles Cedex.

☐ **Abonnement 1 an :** 49 €
☐ **Abonnement 2 ans :** 79 €
☐ **Abonnement 1 an Etudiant :** 39 €
Photocopie de la carte d'étudiant à joindre

☐ **Abonnement 1 an :** 59 €
11 numéros + 1 vidéo ENI au choix :
☐ **Abonnement 2 ans :** 89 €
22 numéros + 1 vidéo ENI au choix :

☐ Vidéo : Big Data avec Hadoop
☐ Vidéo : Framework Spring

PROG 209
Offre limitée, valable jusqu'au 1er septembre 2017

☐ Mme ☐ M. Entreprise : Fonction :

Prénom : Nom :

Adresse :

Code postal : Ville :

email indispensable pour l'envoi d'informations relatives à votre abonnement

E-mail : @

☐ Je joins mon règlement par chèque à l'ordre de Programmez !
☐ Je souhaite régler à réception de facture

*** Tarifs France métropolitaine**

L'été des vieilles machines

Remontons le temps de presque 40 ans pour revenir aux premiers micro-ordinateurs : les Apple II, les Oric, les Commodore et autres Atari et Amiga. Avec quelques Ko, les programmeurs et bidouilleurs faisaient des merveilles ! Nous avions la vraie passion du matériel, cette âme nous l'avons un peu perdue et les ordinateurs d'aujourd'hui ne provoquent plus la même excitation que nous avions. Profitons de l'été pour revoir nos classiques et faire de la programmation old school.

La rédaction

Oric 1 & Oric Atmos

• Mickaël Pointier
Programmeur outils chez **Funcom**
Repousse les limites des « ordinosaures » de son enfance pendant son temps libre
mike@defence-force.org

Les ordinateurs Oric 1 et Atmos furent élus « ordinateur de l'année » respectivement en 1983 et 1984 en France. Leur succès fut probablement plus causé par leurs prix très raisonnables que par leurs performances, mais ces machines étaient probablement les choix idéaux pour apprendre à programmer.

1

Dans les années 80, l'essentiel du marché était occupé par des machines équipées de processeurs 8 bits (en grande majorité basés sur sur le microprocesseur MOS 6502 ou Z80 de Zilog), 64 kilo-octets de mémoire vive, et utilisant soit des cassettes audio, soit des disquettes 5.25 pouces pour stocker les informations. [1]

Ce qui différenciait les différents produits provenait surtout des composants supplémentaires pour aider le processeur à gérer l'affichage plus efficacement, ou à avoir la capacité à produire des sonorités plus évoluées que de simples « BIPs ». Cela provenait aussi de la qualité du logiciel dans la mémoire morte de la machine, ou de la présence de connecteurs d'extension.

Je ne vais pas maintenir le suspens plus longtemps : la raison pour laquelle les machines Oric étaient si peu chères est essentiellement due à la simplicité et aux limitations de la conception de la machine.

Minitel où es-tu ?

En France nous avions le Minitel, les Anglais avaient leur propre standard Vidéotex : le Prestel, abréviation signifiant « Press Telephone ». Originellement les machines Oric n'étaient pas conçues pour les joueurs, la cible était la petite entreprise, la comptabilité familiale ou encore l'utilisation comme terminal bancaire à distance.

Au cœur de la machine se trouve donc l'ULA HCS 10017, un composant capable d'afficher nativement des textes contenant des codes de contrôle vidéotex. L'ULA est aussi en charge de l'arbitrage de bus, rafraîchissement de la mémoire, permettant ainsi de réduire le nombre de composants sur la carte mère. Autour de l'ULA nous trouvons aussi un microprocesseur conçu par MOS Technologie le 6502 cadencé à 1 Mhz, 48 kilo-octets de mémoire vive utilisable (en fait il y a 64 kilo-octets de mémoire installée, mais elle n'est accessible que si un lecteur de disquette est connecté), plus 16 kilo-octets de mémoire morte hébergeant un BASIC Microsoft étendu.

Pour finir, le composant VIA 6522 est capable de fournir des chronomètres programmables, lancer des interruptions, et permet l'accès au générateur de son trois voix AY-3-8912 de General Instrument. A noter que le générateur de son lui-même est utilisé pour accéder au port imprimante (compatible Centronics) et de lire l'état des touches sur le clavier.

Idéal pour débuter

L'Oric est au Commodore 64 ce que l'Atari ST est à l'Amiga : juste un microprocesseur et de la mémoire directement accessible.

Pas de gestion de Sprites ou de Player de Missiles, pas de Copper ou de Display List, et pas non plus de Blitter. Pour les joueurs, le résultat est malheureusement immédiatement visible, les jeux sont souvent plus lents, moins jolis, et moins réactifs que sur les autres machines. Pour le programmeur potentiel par contre cela présente de nombreux avantages : il pourra se consacrer à l'apprentissage de la programmation grâce à la simplicité de la machine qui rend l'entreprise bien plus simple tout en limitant les espoirs quant aux résultats atteignables !

S'il n'en fallait qu'un...

Le manuel d'utilisation de l'Oric Atmos est une référence du genre, et il est bien dommage que l'on ne trouve plus que rarement ce genre de documents pour les produits modernes : complet, progressif, exhaustif et utilisable comme référence même lorsque l'on a atteint la maîtrise de la machine. [2]

2

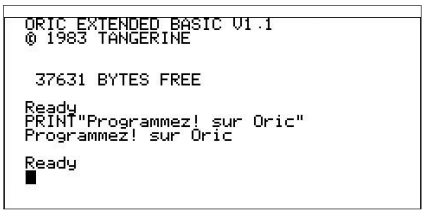
Lorsque l'utilisateur a finalement atteint la page 318 du manuel, il aura appris comment brancher sa machine (et les différents périphériques), ce que signifie le mot B.A.S.I.C. et les informations affichées au démarrage, comment charger et sauver des programmes, comment éditer du code et comprendre les messages d'erreur, la notion de variable et de type (numérique, chaîne de caractère), fait de petits calculs (TVA, circonférence), avant de monter en difficulté en abordant les limites nu-

mériques, encodage des valeurs flottantes, calcul hexadécimal et binaire, etc.
Les chapitres défilent, présentant la programmation structurée (boucles FOR, IF THEN ELSE, GOSUB RETURN, ON GOTO, etc.), les instructions graphiques et sonores, et même une partie de type « Inception » comme le film présentant un petit programme montrant comment le programme est stocké en mémoire par le système, une initiation à la programmation en assembleur 6502 et une série d'annexes contenant les schémas électroniques, emplacement des variables et fonctions systèmes, description de chaque broche de chaque connecteur, et même une section consacrée à la programmation de l'imprimante traceur.

Langages

Il faut environ une seconde pour démarrer un Oric et exécuter une instruction en langage BASIC. Lorsque le message Ready apparaît, il suffit d'entrer une instruction et valider. Par exemple :

```
PRINT"Programmez! sur Oric"
```



Durant la vie commerciale de la machine, l'essentiel des logiciels étaient écrits en BASIC, certains incorporant des routines en assembleur 6502 pour les parties critiques. Il était aussi possible de goûter aux joies de la programmation en LOGO ou Forth si vous pouviez vous procurer le logiciel adéquat. Nous sommes maintenant au 21e siècle, il serait dommage de se limiter à ces seuls choix !

Développement

Le problème du développement natif sur une machine limitée est qu'il est impossible d'utiliser 100% des potentialités. Sur une machine disposant de seulement 64 kilo-octets de mémoire, la mémoire utilisée par l'éditeur, le compilateur, le linker, le système d'exploitation, le débogueur, etc. ne sont pas disponibles pour l'application qui est développée.
Ce problème est résolu par l'utilisation d'un environnement de développement croisé qui est souvent composé d'un ensemble d'outils utilisés pour générer un produit prêt à l'usage sur la plateforme cible, ainsi qu'un émulateur et débogueur permettant de tester rapidement.
Dans cet article je vais utiliser le OSDK, mais il existe d'autres options, tel que CC65.

OSDK

Le OSDK contient un compilateur C, un assembleur 6502, un tokenizateur BASIC, des bibliothèques de fonctions standard, de la documentation et des exemples, ainsi que des outils pour convertir les données graphiques et sonores, compresser les fichiers, ou générer des disquettes au format émulateur.

Il contient aussi un émulateur et un débogueur symbolique. Tous les composants sont optionnels, et le code source est disponible.
Voyons comment écrire un simple programme, le fameux « Hello World », en utilisant le BASIC, le C et l'Assembleur.

Principes de fonctionnement

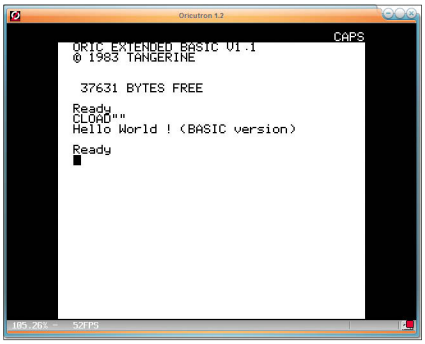
Les premières versions de l'OSDK ne fonctionnaient que sous MS-DOS, ce qui explique la présence de nombreux fichiers scripts BAT et de variables d'environnement.
Typiquement un projet OSDK est définit par trois fichiers qui sont l'équivalent de ce que pourrait faire un fichier script makefile, l'avantage étant que l'on peut facilement utiliser le système en mode graphique en double cliquant les fichiers avec l'extension BAT.
Le script osdk_built.bat sert à lancer la génération du projet et est rarement modifié.
Le script osdk_execute.bat est utilisé pour exécuter le programme dans une nouvelle session émulateur.
Le script osdk_config.bat contient les paramètres du projet à compiler, définis par l'intermédiaire de variables d'environnement, dont essentiellement:
OSDKNAME – Nom du programme
OSDKFILE – Modules à compiler
OSDKADDR – Adresse du programme
A noter : certains projets contiennent aussi un fichier osdk_makedata.bat qui doit être lancé pour générer les données graphiques et sonores.
Le répertoire /sample contient un certain nombre d'exemples prêts à compiler démontrant comment le système fonctionne.

Hello World (en BASIC)

Vous trouverez la version BASIC dans le sous répertoire basic/hello_world.
Le fichier main.bas contient le code source du programme :

```
10 '
20 ' This is the standard
30 ' "HELLO WORLD" sample
40 ' written in BASIC
50 '
60 PRINT "Hello World ! (BASIC version)"
```

En appelant osdk_build, le programme est converti au format fichier cassette utilisé par les émulateurs Oric et se trouve dans le sous répertoire BUILD sous le nom HW.tap.
La compacité est dû au format qui est « tokenizé » (opération qui consiste à remplacer les *mots clefs* par une valeur numérique unique les identifiants chacun), chaque instruction occupe seulement un octet en mémoire et c'est l'interpréteur BASIC en ROM qui se charge du décodage.
En lançant osdk_execute, l'émulateur Oricutron est appelé et le programme directement exécuté.



Notez que le programme complet occupe seulement 157 octets !

Hello World (en C)

Vous trouverez la version C dans le sous répertoire /c/hello_world_simple.
Le fichier main.c contient le code source du programme :

```
//
// This is the standard "HELLO WORLD" sample
//

#include <lib.h>

void main()
{
  printf("Hello World !\n");
}
```

La procédure est identique et le résultat similaire, a l'exception de la taille du programme de 1045 octets !

Comparé à la version BASIC, le code est directement exécutable par le microprocesseur, bien moins compact, mais aussi bien plus rapide à l'exécution. Si vous êtes curieux vous pouvez regarder le contenu du fichier linked.s dans le sous-répertoire osdk/TMP. Il contient le code assembleur généré par le compilateur C ainsi que les fonctions des bibliothèques utilisées par le programme avant que l'assembleur ne soit invoqué.

Hello World (en Assembleur)

La version 6502 est dans le sous répertoire /assembly/hello_world_assembly.
Le fichier main.s contient le code source du programme :

```
_main
  ldx #0

read
  lda message,x
  bne write
  rts

write
  sta $BB80,x

  inx
  jmp read

message
  .asc "Hello World !",0
```

Ce programme ne fait pas exactement la même chose que les versions en BASIC et en C : les exemples précédents utilisent la fonction d'affichage du système qui prend en charge le déplacement du curseur, et qui lorsque l'affichage a atteint la fin de l'écran lance le défilement vers le haut. Cette version assembleur se contente d'afficher le message Hello World à un emplacement fixe situé sur la première ligne de l'écran. La taille totale de l'exécutable est de 47 octets, incluant 14 octets d'informations d'entête du format cassette.

Performance et taille

Comparer la performance des différents langages est relativement difficile. Au niveau du code, le BASIC est de loin le langage le plus compact. Le format « tokenisé » permet de stocker une large quantité de code compliqué de façon efficace, ce qui rend aussi les programmes rapides à charger. D'un autre côté, la gestion des variables, tableaux, ou données en général, n'est pas particulièrement efficace, le

BASIC ne connaissant que deux types de données: chaîne de caractère, et valeurs en virgule flottante.
Les programmes C pourraient être plus efficaces, malheureusement nous avons ici un vieux compilateur qui n'est pas très à l'aise avec le jeu d'instruction limité du 6502. Malgré tout la performance des programmes en C est largement supérieure à l'équivalent en BASIC.
L'assembleur pur est de loin la meilleure solution sur cette machine, donnant accès à de nombreuses optimisations impossibles à effectuer en langage C; contrairement au BASIC, il peut se passer du système d'exploitation et donc permet de récupérer 16 kilo octets de mémoire vive supplémentaire. Pour donner un ordre de grandeur : la différence de vitesse entre le code C et le code assembleur fonctionnellement équivalent et d'un facteur 16 a 64 selon la complexité du code.
Sur du code simple la différence n'est pas trop importante, mais dès que l'on commence à jongler avec les variables, les boucles multiples, les conditions de sorties, etc. le code du compilateur devient totalement inefficace.
On retrouve le même ordre de grandeur entre le C et le BASIC. [3]
Exemple concret : remplir l'écran en cyclant avec toutes les couleurs disponibles

```
10 HIRES
20 FOR I=0 TO 8000
30 : POKE #A000+I,16+(I AND 7)
40 NEXT I
```

Cette simple version en BASIC prend environ 1 minute et 47 secondes pour remplir tout l'écran.

```
void main()
{
  char* screen;
  hires();
  for (screen=(char*)0xa000;
       screen<(char*)(0xa000+8000);
       screen++)
  {
    *screen=16+((int)screen & 7);
  }
}
```

Ce code C équivalent ne prend que deux secondes pour remplir l'écran, c'est à dire plus de 50 fois plus rapide que la version BASIC.

```
_main
  jsr _hires

  ldy #200
loop_y
  ldx #0
loop_x
  txa
  and #7
  ora #16
__write
  sta $a000,x
  inx
  cpx #40
  bne loop_x

  clc
  lda __write+1
  adc #40
  sta __write+1
  bcc skip
  inc __write+2
skip

  dey
  bne loop_y
  rts
```

Et pour terminer, cette version assembleur qui fait la même chose en environ une seconde et qui serait facilement optimisable en déroulant le code.

Ressources

Le but de cet article n'était pas de faire de vous un programmeur Oric mais simplement de vous donner un aperçu de ce qui est faisable.
Si vous être intéressé, n'hésitez pas à télécharger le OSDK, à vous inscrire sur le forum des développeurs de Defence Force, voire même à vous abonner au magazine mensuel du Club Europe Oric. •

Quelques liens utiles:

- OSDK : http://osdk.org
- Forums Defence Force : http://forum.defence-force.org
- Oric.org : https://www.oric.org
- Source code des projets : http://miniserve.defence-force.org/svn/users/dbug/programmez/

Vous êtes aussi le bienvenu sur notre canal IRC #Oric sur IRCnet!

Coder une **cracktro** sur Amiga

Partie 1

• Denis Duplan
sociologue et développeur à ses heures.
Blog : http://www.stashofcode.fr

A l'occasion de la sortie du deuxième documentaire de la formidable série From bedrooms to millions cette fois consacré à l'Amiga de Commodore, prenons un instant pour revisiter ce qui fut un des nombreux aspects de la scène de l'époque : la production de cracktros. Nous prendrons pour exemple une cracktro codée il y a un quart de siècle pour le célèbre groupe Paradox (Figure 1).

Quelques nostalgiques de l'épopée de l'Amiga ont mis en ligne des vidéos de cracktros sur YouTube, dont cette dernière. Par exemple, vous pouvez voir et entendre ce qu'elle donne à l'URL suivante : https://www.youtube.com/watch?v=UQ6R32HAp4Y

Cette cracktro a été choisie parce que son code a été retrouvé et qu'elle exploite le Blitter et le Copper, deux des coprocesseurs de l'Amiga dont il sera ainsi possible de souligner l'originalité de l'architecture.

Cet article est le premier d'une série de deux. Après avoir rappelé comment mettre en œuvre un environnement de développement en assembleur 68000 dans le contexte d'un émulateur Amiga, il présentera un des deux coprocesseurs graphiques, le Blitter. Le second article présentera l'autre coprocesseur graphique, le Copper, et fera une petite synthèse sur l'intérêt que revenir sur le passé peut présenter aujourd'hui.

Vous pouvez télécharger le code (à peine un gros millier d'instructions en assembleur 68000) et les données de la cracktro à l'URL suivante : http://www.stashofcode.fr/code/coder-une-cracktro-sur-amiga-1/cracktro.zip

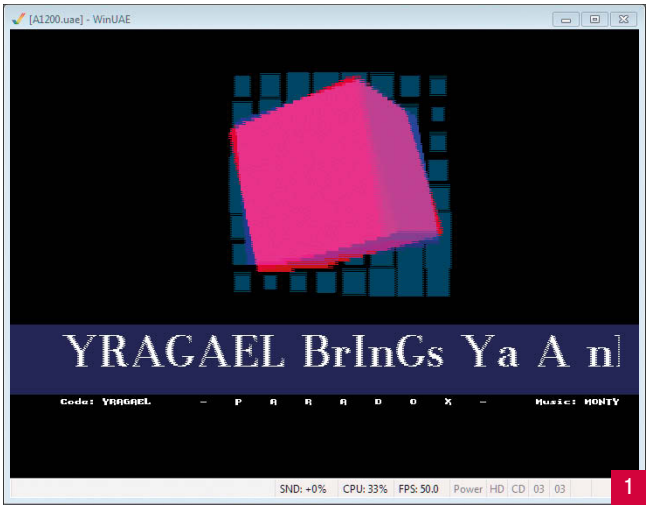
Emuler l'Amiga dans Windows

WinUAE est l'émulateur par excellence pour faire revivre l'Amiga dans le contexte de Windows. Toutefois, il ne suffit pas de le récupérer pour pouvoir l'utiliser. Il faut aussi se procurer la ROM de l'Amiga, le Kickstart (au moins dans la version 1.3). Par ailleurs, pour pouvoir compiler la cracktro, il faut installer le système d'exploitation, le Workbench (au moins dans la version 1.3). Kickstart et Workbench sont encore soumis à des droits. Ils sont commercialisés pour une dizaine d'euros par Amiga Forever.

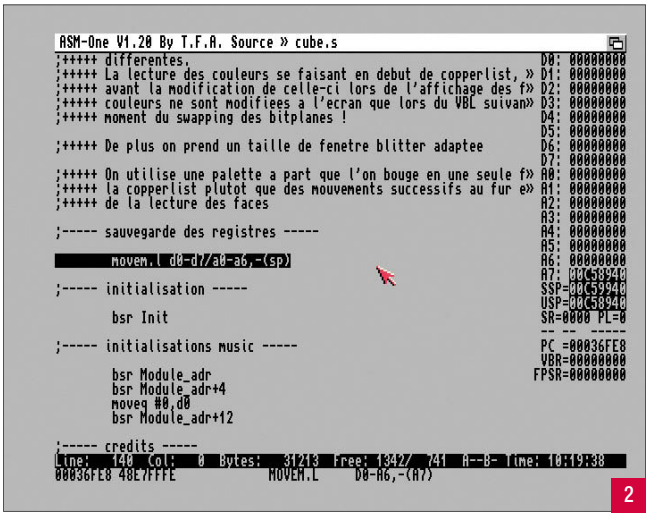
Comme toute démonstration d'un savoir-faire en matière de programmation, de graphisme et de musique digne de ce nom à l'époque, la cracktro a été codée en assembleur 68000. Dans cet article, nous nous concentrerons sur la partie du code qui gère les effets graphiques (d'ailleurs, qu'il soit rendu à César ce qui revient à César : la partie qui joue la musique est « Coded by Lars 'ZAP' Hamre/Amiga Freelancers » et « Modified by Estrup/Static Bytes »).

Il n'est pas question ici de s'étendre sur la manière d'utiliser WinUAE. L'objet de cet article est de rappeler ce que c'était que de programmer directement le hardware de l'Amiga et d'essayer d'en tirer quelques leçons pour aujourd'hui, pas d'inciter à programmer de nouveau de cette manière. Seuls les coders de l'époque pourraient être motivés à l'idée de compiler la cracktro à partir des données mises à disposition, et encore. Ceux-là seront nécessairement assez chevronnés pour s'y retrouver dans WinUAE et créer une configuration inspirée de l'Amiga 500 ou version ultérieure à mémoire étendue et émulation de disque dur. Les quelques informations fournies ci-après leur permettront de se rafraîchir la mémoire pour assurer la suite.

Pour compiler le source et le lier avec les données au sein d'un exécutable, il vous faudra utiliser ASM-One (Figure 2). Après avoir installé le Workbench sur une émulation de disque dur (le classique volume DH0:), vous devrez installer reqtools.library dans le répertoire Libs du système pour



La cracktro en cours d'exécution dans WinUAE.



Exécution de la cracktro en mode Debug dans ASM-One.

faire fonctionner ASM-One. Vous devrez aussi utiliser une commande du Shell pour assigner SOURCES: au répertoire contenant le code et les données (par exemple : assign SOURCES: DH0:cractros si vous avez déposé le contenu de l'archive dans un sous-répertoire cube qui s'y trouve). Après avoir lancé ASM-One, vous pourrez allouer un espace de travail en mémoire quelconque (Chip ou Fast) de 100Ko par exemple, puis charger le source via la commande R (Read) et le compiler via la commande A (Assemble), non sans avoir spécifié qu'il faut ignorer la casse en vous rendant dans le menu Assembler, sous-menu Assemble pour activer l'option UCase=LCase. Vous pouvez ensuite enregistrer l'exécutable via la commande WB (Write Binary).

Pour ce qui concerne la documentation, la référence de tout program-

meur (nous dirons « coder » pour revenir à l'esprit de l'époque) était le *Amiga Hardware Reference Manual* (Addison-Wesley en avait édité une édition plus agréable à lire). Précis et clair car rédigé par les ingénieurs de Commodore eux-mêmes, le manuel donnait toutes les informations requises pour s'adonner à ce que d'aucuns dénonçaient comme du *metal bashing*, c'est-à-dire la programmation directe du hardware en assembleur en court-circuitant le système d'exploitation, d'ailleurs totalement coupé le temps de l'exécution.

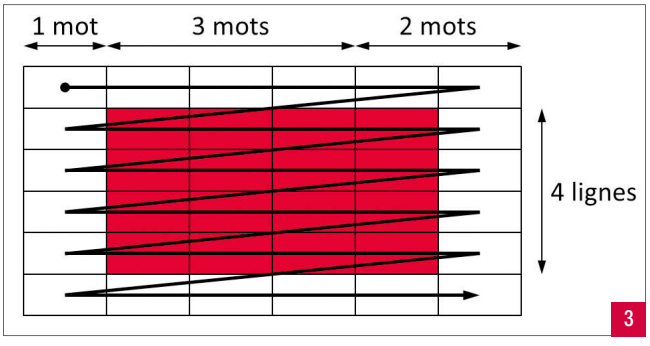
Tout codeur français qui a fait ses armes dans les années 80 sait la petite guerre qui a opposé les adorateurs de l'Atari ST et de l'Amiga, les deux machines 16 bits en concurrence sur le marché national. Objectivement, il faut noter que les capacités de l'Amiga dépassaient de loin celles de l'Atari ST dans le domaine du graphisme. C'est qu'en la matière, le coder pouvait s'appuyer non seulement sur le processeur Motorola 68000, mais aussi et surtout sur deux coprocesseurs très puissants : le Blitter et le Copper. Commençons par présenter le premier. Le Blitter est capable de copier des blocs et de tracer des droites. Son contrôle s'effectue en positionnant des bits de registres 16 bits résidant à des adresses spécifiques (par exemple, le registre de contrôle du Blitter BLTCON0 se trouve à l'adresse $DFF040 ou $00DFF040 pour être plus exact, le 68000 adressant la mémoire sur 32 bits).

Copier des blocs de mémoire

Dans sa fonction de copie, le Blitter peut lire mot par mot (16 bits) jusqu'à trois blocs à des adresses différentes (les sources A, B et C) et les combiner bit à bit. Pour chaque bloc, il faut spécifier l'adresse 32 bits du premier mot via les registres BLTxPTH et BLTxPTL correspondant au mot de poids fort et au mot de poids faible de l'adresse en question. Il faut aussi spécifier le modulo via le registre BLTxMOD, c'est-à-dire le nombre d'octets à ajouter à l'adresse du dernier mot d'une ligne du bloc pour adresser le premier mot de la ligne suivante. Par exemple, sur la figure (Figure 3), le modulo est de 3 mots (la mémoire est un espace d'adressage linéaire, évidemment, mais le Blitter permet donc de se la représenter comme une surface). Une fois les adresses (obligatoirement paires) spécifiées, il suffit de spécifier la largeur et la hauteur des blocs (obligatoirement identiques) dans le registre BLTSIZE. Un bloc peut faire jusqu'à 1024 x 1024 bits (un peu moins en largeur si le décalage et le masquage présentés plus loin sont utilisés). L'unité étant le mot, la hauteur est spécifiée à l'aide de l'octet de poids fort de BLTSIZE, et la largeur à l'aide de son octet de poids faible, soit en pseudo-code :

```
BLTSIZE = (hauteur << 8) + largeur
```

BLTSIZE est un strobe, c'est-à-dire un registre dont le simple accès en écriture déclenche une action, en l'occurrence la copie attendue.



Adressage d'un bloc de mémoire par le Blitter.

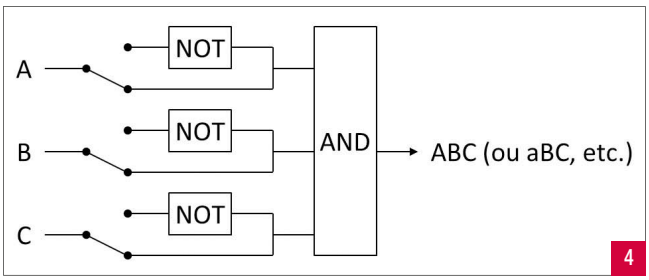
La combinaison des sources correspond à une combinaison par OR de huit combinaisons par AND des sources. Nécessairement assez complexe, la formule générale est spécifiée à l'aide d'une combinaison de bits de l'octet de poids faible d'un des registres 16 bits de contrôle du Blitter, BLTCON0 (X désigne le bit de la source X et x désigne le NOT de ce bit) :

Combinaison	Bit de BLTCON0
abc	0
abC	1
aBc	2
aBC	3
Abc	4
AbC	5
ABc	6
ABC	7

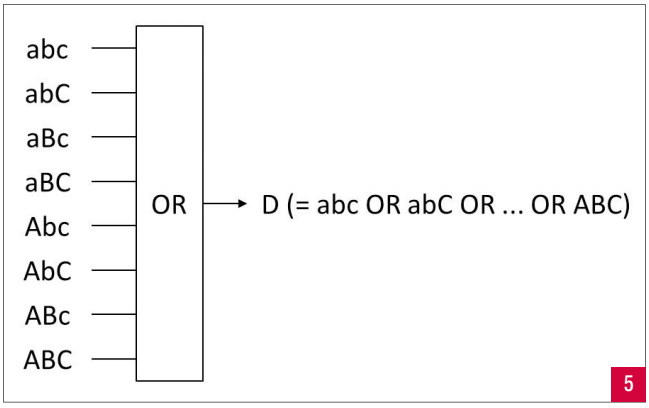
Ainsi, le bit de la destination D résultant de la combinaison des bits issus des sources A, B et C est d'abord calculé de huit manières en combinant par AND des bits des sources éventuellement inversés (Figure 4).

Puis ces huit versions de D sont combinées par OR pour déterminer le bit final (Figure 5). Ce fonctionnement général peut être raffiné, car les registres de contrôle du Blitter BLTCON0 et BLTCON1 permettent de spécifier bien d'autres choses :

- Activer ou désactiver les sources A, B et C et la destination D (désactiver la destination permet de simuler une copie pour vérifier, via un bit de contrôle positionné par le Blitter, si elle n'a généré que des bits à 0 : un moyen pour tester sans calcul une collision au pixel près). En fait, toutes les sources sont toujours combinées et le résultat renvoyé sur la destination en transitant mot par mot par des registres de données BLTxDAT. Toutefois, en désactivant une source, il est possible de figer la valeur du registre de données correspondant, comme si le même mot était lu à une adresse fixe (sauf qu'il n'est donc pas lu en mémoire, étant simplement lu dans BLTxDAT dans lequel il est possible d'écrire le mot souhaité avant la copie).



Première phase de la combinaison bit à bit des sources par le Blitter.



Seconde phase de la combinaison bit à bit des sources par le Blitter.

- Décaler de 0 à 15 bits sur la droite les mots lus aux sources A et B (pas C). Il s'agit d'un décalage par barillet, c'est-à-dire que tout bit sorti sur la droite du mot lu à l'adresse X est réintroduit sur la gauche du mot lu à l'adresse X+2. Et pour le premier mot d'une ligne du bloc adressé par une source, direz-vous ? Des 0 sont introduits sur sa gauche.
- Lire les blocs par adresse croissante ou décroissante. Cela permet d'écraser une source. Par exemple, pour remonter une image d'une ligne, il faut la copier par adresse croissante en partant du premier mot de la première ligne recopié au premier mot de la ligne du dessus (mode descendant). Pour la descendre d'une ligne, il faut la copier par adresse ascendante en partant du dernier mot de la dernière ligne recopiée au dernier mot de la ligne du dessous (mode ascendant).

Enfin, deux registres BLTAFWM et BLTALWM permettent de définir des masques à appliquer au premier mot et au dernier mot lus à la source A (pas B ni C). Pourquoi faire ? Pour détourer des sprites par exemple – des sprites logiciels, le Copper gérant des sprites, pour leur part matériels.

Dans la cracktro, une copie avec décalage réalisée à chaque trame par le Blitter permet de produire le scroll (un bloc à l'adresse calculée en a1 est recopié sur lui-même en mode ascendant en décalant ses bits du nombre de bits correspondant à la vitesse du scroll en pixels par trame) :

```
lea $DFF000,a5
moveq #2,d1         ;vitesse du scroll
ror.w #4,d1
bset #1,d1          ;mode ascending
movea.l Screen1_adr,a1
add.w #(ScrollHeight+FontHeight)*NbPlane1*SizeX1/8-2,a1
move.w #%0000010111001100,BLTCON0(a5)
move.w d1,BLTCON1(a5)
move.w #$0000,BLTDMOD(a5)
move.w #$0000,BLTBMOD(a5)
move.l a1,BLTBPTH(a5)
move.l a1,BLTDPTH(a5)
move.w #SizeX1/16+64*FontHeight*NbPlane1,BLTSIZE(a5)
```

Pour le détail, c'est donc 11001100 qui est stocké dans l'octet de poids faible de BLTCON0 pour copier à l'identique les bits de la source B dans la destination. En effet, la combinaison logique mise en œuvre est alors aBc + aBC + ABc + ABC, c'est-à-dire que le bit de la source B est retenu quelles que soient les valeurs des bits des sources A et C (par ailleurs désactivées). Mais j'oubliais... En même temps qu'il copie des blocs de mémoire, le Blitter peut remplir ces derniers en positionnant tous les bits rencontrés sur une ligne. Dans ce mode, le Blitter ne fait rien tant que le bit lu n'est pas 1, positionne tous les bits lus par la suite jusqu'à lire un bit 1, et reboucle alors. Il est possible d'inverser ce fonctionnement, contraignant donc le Blitter à positionner les bits lus dès le début tant qu'il n'a pas lu un bit 1, ne rien faire jusqu'à lire un bit 1, et reboucler alors (Figure 6). Il existe deux variantes de ce remplissage, l'une où les bits limitrophes à gauche sont maintenus (*inclusive-fill*) et l'autre où ces derniers sont effacés (*exclusive-fill*) (Figure 7).

L'*exclusive-fill*, pourquoi faire ? Pour permettre de produire des surfaces remplies très précises, où un sommet figurant sur une ligne n'était représenté que par un pixel et non les deux pixels juxtaposés qu'il faut bien dessiner avant de lancer le remplissage pour éviter que le Blitter ne remplisse la ligne n'importe comment.

Dans la cracktro, cette fonctionnalité est utilisée pour remplir les surfaces des cubes qui sont donc tracées en s'assurant de ne faire figurer qu'un point par ligne le long de chaque côté. Et pour tracer ces côtés, c'est... le Blitter qui est encore utilisé, comme nous allons le voir plus loin.

Pour terminer sur cette fonction de copie du Blitter, il faut noter que toutes les possibilités qui ont été évoquées (décalage, masquage, combinaison logique des sources et remplissage) peuvent être combinées sans pénalité, étant réalisées les unes après les autres dans un pipeline. Le manuel donne quelques conseils pour exploiter le remplissage du pipeline, mais il s'agit là d'un sujet particulièrement avancé – dont les ingénieurs de Commodore ne garantissaient par ailleurs pas la pérennité.

Tracer des droites

En positionnant un bit particulier de BLTCON0, il est possible de demander au Blitter non plus de copier un bloc en le remplissant ou non, mais de tracer une droite, d'au plus 1024 pixels, à l'aide d'un motif. Dans ce mode, le Blitter interprète différemment certains bits des registres BLTCON0 et BLTCON1.

Pour tracer une droite entre A (x_A, y_A) et B (x_B, y_B), il faut connaître :

- les coordonnées du point de départ ;
- l'octant du repère de centre A dans lequel se trouve B ;
- les valeurs absolues des différences d'abscisses et d'ordonnées.

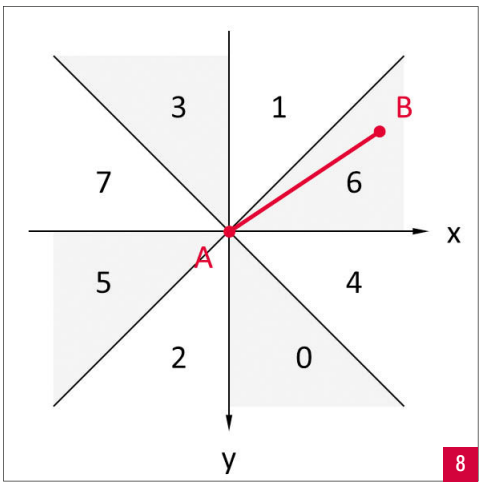
(Figure 8)

Pour rajouter à l'ésotérisme, le Blitter utilise bien le numéro de l'octant, mais il faut lui fournir la pente de l'image de la droite dans l'octant 6. Pour cette raison, des grandeurs dx et dy qui sont utilisées plus loin doivent être calculées ainsi :

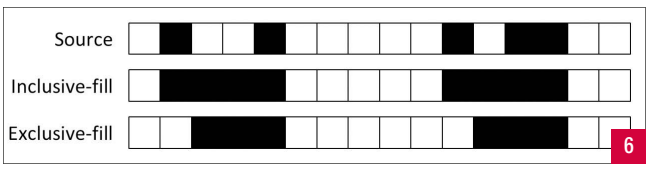
dx = max (abs (y_B - y_A), abs (x_B - x_A))

dy = min (abs (y_B - y_A), abs (x_B - x_A))

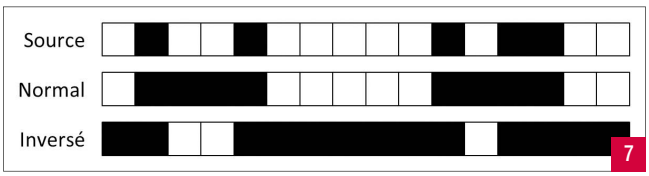
Divers registres doivent être utilisés pour fournir au Blitter toutes les informations dont il a besoin afin de tracer la droite AB :



Octants permettant de préciser la position de points d'une droite au Blitter.



Remplissage normal et inversé par le Blitter.



Inclusive-fill et exclusive-fill du Blitter.

Registre	Usage
BLTCPTH et BLTCPTL	Adresse du mot du bitplane contenant le bit correspondant au pixel A
BLTDPTH et BLTDPTL	Idem
BLTCMOD	Largeur d'un bitplane en octets
BLTDMOD	Idem
BLTAMOD	4 * (dy - dx)
BLTBMOD	4 * dy
BLTAPTL	(4 * dy) - (2 * dx)
BLTAFWM	$FFFF
BLTALWM	$FFFF
BLTADAT	$8000
BLTBDAT	Motif de la droite sur 16 pixels ($FFFF pour une droite pleine)
BLTCON0	La combinaison d'un certain nombre de bits qui doivent être positionnés par exigence système et de bits significatifs, ces derniers étant : - quatre bits correspondant à la position du pixel A dans le mot dont l'adresse est fournie via BLTCPTH/BLTDPTH et BLTCPTL/BLTDPT) ; - huit bits correspondant à la combinaison logique du pixel du masque fourni via BLTADAT, du motif fourni via BLTBDAT et de la destination tiré de BLCDAT.
BLTCON1	La combinaison d'un certain nombre de bits qui doivent être positionnés par exigence système et de bits significatifs, ces derniers étant : - quatre bits correspondant à la position du premier bit à utiliser pour tracer à la droite dans le motif fourni via BLTBDAT ; - un bit indiquant si (4 * dy) - (2 * dx) est négatif ; - trois bits donnant le numéro de l'octant ; - un bit indiquant si le Blitter ne doit tracer qu'un pixel par ligne de pixels dans le bitplane.
BLTSIZE	((dx + 1) << 6) + 2

Le tracé d'une droite est une opération de copie de bloc combinant trois sources A, B et C où C correspond au bitplane dans lequel la droite doit être tracée. A est le bit à positionner dans le bitplane pour y tracer un pixel de la droite, B le motif qui doit déterminer si le pixel courant de la droite doit effectivement être tracé dans le bitplane. La combinaison des sources utilisée est donc D=AB+AC, mais d'autres peuvent être envisagées – notamment D=ABC+AB pour tracer une droite qu'il sera possible d'effacer en la traçant simplement de nouveau. Le tracé de droite est lancé comme une copie de bloc, en écrivant dans le registre BLTSIZE.
Dans la cracktro, après initialisation de certains registres (BLTALWM avait été oublié !)… :

```
move.w #$FFFF,BLTBDAT(a5)
move.w #SizeX0/8,BLTCMOD(a5)
move.w #SizeX0/8,BLTDMOD(a5)
move.w #$8000,BLTAFWM(a5)
move.w #$8000,BLTADAT(a5)
```

…le tracé de droite a été factorisé dans la routine suivante :

```
;*************** TRACE DE DROITES ***************
; Entrées : A5=$DFF000, A0=adresse bitplane, D0=Xi, D1=Yi, D2=Xf, D3=Yf
; Modifie : A6,D5,D6

DrawLine:

;----- ordonnancement des points -----

  cmp.w d1,d3
  beq DrawLine_End
  bge DrawLine_UpDown
  exg d0,d2
  exg d1,d3
DrawLine_UpDown:
  subq.w #1,d3

;------ calcul adresse de depart de la droite -----

  moveq #0,d6
  move.w d1,d6
  lsl.l #3,d6
  move.l d6,d5
  lsl.l #2,d5
  add.l d5,d6    ;d6=y1*nbre octets par ligne
  add.l a0,d6    ;+adresse depart bitplane
  moveq #0,d5
  move.w d0,d5
  lsr.w #3,d5
  bclr #0,d5
  add.l d5,d6    ;+x1/8

;----- recherche de l'octant -----

  moveq #0,d5
  sub.w d1,d3   ;d3=Dy=y2-y1
  bpl.b Dy_Pos
  bset #2,d5
  neg d3
Dy_Pos:
  sub.w d0,d2   ;d2=Dx=x2-x1
  bpl.b Dx_Pos
  bset #1,d5
  neg d2
Dx_Pos:
  cmp.w d3,d2  ;Dx-Dy
  bpl.b DxDy_Pos
  bset #0,d5
  exg d3,d2  ;ainsi d3=Pdelta et d2=Gdelta
DxDy_Pos:
  add.w d3,d3   ;d3=2*Pdelta

;----- BLTCON0 -----

  and.w #$F,d0
  ror.w #4,d0
  or.w #$B4A,d0

;----- BLTCON1 -----

  lea Octant_adr,a6
  move.b (a6,d5.w),d5
  lsl #2,d5
```

```
  bset #0,d5
  bset #1,d5

;----- BLTCON1, BLTBMOD, BLTAPTL, BLTAMOD -----

  WAITBLIT

  move.w d3,BLTBMOD(a5)
  sub.w d2,d3
  bge.s DrawLine_NoBit
  bset #6,d5
DrawLine_NoBit:
  move.w d3,BLTAPTL(a5)
  sub.w d2,d3
  move.w d3,BLTAMOD(a5)

;----- BLTSIZE -----

  lsl #6,d2
  add.w #66,d2

;----- lancement blitter -----

  move.w d5,BLTCON1(a5)
  move.w d0,BLTCON0(a5)
  move.l d6,BLTCPTH(a5)
  move.l d6,BLTDPTH(a5)
  move.w d2,BLTSIZE(a5)

;----- fin -----

DrawLine_End:

  rts
```

Comme déjà mentionné, ce tracé de droite s'appuie sur une fonctionnalité du Blitter qui permet de limiter le tracé de la droite à un pixel par ligne de pixels dans le bitplane (Figure 9). Si toutes les faces étaient tracées dans une même surface (Figure 9), il est clair que le Blitter ne pourrait pas remplir chacune correctement. Dans la cracktro, une astuce tient compte du fait qu'un cube n'expose jamais plus que trois faces simultanément, deux à deux partageant un côté et un seul, pour produire un tracé que le Blitter peut remplir. Soient A, B et C les trois faces visibles (qui peuvent n'être qu'une ou deux) à un instant donné. Dans un bitplane, A et C sont tracées en omettant soigneusement le côté qu'elles ont en commun, tandis que dans un autre bitplane, B et C sont tracées en prenant la même précaution (Figure 10). Dans ces conditions, il est possible de remplir chacun des bitplanes au Blitter sans difficulté si bien que :

- les bits des pixels de A sont à 1 dans le premier bitplane et à 0 dans le second ;
- les bits des pixels de B sont à 0 dans le second bitplane et à 1 dans le second ;
- les bits des pixels de C sont à 1 dans les deux bitplanes.

Comme c'est la combinaison des bits d'un pixel figurant dans les bitplanes qui donne le numéro de la couleur dans laquelle ce pixel doit être affiché ; chacune des faces peut ainsi être affichée dans une couleur particulière, moyennant un petit arbitrage attribuant les pixels du côté commun à A et B à l'une ou l'autre de ces faces – le remplissage est un *exclusive-fill*. Au final, il n'a ainsi fallu que deux opérations de remplissage au Blitter, et non trois, pour remplir trois faces de couleurs distinctes (Figure 11).

Un fonctionnement en parallèle du CPU

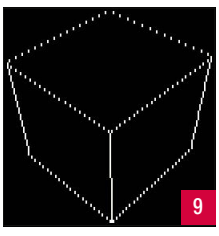
Pour terminer sur le Blitter, il faut noter que fonctionnant en parallèle du processeur car disposant d'un accès direct en mémoire (DMA) – il est même possible d'interdire au CPU de lui voler des cycles d'accès à la mémoire –, il suffit de lancer une copie (avec ou sans remplissage) ou un tracé de droites et reprendre comme si de rien n'était. In fine, il faut bien s'assurer que le Blitter avait terminé sa tâche, ce qui s'effectue en testant un bit du registre de contrôle DMACONR. Dans la cracktro, ce test revient fréquemment si bien que pour s'éviter d'en faire une routine à laquelle il aurait fallu sauter puis revenir en perdant des cycles d'exécution, elle figure sous la forme d'une macro – le test est doublé pour une raison très particulière que les amateurs pourront découvrir dans le manuel :

```
WAITBLIT: macro
Wait_Blit0\@
      btst #14,DMACONR(a5)
      bne Wait_Blit0\@
Wait_Blit1\@
      btst #14,DMACONR(a5)
      bne Wait_Blit1\@
      endm
```

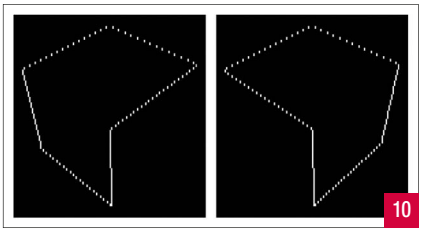
D'une manière générale, il n'y avait pas de petites économies en cycles d'exécution pour « tenir dans la trame » (ie : s'assurer qu'une image était reproduite à chaque balayage de l'écran, soit 50 fois par seconde dans le monde PAL, pour produire une animation fluide à l'écran), ce qui conduisait à privilégier la répétition de code, donc les macros telles que WAITBLIT, sur les appels à des routines. •

Liens utiles

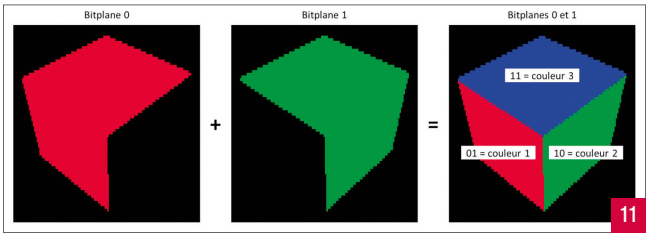
From Bedrooms to Billions: The Amiga Years :
http://www.frombedroomstobillions.com/amiga/
WinUAE : http://www.winuae.net/
Amiga Forever : https://www.amigaforever.com/
ASM-One : http://www.theflamearrows.info/documents/ftp.html
ReqTools : http://aminet.net/package/util/libs/ReqToolsUsr.lha
Amiga Hardware Reference Manual :
https://archive.org/details/Amiga_Hardware_Reference_Manual_1985_Commodore



Tracé de contours de faces en vue de leur remplissage par le Blitter.



Tracé des contours de surfaces contigües pour le remplissage par le Blitter.



Coloration des faces visibles.

Pascal UCSD sur **Apple II**

Partie 1

• Alain Zanchetta
Software Development Engineer dans l'équipe **Microsoft HoloLens.**

Alors qu'il a été le premier ordinateur personnel à proposer des graphismes en couleur et haute résolution, l'Apple II a rapidement été surclassé dans ce domaine en particulier, et dans tout ce qui touche au jeu en général par les machines plus récentes comme le Commodore 64. En revanche, il y a un domaine où l'Apple II n'a été égalé que bien plus tard, et ce, uniquement par des ordinateurs bien plus puissants, c'est dans la variété des langages de programmation disponibles. Nous allons nous intéresser aujourd'hui au Pascal UCSD, créé par l'Université de Californie de San Diego et diffusé par Apple.

Après un tour d'horizon du système, nous verrons comment écrire et exécuter un programme simple, puis quelles sont les spécificités de cette implémentation du Pascal sur Apple II. Nous finirons par une petite comparaison de ses performances avec le Basic Applesoft.

Le Système UCSD

Le Pascal UCSD ne peut être dissocié du système d'exploitation sur lequel il s'exécute : l'idée initiale de ses créateurs était de fournir un environnement capable de s'exécuter de manière identique sur différents types de machines, y compris des ordinateurs limités comme les premières machines 8 bits. Pour atteindre cet objectif, on a recours à une machine virtuelle avec son jeu d'instruction appelé p-code : ainsi, l'écriture d'un interpréteur de p-code est suffisante pour ensuite exécuter tous les programmes compilés pour cette machine virtuelle… Ce concept semble familier ? Effectivement, James Gosling (le concepteur du langage Java) indique dans un article de 2012 que l'idée de la machine virtuelle Java lui est venue d'un compilateur de P-Code qu'il avait écrit pendant ses études.

Le système UCSD est donc un environnement complet, comprenant son propre système d'exploitation et les outils associés et non un compilateur indépendant permettant de générer des programmes s'exécutant sur les systèmes d'exploitation naturels de l'Apple II, DOS 3.3 ou ProDOS (Kyan Pascal, apparu bien plus tard, comblera ce manque pour ProDOS). Ceci n'est pas gênant si on utilise le Pascal UCSD pour développer des programmes autonomes mais comme les transferts entre les différents systèmes d'exploitation de l'Apple II sont délicats, cet isolement interdit quasiment tout développement d'outil de manipulation de données comme par exemple les images qui pourraient être exportées par d'autres programmes ou sauvées par l'utilisateur depuis DOS 3.3.

Au niveau du matériel, utiliser le système UCSD demande au minimum 64 Ko de mémoire vive : les machines comme l'Apple II+ étaient généralement vendues avec 48Ko de mémoire et il fallait donc acheter une carte d'extension de 16Ko supplémentaires (c'est la raison pour laquelle ce type de carte était appelé Language Card). Une carte 80 colonnes est aussi la bienvenue même si le système fonctionne sans : appuyer sur CTRL-A permet de basculer entre les 40 colonnes de gauche et les 40 colonnes de droite…

Le lecteur de disquettes est bien évidemment indispensable – pas de Pascal sur cassette – et il est fortement recommandé d'en posséder deux. La documentation Apple indique comment utiliser le système UCSD avec un seul lecteur mais on passe alors son temps à changer de disquette : la plupart des commandes du système sont des programmes autonomes – éditeur, compilateur, assembleur, éditeur de liens, gestionnaire de fichiers – occupant une grande partie de l'espace disponible sur les 140 Ko des disquettes 5¼". Voici les quatre disquettes constituant le système Pascal UCSD de l'Apple II :

- « Apple0: » contient les programmes permettant de fonctionner avec un seul lecteur... Après avoir démarré sur la disquette Apple1 par manque de place sur une seule disquette !
- « Apple1: » contient le système, le gestionnaire de fichiers et l'éditeur de texte. C'est aussi elle qui contiendra le fichier de travail comme on le verra bientôt.
- « Apple2: » contient le compilateur, l'éditeur de liens, l'assembleur et les bibliothèques de base.
- « Apple3: » contient le système ainsi que quelques outils secondaires et des exemples de programmes.

Il y a eu quatre versions du Pascal UCSD pour l'Apple II : 1.0, 1.1, 1.2 et 1.3. Outre des améliorations de l'environnement et des corrections de bug, la version 1.2 apporte le support des 128 Ko de mémoire et la version 1.3 celui des disquettes 3½". Cette version 1.3 n'est pas disponible sur les principaux sites de téléchargement d'images disque pour Apple II comme le serveur ftp Asimov car elle est encore vendue par Syndicomm, qui possède d'ailleurs un assez large éventail d'outils de développement pour Apple II.

Contrairement aux systèmes d'exploitation habituels sur les machines de l'époque, le système UCSD ne présente pas à l'utilisateur une invite de commande mais se pilote via des menus. L'écran d'accueil permet de choisir la fonction à exécuter : chaque programme lancé depuis cet écran, ou chaque sous-menu, aura un fonctionnement similaire mais des commandes différentes. C'est un peu comme l'interface d'un Macintosh sans les graphismes ni la souris… Démarrons maintenant un Apple II avec la configuration standard à deux lecteurs de disquettes, c'est-à-dire « Apple1: » dans le premier lecteur et « Apple2: » dans le second. Le menu principal s'affiche en haut de l'écran, la version du système est confirmée par le [1.2] en fin de ligne et rappelée dans le message d'accueil, accompagnée de la taille mémoire du système (64 Ko ici). [1]

Le gestionnaire de fichiers se lance par la touche F (File) et présente l'ensemble de ses commandes dans un nouveau menu :

```
Filer: G(et, S(ave, W(hat, N(ew, L(dir, R(em, C(hng, T(rans, D(ate, Q(uit [1.2]
```

Le ? permet d'afficher quelques commandes supplémentaires :

```
Filer: B(ad-blks, E(xt-dir, K(rnch, M(ake, P(refix, V(ols, X(amine, Z(ero [1.2]
```

L permet d'afficher tous les fichiers d'un disque : il faut alors préciser quel lecteur on désire afficher. Comme le système UCSD est portable, la nomenclature classique par numéro de Slot et lecteur de disquette (S6, D1)

ne fonctionne pas, et tous les périphériques se voient attribuer un numéro. La Commande Vols du gestionnaire de fichiers permet d'afficher les volumes accessibles :

```
Filer: G(et, S(ave, W(hat, N(ew, L(dir, R(em, C(hng, T(rans, D(ate, Q(uit [1.2]
Vols on-line:
  1  CONSOLE:
  2  SYSTERM:
  4 # APPLE1:
  5 # APPLE2:
  6  PRINTER:
  7  REMIN:
  8  REMOUT:
Root vol is - APPLE1:
Prefix is   - APPLE1:
```

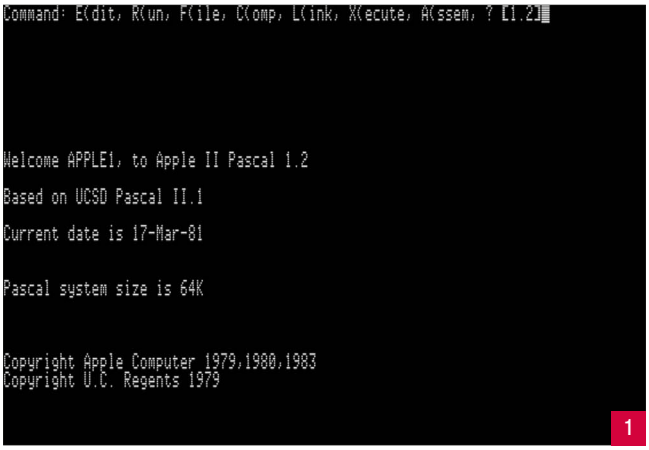
On peut voir en 4 et 5 les disquettes insérées. Des numéros supplémentaires sont affectés aux éventuels lecteurs connectés dans les ports 4 (volumes #9 et #10) et 5 (volumes #11 et #12), non disponibles dans l'exemple ci-dessus. Voici le contenu de la disquette nommée Apple1 :

```
Filer: G(et, S(ave, W(hat, N(ew, L(dir, R(em, C(hng, T(rans, D(ate, Q(uit [1.2]
APPLE1:
SYSTEM.APPLE     32 25-Dec-83
SYSTEM.PASCAL    43 25-Dec-83
SYSTEM.EDITOR    47 25-Dec-83
SYSTEM.FILER     29 25-Dec-83
SYSTEM.LIBRARY   39 25-Dec-83
SYSTEM.MISCINFO   1 25-Dec-83
SYSTEM.CHARSET    2 14-Jun-79
SYSTEM.SYNTAX    12 25-Dec-83
8/8 files<listed/in-dir>, 211 blocks used, 69 unused, 69 in largest
```

Cette liste amène une petite remarque : la synthèse indique le nombre de fichiers, le nombre de blocs utilisés (UCSD utilise des blocs de 512 octets comme ProDOS et non des blocs de 256 octets comme DOS 3.3), le nombre de blocs inutilisés et la taille du plus grand espace non inutilisé. En effet, le système UCSD n'est pas capable de stocker un fichier sur des blocs non contigus ! Il faut donc régulièrement utiliser la commande Krunch pour compacter ses disquettes et mettre tous les blocs libres à la fin. Les noms de disquettes se terminent par « : » et ceci permet de saisir en une seule étape le nom d'un fichier à manipuler dans le gestionnaire de fichier : ainsi « Apple1:Hello.text » désigne le fichier « Hello.text » sur la disquette « Apple1: ». A noter : la plupart des commandes du gestionnaire de fichier acceptent des métacaractères (wildcards) permettant de manipuler plusieurs fichiers à la fois.

Ecrire et exécuter un programme Pascal

Si nous sommes aujourd'hui habitués au cycle Edition – Compilation – Exécution, celui-ci était assez nouveau dans le monde des ordinateurs 8 bits. L'éditeur plein écran – après une prise en main assez délicate et quelques pertes de saisies – était en particulier très impressionnant sur un Apple II dont le Basic n'avait aucune possibilité de correction sérieuse. Ce cycle Edition – Compilation – Exécution s'appuie sur la notion de fichier de travail : le fichier de travail source – SYSTEM.WRK.TEXT – est la cible de l'éditeur et du compilateur, sa version compilée – SYSTEM.WRK.CODE – est le programme que la commande RUN va lancer (la commande XECUTE du menu principal permet de lancer n'importe quel fichier .CODE présent sur le disque). Sur un système vierge,



1

le premier lancement de l'éditeur va proposer la création d'un nouveau fichier de travail :

```
>Edit:
No workfile is present. File? ( <ret> for no file <esc-ret> to exit )
:
```

Pour commencer un nouveau programme, il suffit d'appuyer sur RETURN ; sinon il est possible de saisir le nom du fichier à modifier.
La fonction Get du gestionnaire de fichiers permet de désigner un fichier comme la cible par défaut de l'éditeur de texte mais un certain nombre de commandes vont malgré tout recréer ou cibler SYSTEM.WRK.TEXT…
L'éditeur de texte fonctionne un peu à la manière de « vi », basculant entre un mode de commandes et un mode d'insertion. On entre dans le mode d'insertion avec I… et on en sort via CTRL-C . Attention aux anciens réflexes car ESCAPE va tout simplement revenir en mode commandes après avoir abandonné toute la saisie effectuée…
Saisissons un programme avec une erreur :

```
>Edit: A(djst C(py D(lete F(ind I(nsrt J(mp R(place Q(uit X(chng Z(ap  [1.2]
Program Hello;

var i;

begin
   for i:=0 to 10 do
   begin
      writeln('Hello world');
   end;
end.
```

La commande QUIT permet de sauvegarder sur le disque le fichier modifié et on peut maintenant essayer de le compiler :

```
Compiling...




Apple Pascal Compiler [1.2]
<  0>..

var i; <<<<
Line 2, error 6: <sp>(continue), <esc>(terminate), E(dit
```

A la première erreur, le compilateur s'arrête et offre différentes possibilités : l'espace va lui faire reprendre la compilation et passer à l'erreur suivante, ESCAPE arrête la compilation et EDIT va non seulement relancer l'éditeur mais aussi positionner celui-ci sur la ligne fautive !

```
Illegal symbol (maybe missing or extra ';' on line above) .  Type <sp>
Program Hello;

var i;


begin
   for i:=0 to 10 do
   begin
      writeln('Hello world');
   end;
end.
```

Une fois le programme corrigé :

```
Program Hello;

var i : integer;

begin
   for i:=0 to 10 do
   begin
      writeln('Hello world');
   end;
end.
```

On peut lancer avec succès la compilation :

```
Apple Pascal Compiler [1.2]
<  0>.....
HELLO    [ 2271 words]
<  5>....
9 lines
Smallest available space = 2271 words
```

Le programme de travail se lance via la commande RUN :

```
Command: E(dit, R(un, F(ile, C(omp, L(ink, X(ecute, A(ssem, ? [1.2]
Running...
Hello world
Hello world
Hello world
Hello world
Hello world
Hello world
Hello world
Hello world
Hello world
Hello world
Hello world
```

Les spécificités de l'implémentation Apple II

Ni C++ ni Java n'existaient à l'époque de l'Apple II et le Pascal était donc le langage de choix pour l'enseignement de la programmation dans les grandes écoles et universités. A ce titre, le Pascal UCSD remplit totalement son rôle et il n'y a que très peu de constructions standards du Pascal qui ne soient pas supportées. Un exemple notable est l'impossibilité de passer une procédure ou fonction en tant que paramètre d'une autre procédure ou fonction.
Les limitations se situent donc essentiellement au niveau :

- des types supportés ;
- des bibliothèques disponibles ;
- et bien entendu de la mémoire disponible.

Les types

Le type INTEGER n'est représenté que sur deux octets, ce qui limite son champ d'action à l'intervalle [-32768, 32767]. Il existe en revanche un type appelé LONG INTEGER dans la documentation et qui peut représenter des nombres jusqu'à 36 chiffres :

```
program Factorial;

type long_integer = integer[30];

procedure Calc(n : integer;var f : long_integer);
 var i : integer;
begin
 f := 1;
 for i:=1 to n do
  f := i * f;
end;

procedure FactTest;
  var number : integer;
  fact : long_integer;
begin
 repeat
 begin
  Write('n=?');
  Read(number);
  if number > 0 then
  begin
   Calc(number, fact);
   write(number);
   write('! = ');
   writeln(fact);
  end;
 end
 until number = 0;
end;

(* Entry Point *)
begin
 FactTest;
end.
```

```
Command: E(dit, R(un, F(ile, C(omp, L(ink, X(ecute, A(ssem, ? [1.2]
Running...
n=?8
8! = 40320
n=?18
18! = 6402373705728000
n=?0
```

Les Bibliothèques

Les principales bibliothèques disponibles pour le Pascal UCSD de l'Apple II sont :

- Transcend : fonctions mathématiques (logarithmes, trigonométrie) ;
- Turtlegraphics : fonctions de dessins ;
- Applestuff : fonctions spécifiques à l'Apple II comme la lecture des Paddle ou l'accès aux sorties TTL du port jeu, ainsi qu'une fonction Note permettant d'émettre un son de tonalité et longueur paramétrable.

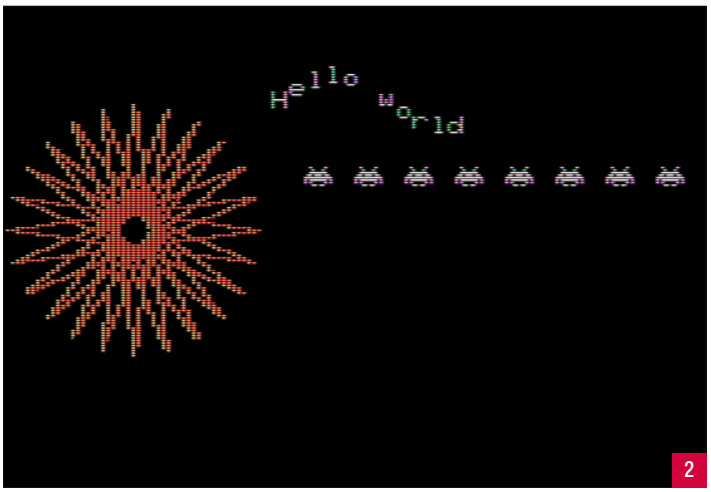
Attardons-nous un peu sur la bibliothèque **Turtlegraphics**. Comme son nom le laisse présager, une partie des fonctions qu'elle offre est inspirée du Logo. Il est néanmoins possible d'effectuer des tracés de manière classique en utilisant des coordonnées absolues. Cette bibliothèque apporte trois ensembles de fonctions :

- Le tracé de lignes ;
- L'affichage de texte en mode graphique : non seulement, c'est une fonctionnalité qui n'existait pas en Basic Applesoft (on parle bien ici d'afficher du texte n'importe où dans la page HGR et non des quatre lignes de texte en bas de l'écran qui étaient bien séparées des graphismes) mais il est possible de définir et utiliser ses propres polices de caractères ;
- L'affichage de bitmaps, avec la possibilité d'effectuer des opérations logiques entre le bloc et la partie de l'écran où on le dessine : c'est ce qui est le plus proche d'un sprite même si aucune fonction d'animation ou de détection de collision n'est disponible.

Le programme suivant illustre ces trois types de tracés.

```
program GraphDemo;

uses Transcend, TurtleGraphics;

type shape=packed array[0..7,0..10] of boolean;

var s : string;

procedure DrawStar(radius : integer; centerX : integer; centerY : integer);
var i, n, x, y : integer;
   angle : real;
begin
   n := 0;
   PenColor(none);
   for i:=0 to 24 do
   begin
      angle := n * 6.2831 / 24;
      x := centerX + round(radius * cos(angle));
      y := centerY + round(radius * sin(angle));
      MoveTo(x,y);
      PenColor(Orange);
      n := n + 13;
   end;
end;

procedure DrawMessage(s : string; startX : integer; startY : integer);
var i : integer;
begin
   for i:=1 to length(s) do
   begin
      PenColor(none);
      MoveTo(startX + 7 * i, startY + round(10 * sin((i-1)/2)));
      PenColor(white);
      Wchar(s[i]);
   end;
end;

procedure DrawShapes(x : integer; y : integer; n : integer);
   var sp1 : shape;
   procedure Parse(var sp : shape; row : integer; s : string);
      var col : integer;
   begin
      for col:=1 to length(s) do
        sp[row,col-1] := (s[col]<>' ');
   end;
begin
   Parse(sp1, 7, '  o     o  ');
   Parse(sp1, 6, '   o   o   ');
   Parse(sp1, 5, '  ooooooo  ');
   Parse(sp1, 4, ' oo ooo oo ');
   Parse(sp1, 3, 'ooooooooooo');
   Parse(sp1, 2, 'o ooooooo o');
   Parse(sp1, 1, 'o o     o o');
   Parse(sp1, 0, '   oo oo   ');
   for n:=1 to 10 do
      DrawBlock(sp1, 2, 0, 0, 11, 8, x + n * 20, y, 10);
end;


begin
   InitTurtle;
   DrawStar(50, 51, 102);
   DrawMessage('Hello world', 100, 150);
   DrawShapes(100, 120, 8);;
   read(s);
end.                                                    [2]
```

Suite au prochain numéro



2

Commodore 64 & la démoscène

• Jean-François Richard
Chef de Projet - **Sodifrance**
Co-fondateur **AmigaVibes**
Rédacteur **Revival Gamers**

Intéressons-nous à un micro-ordinateur ayant marqué les esprits, que ce soit par son nombre d'unités vendues ou par la longévité de sa scène «underground». C'est par lui que tout a commencé en se démocratisant, ce qui est dans l'état d'esprit du slogan de la marque «Computers for the masses, not for the classes». Certains d'entre vous sourient, et ils ont reconnu le C64.

Nous nous intéresserons plus particulièrement à la *demoscene* et ses productions marquantes dans le temps. Mais retournons au commencement et à l'émergence de ce mouvement caché.

In the Beginning

Revenons un peu plus de trente ans en arrière, dans les années 1982-85, ce sont celles qui ont connu l'émergence de la micro-informatique familiale concomitante à la perte de vitesse des consoles de première génération (**Atari VCS 2600**, **Mattel Intellivision**, **CBS Colecovision** et **Videopac Philips**) et au premier *crack* de l'univers des jeux vidéo. Tout était prêt pour accueillir un micro-ordinateur familial : le **Commodore C64**, qui, à l'instar des consoles citées ci-dessus, permettait de copier les cassettes et/ou les disquettes, ce qui était impossible avec les cartouches des consoles. De plus, le catalogue de jeux et de programmes était exponentiel sur les micro-ordinateurs, et c'est par le biais des amis ou de la famille que l'on augmentait sa collection, les finances étant souvent limitées (argent de poche souvent ou déception des achats après avoir lorgné sur une superbe jaquette).

Enfin les clubs d'informatique se multipliaient en même temps que les «plans informatique pour tous», et c'est de ces clubs que nous allons parler. Car les membres s'échangeaient en leur sein leur collection, mais allaient bien plus loin, en mettant leur connaissance en commun, comme passer du langage **Basic** à l'assembleur sur des machines peu documentées à l'époque. Ainsi, des experts émergèrent et des personnes en demandèrent plus, comme d'ajouter des vies infinies ou de passer au niveau suivant pour un de leur jeu. Les experts devinrent vite des *crackers* qui fournissaient les jeux avec un message sur l'écran titre du jeu «Cracked by XYZ» et ajoutèrent rapidement des *trainers* pour la satisfaction des joueurs. Assez rapidement, ces hackers de génie voulurent démarquer leur production de celles des autres ou même des jeux; ils introduisirent une petite présentation en dehors du jeu avec des effets qui permettaient de différencier les groupes de *crack*. Certains se souviennent sans doute de ***cracktro*** qu'ils regardaient scotché à leur moniteur.

Les versions de jeux furent plus copiées que d'autres à cause de la *cracktro* les accompagnant. Et le premier micro-ordinateur avec une réelle *demoscene*, euh, *cracktroscene*, c'est le **C=64**, grâce à sa très grande popularité, ainsi qu'à l'impressionnant nombre de jeux publiés dessus. Et comme nous l'avons vu, c'est l'attrait des jeux qui a engendré les premiers groupes de *hacking* tels que **G.C.S** (**German Cracking Service**), ou bien les anglais de **Yak Society.** Et c'est grâce à ces mêmes hackers que la scène est née avec leur volonté de toujours se surpasser dans leur *cracktro* ainsi que leur esprit de compétition à lancer la scène qui s'est séparée de la *cracktroscene* et des *copy-party*. Mais revenons un peu sur cette période et les groupes de crack émergeants.

Le retour du JEDI ou les trois «O»

JEDI est un groupe basé en Allemagne en 1983, s'appelant aussi JEDI 2001 (avant l'an 2000, tout ce qui avait un 2000 était à la mode ;-)). C'est un *cracker* ayant pour *nickname* **OTD** qui, avec **Oleander** et **1103** ont formé ce groupe. Mais comment expliquer ce nom de JEDI, tout simplement par la connaissance de ses membres. Ainsi Oliver **J**oppich est le **J**, son pseudonyme est 1103 qui est aussi sa date de naissance. Ainsi l'on remarque une première chose dans ce microcosme du *hacking*, tout à une signification bien précise si l'on sait lire entre les lignes. Après la découverte du J, passons au «**E**» pour Oliver **E**ickemeier aussi connu sous le pseudo Oleander, provenant de son «*Kinderzimmer*». Et le **D**, pour Oliver Thomas **D**ietz (OTD la boucle est bouclée). Et le **I** me direz-vous, c'est pour **I**nc ou le I de Dietz. Les 3 «O» ont oeuvré sur de nombreuses *cracktro* et c'est le premier groupe connu en Allemagne et basé sur le Rhin.

Et c'est après quelques centaines de *cracks*, qu'OTD a programmé des extensions hardware **64er DOS** et **Prologic DOS**, alors que 1103 et Oleander ont produit le **Speed DOS**.

ECA 1998 travaille comme les crackers du groupe JEDI, mais en solo. A coté de ses activités illégales, il a lui aussi des activités légales. Il est surtout connu pour avoir créé le concept des

Les grandes lignes des premières années «crack» sur C64 en quelques groupes

1982-1983 :	Jedi, KBR (**K**otz**br**ocken), Antiram, GCS (German Cracking Service).
1984 :	TBC (c'est le premier groupe à avoir fixé des jeux américains en version PAL). Yak Society (ayant cracké tous les jeux de l'éditeur Elite et premier groupe de crack Anglais) et Teeside Cracking Service.
1985 :	Section 8, Megabyte ou encore le premier groupe Danois : Federation Against Copyright.
...	
	an ellipse in Time
...	
2017 :	certains jeux *homebrew* C64 sont réalisés avec une *cracktro* et un *trainer* en plus pour retrouver l'esprit des années 80.

«*releases*», ce qui veut dire que ses *cracks* sont bien packagés, avec un *trainer* inclus.
La majorité des premiers *crackers* sont dans leur grande majorité des personnes qui travaillent maintenant dans des compagnies d'informatique. Ils ont été recherchés pour leur compétence et leur innovation, ainsi que pour leur expertise dans la connaissance des machines de l'époque. Car ils montraient une telle connaissance des machines que parfois, ils découvraient des capacités non répertoriées et inconnues des ingénieurs à l'origine de ces micro-ordinateurs. Ils étaient ainsi parfois à des niveaux avancés fortement recherchés par les compagnies pour leur talent de programmeur ou tout simplement pour les exploiter dans des jeux. Ainsi, le groupe TBC a réussi à convertir des programmes C64 fonctionnant sur les modèles Américains au format NTSC en des jeux compatibles avec le format PAL de nos contrées.
Et la scène *crack* devient florissante. C'est le moment où la compétition s'installe et les *cracktro* vont devenir de plus en plus impressionnantes. La compétition est également au niveau du groupe qui sera le plus rapide à déplomber les jeux et à les distribuer. Il y a donc des *crackers* ou déplombeurs ;-), il peut y avoir des personnes dédiées aux *trainers*, et d'autres au code des effets d'une *cracktro*, et au moins un musicien et un graphiste pour marquer les esprits des gens. Car cela devient un programme indépendant du jeu qu'il véhicule autant pour la renommée du groupe que sa distribution, qui, parfois, ne se fait plus pour le jeu mais pour la prouesse de la *cracktro*.
Ainsi se termine notre petit tour dans la scene *cracktro* qui montre que les *cracktro* deviennent intéressantes et que certains vont se séparer de ce courant pour en créer un nouveau qui est celui de la *demoscene,* toujours underground, mais qui n'a plus d'activités illégales. Mais ceci sera le sujet d'un prochain article. •
JeFfR3y ^ VM

La Retro **Gaming** Connexion

• Jean-François Richard
Chef de Projet - **Sodifrance**
Co-fondateur **AmigaVibes**
Rédacteur **Revival Gamers**

La RGC (Retro Gaming Connexion) est une convention se déroulant chaque année en Automne dans la salle des fêtes de Meaux. Elle est l'oeuvre de l'asso RGC, association de loi 1901, dont le but est de promouvoir et sauvegarder le patrimoine vidéo-ludique, ainsi que de mettre en relation les utilisateurs de ces machines rétro. Elle a été créée en 2005 par six passionnés de culture vidéo-ludique aux origines diverses (journaliste, animateur multimédia, technicien piscine, etc.).

Dès le début, le groupe a organisé diverses manifestations dans le but de réunir des passionnés de tous horizons. Eh si ! Chaque année, elle réunit beaucoup de français mais pas uniquement : des amis helvètes ou belges, mais aussi quelques Allemands, Américains et Anglais viennent. Mais revenons aux origines.

Back 2 the Roots

Tout commença par la **Jaguar Connexion,** qui a connu cinq éditions et qui était dédiée à la console Jaguar d'Atari. Ces premières conventions ont remporté un vif succès et se sont rapidement imposées comme une référence. Mais comme ces conventions se limitaient à la Jaguar ou aux autres machines de la marque Atari, une évolution logique était d'ouvrir à d'autres consoles ou micro-ordinateurs. Ainsi les activités de l'association se sont élargies et se sont concentrées sur deux conventions majeures : la RGC et l'AC.

AC : the little RGC for Amiga Amstrad Atari ... Connexion

L'AC est une convention, se déroulant chaque année vers le mois d'Avril, qui est plutôt dédiée aux micro-ordinateurs. Elle se déroule dans une petite ville de Seine-et-Marne à **Congis-sur-Thérouanne**. Le rassemblement se déroule durant un week-end dans la salle des fêtes dans une ambiance conviviale et familiale. Le programme de cette manifestation est conséquent, il y a des présentations de projets ou de jeux touchant au retro-gaming, une *speed-coding gaming* avec des lots pour les participants, et, souvent avec à la clé de nouveaux jeux *home-brew* pour nos vieilles machines. Il y a aussi un concours *gaming* et des jeux comme des quizz. Attention, c'est une convention en places limitées, elle n'accepte que les personnes s'étant inscrites au préalable (100-120 places).

RGC : Connexion for Retrogamers

La RGC qui nous intéresse, comme l'AC, est une manifestation qui n'est ouverte qu'aux inscrits ; elle réunit pour un week-end quelques 300 passionnés de consoles et de micro-ordinateurs rétro, c'est à dire allant de la première génération de console 8 bits (l'Atari VCS, la Mattel Intellivision, la ColecoVision de CBS ou la Videopac de Phillips) à des consoles comme la PSP, une machine devenue rétro lorsque sa production s'est arrêtée. Plusieurs concours et quizz sont organisés avec de magnifiques lots à la clé. Un espace musical attire toujours beaucoup de monde, dont les plus jeunes avec même du «Guitar Hero» ou du «Rock Band». Il y a aussi des espaces dédiés aux micro-ordinateurs avec les Amiga et Atari, mais aussi pour les plus anciens comme les MSX, CPC et C64. Et chaque année, un thème principal est choisi, et un pôle avec des jeux sur ce thème est mis en avant sur des micros et consoles. Il y a toujours un pôle dédié à Nintendo ayant pour thème une licence majeure de la marque comme Zelda l'année dernière, ou encore Mario Kart durant une autre édition. Il y a souvent une Lan dédiée à Steel Batallion. Et, cerise sur le gâteau, chaque année un concert a lieu le samedi soir. Enfin, pour aller jusqu'au bout de la passion, une petite bourse d'achats, d'échanges et de ventes a été mise en place et permet souvent de négocier ou de trouver les perles rares pour nos collections…
Cette convention aura lieu cette année le **Samedi 30 Septembre et le Dimanche 01 Octobre 2017**. •

Site de la RGC (en construction)
http://www.retro-gc.fr/
Site de la RGP : http://rgplay.fr/

Création de disquettes sur **Amiga**

• Jean-François Richard
Chef de Projet - **Sodifrance**
Co-fondateur **AmigaVibes**
Rédacteur **Revival Gamers**

Aujourd'hui, nous allons nous pencher sur un mode opératoire qui devrait ravir plus d'un d'entre vous. Alors ne perdons plus une minute et entrons dans le vif du sujet. Tout d'abord, nous allons décrire les pré-requis, car il y en a.

Il vous faut un Amiga 600, 1200 ou 4000 possédant un slot PCMCIA en série. Si vous avez un de ces micro-ordinateurs de la gamme des Commodore Amiga, vous êtes l'heureux possesseur d'un des modèles permettant de brancher dessus un adaptateur de carte Compact Flash au format PCMCIA. Et là, cela va être rapidement le bonheur.

Ensuite, il suffit d'acheter une carte Compact Flash, cela n'est plus très cher, et une capacité de 4 Go est tout à fait acceptable, car n'oublions pas que les supports de l'époque étaient de l'ordre d'une disquette de 880 Ko et que les jeux les plus imposants faisaient de l'ordre de 5 à 10 disquettes (4 à 8 Mo).

Vous aurez aussi besoin d'un lecteur de carte de Compact Flash sur PC, cela nous aidera grandement dans le création d'une partition compatible au format FAT32. Ainsi, la carte est compatible avec un micro-ordinateur équipé de Windows. Après l'avoir formatée au format FAT32 et lui avoir donné un petit nom comme AMIGA1200.

Votre Compact Flash dans un explorateur Windows [1]

Il est alors indispensable de rendre cette carte compatible avec l'Amiga, pour cela il suffit de télécharger la petite librairie suivante FAT95 et l'outil CFD (voir référence en fin d'article).

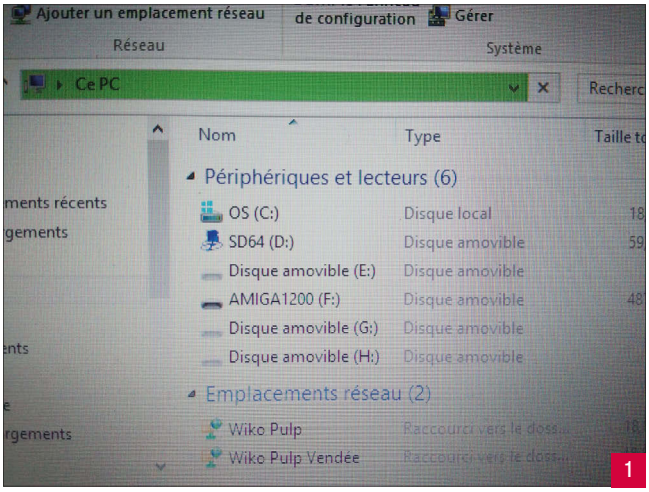
Décompressez cette archive avec un utilitaire comme 7Zip dans la racine de votre Compact Flash.

Alors, vous pouvez alimenter votre carte avec des répertoires, sous-répertoires et des fichiers images de disquettes .ADF (ou .DMS). Enfin, une fois toutes ces opérations effectuées, ils ne vous reste plus qu'à retirer la carte Compact Flash de votre PC et à l'insérer pour la première fois dans votre adaptateur au format PCMCIA. Mettez en route l'Amiga et faites chauffer les turbines en chargeant un Workbench 3.1 de préférence (ou au minimum WB 2.0) par le biais de votre disque dur ou via une disquette de boot. Nous voici dans l'atelier Amiga et votre carte Compact Flash est reconnu.

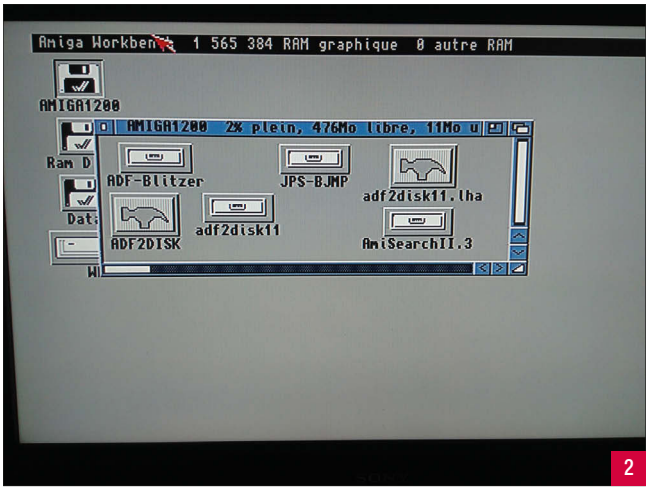
Le Workbench 3.1 et ADF-Blitzer [2]

Insérez une disquette vierge dans votre Amiga, si elle ne l'est pas, formatez-la.

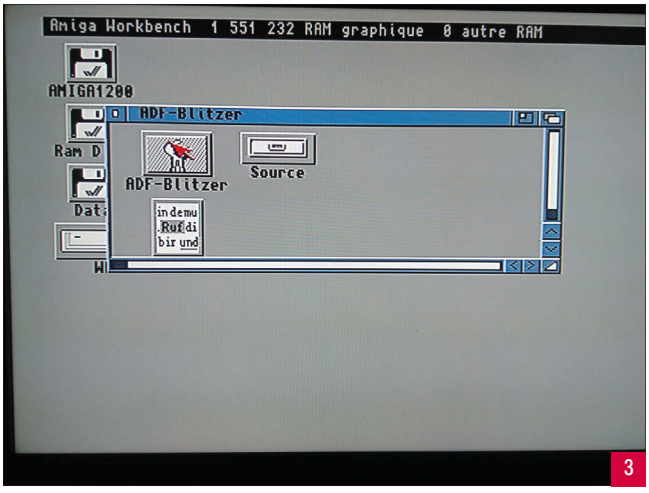
Ouvrez l'arborescence de votre carte en



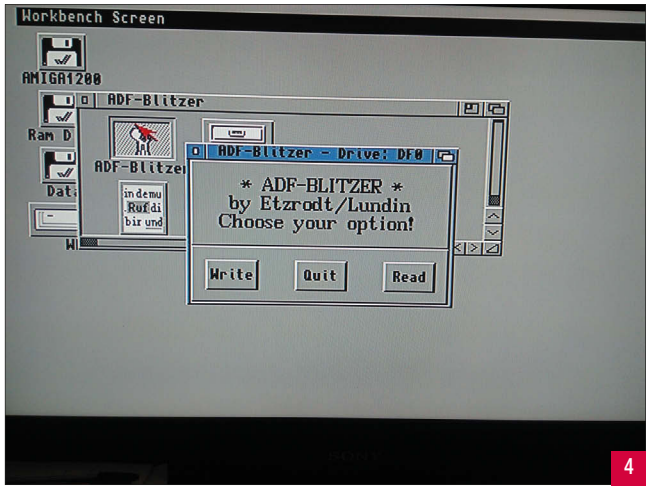
1



2



3



4

double-cliquant dessus.
Ouvrez sur le dossier ADF Blitzer, puis cliquez sur l'icône d'ADF-Blitzer que nous venons d'installer précédemment sous Windows.

Dans le dossier ADF-Blitzer [3]

Cliquez sur le bouton Write

Ecrivez votre disquette [4]

Sélectionner l'image de la disquette au format ADF qui vous intéresse, dans mon cas c'est une image que j'ai réalisée de la mythique trackmo de Scoopex : la Mental Hangover.

Sélectionnez l'image de la scoopex Mental Hangover en ADF [5]

C'est maintenant que la magie va s'opérer avec une copie de votre image sur la disquette physique et si votre lecteur, ainsi que la disquette sont en bonne condition, la copie sera relativement rapide, en une minute environ.

Il va falloir attendre dans les 40 secondes pour la copie [6]


9

La disquette au format Amiga-DOS recréée [7]

Il vous suffit alors d'éteindre votre Amiga via un Soft Reset (Touches Amiga Gauche + Amiga droit + Ctrl) ou par un Hard Reset, le kickstart va booter sur la disquette et c'est parti comme dans les années 80. Vous pourrez ainsi vous reconstituer une collection de disquettes pour les conventions retrogaming ou tout simplement pour votre plus grand plaisir.

L'écran titre de la mythique Mental Hangover [8]

Le logo de SCX : un pentagramme / Generation Ahead [9]

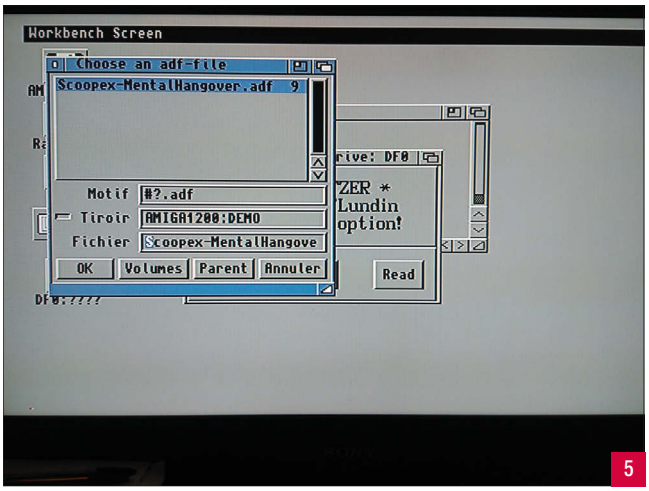
Un autre intérêt de ce que nous venons d'exposer est d'archiver et préserver votre collection, c'est très simple, tout se déroule comme nous venons de le décrire, il vous suffira de créer une image ADF de votre disquette au lieu de l'inverse et cela via le bouton READ. Ainsi l'ensemble de votre collection de disquettes, si elle était conservée dans de bonnes conditions sera virtualisée et stockée sur un support différent de vos disquettes d'origine. •

Adresses :

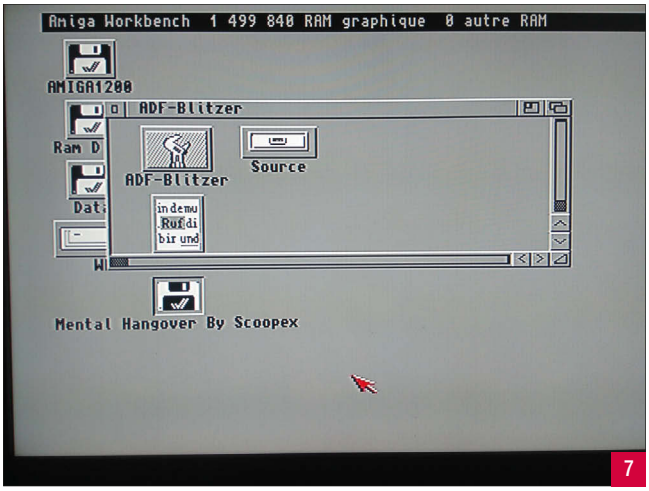
Librairie FAT95 et outil CFD :
http://obligement.free.fr/articles/pcmciacompactflash.php
Application ADF Blitzer :
Sur Obligement (site français) :
http://obligement.free.fr/logiciels/adfblitzer.lha
Sur Aminet (site anglais) :
http://aminet.net/disk/misc/adfblitzer.lha

Les conditions dégradant le bon fonctionnement des disquettes :

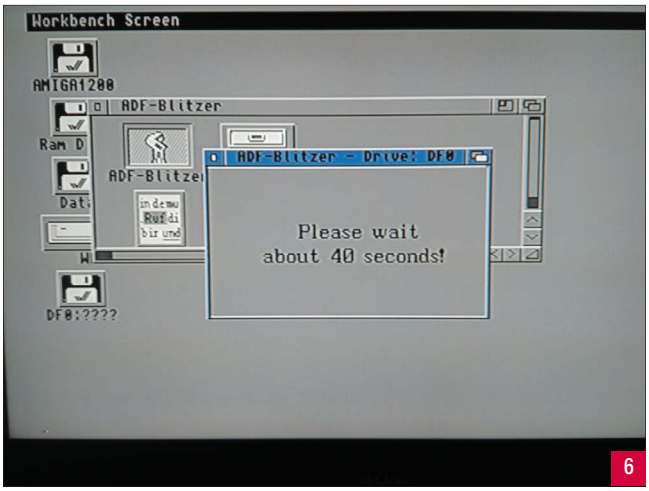
- la poussière ;
- l'humidité ;
- les objets magnétiques (entraînant une démagnétisation du support);
- la lumière.



5



7



6

8

La gamme de l'ATARI ST

Frédéric Sagez
Codeur du groupe
NoExtra-Team
fsagez@gmail.com

Dans l'ère du multimédia et de la micro-informatique de loisirs des années 80, Atari va sortir le grand jeu face à ses concurrents qui sont le Macintosh d'Apple, l'Amiga de Commodore ou l'Amstrad CPC en produisant la nouvelle gamme Atari ST et sortir petit à petit des consoles familiales de jeux.

1

L'Atari ST a soufflé ses 30 années d'existence en 2015 et mérite toujours son surnom de machine « increvable ». Petit retour en arrière…

Un peu d'histoire

Dans les années 80, *Jack Tramiel* qui dirigeait la société Commodore en 1984 décida de la quitter pour créer sa propre société avec sa petite famille. Avec beaucoup de panache, il racheta la société Atari à la Warner et nomma son fils aîné *Sam Tramiel* comme dirigeant. *Jack Tramiel* étant soucieux de trouver un digne successeur des machines à la famille Atari 8 bits à usage familial, et connaissant beaucoup de monde chez les autres concurrents de l'époque, il lancera la gamme ST pour ouvrir la voie des nouveaux ordinateurs composés de microprocesseurs 16 bits. La première version de cet ordinateur personnel sera le 520 ST; il sera présenté pour la première fois au Winter Consumer Electronics Show en janvier 1985. L'unité centrale est sous la forme d'un gros clavier à laquelle est rattaché un lecteur de disquette externe 360ko simple face, un moniteur monochrome et une souris. L'architecture est assez simple; elle est basée sur le processeur 68000 de Motorola cadencé à 8 Mhz, elle dispose d'un bus externe de 16 et 32 bits en interne, d'où la signification « ST » qui veut dire en anglais « Sixteen / Thirty two », et équipée de 512 kilo-octets de mémoire par défaut. L'atout de cette machine est son environnement graphique style « DESKTOP » avec des icônes et des fenêtres appelé GEM (Graphical Environment Manager) qui fut créé par la compagnie Digital Research de *Gary Kildall*. Son système d'exploitation non multitâche s'appelle le TOS et est ironiquement surnommé le *Tramiel Operating System*, ancêtre du CP/M développé par Digital Research dont la première version se charge sur l'ordinateur via une simple disquette. [1]

The power without the price!

Les performances au moindre prix. Le but pour *Jack Tramiel* était de fabriquer une machine suréquipée moins chère que ses concurrents. Et c'est vrai que la machine est bardée de connectiques qui l'ouvrent vers d'autres périphériques. Du côté du standard, une interface série et parallèle pour y connecter une imprimante ou communiquer avec un autre ST par exemple, 2 prises pour y connecter une souris ou des joysticks, ainsi que 2 prises MIDI pour la musique. Pour les connectiques spécifiques à l'Atari nous avons une prise pour le moniteur, une interface pour relier un deuxième lecteur de disquette, un connecteur ACSI pour connecter un disque dur, et, pour finir, le port cartouche utilisé comme une extension hardware. Les capacités graphiques permettent d'utiliser 3 résolutions possibles : Basse résolution (320×200 pixels en 16 couleurs) permettant d'exploiter une palette de 512 couleurs, Moyenne résolution (640×200 pixels en 4 couleurs) et Haute résolution dit « noir et blanc » (640×400 pixels en 2 couleurs soit monochrome), cette dernière résolution ne peut être utilisée que sur un moniteur Atari car cela est dû à sa haute fréquence de rafraichissement de 70Hz. La partie sonore est centrée sur le composant Yamaha YM2149 qui est un générateur de son/bruit programmable (Programmable Sound Generator) sur 3 voies. Petit plus : une disquette est fournie avec des accessoires pour exploiter les périphériques et le Bureau ainsi qu'un Basic et Logo pour débuter en programmation.

La course à l'évolution

Il faut vite faire évoluer la « bête » et la première évolution sera d'unifier le lecteur de disquette à l'unité centrale et d'augmenter les capacités aux disquettes double face soit une capacité de 720ko. On va installer 2 ROMS supplémentaire pour y intégrer le TOS afin de ne plus le charger via une disquette et aussi d'augmenter la mémoire de 512ko à 1024ko pour les jeux et programmes gourmands. C'est donc en 1986 que va sortir le modèle 520 STF avec 512ko de mémoire et le modèle 1040 STF avec 1040ko soit 1Mo de mémoire au total. Mais la partie digitale sonore n'est toujours pas au rendez-vous et la partie graphique laisse à désirer contrairement à son homologue l'Amiga 500 de Commodore. Pour un meilleur confort sonore, des cartes digitaliseurs sonores comme la ST Replay ou la carte son MV16 (vendue avec le jeu BAT d'Ubisoft) peuvent être utilisées via le port cartouche. Pour la partie graphique, cela était plus embêtant car les routines graphiques qui sont intégrées dans le TOS et ne sont pas très optimisées. Pourtant les ingénieurs ont intégré des fonctionnalités spécifiques liées aux capacités graphiques pour développer des jeux, les instructions LINE-A supportent la gestion et l'utilisation de Sprites ou de déplacement de zone graphique mais leur utilisation n'était pas assez performante, ce qui obligea les développeurs à écrire et à optimiser leurs propres routines graphiques. A noter qu'un logiciel sera très utilisé à cette époque ; TURBO ST, de la société SoftTrek est un programme qui se chargeait en mémoire et qui améliorait sensiblement l'affichage graphique. Et il faudra attendre la sortie de la gamme MEGA ST en 1987 dédiée aux professionnels (Traitement de texte, PAO et MAO) pour avoir les capacités graphiques attendues. Les « boîtes à pizza » comme était surnommées les unités centrales avec le clavier détaché existaient en 3 versions : MEGA 1 pour 1Mo, MEGA 2 pour 2Mo ou MEGA 4 pour 4 Mo de mémoire en interne. Et

2 *Atari STF Versus Atari STE*

bien sûr, ils sont équipés du composant graphique dénommé le Blitter, déjà utilisé sur l'Amiga mais pas avec les mêmes caractéristiques. Rajouter un composant dédié à 100% pour le graphisme et qui ne sollicite pas du tout le 68000 a été une grande évolution pour la gamme du ST, mais malheureusement elle fut très tardive, et les acteurs des logiciels et du jeu ont demandé à ce que l'on puisse avoir la possibilité de désactiver le Blitter pour pouvoir utiliser les anciennes, mais optimisées, routines graphiques. C'est un drame car très peu de logiciels ou de jeux utiliseront réellement les vraies capacités graphiques du ST pour raison de compatibilité.

E pour Enhancement

En 1989 sort l'ultime machine : l'Atari STE avec un « E » comme enhancement comme pour indiquer de grosses améliorations du côté du hardware [2]. Le Blitter est intégré d'office et on étend la palette graphique à 4096 couleurs ce qui permettra d'exploiter les nuances de couleurs. Cette nouvelle série est caractérisée par son système d'exploitation dénommé « Rainbow TOS », faisant référence à un arc-en-ciel qui corrigera pas mal de bugs avec une meilleur détection des périphériques externes. On peut upgrader sa mémoire comme on le souhaite en ajoutant des barrettes de type SIMM comme pour les PC; vous pouviez donc avoir un Atari 520 STE gonflé avec 4Mo de mémoire en interne. Pour les jeux multi-joueurs on rajoutera deux ports de joystick supplémentaires. Et enfin, on s'occupe de la partie sonore en rajoutant un composant audio DMA qui a la capacité de lire des échantillons sonores 8 bits jusqu'à 50 KHz en mono ou stéréo ! A noter que l'Atari ST peut lire des disquettes provenant d'un PC sans problème et à partir du STE, le formatage sera compatible à 100% vers les PC depuis cette version.

Plat de résistance

La gamme des ST à base de processeur 68000 de Motorola va se terminer en apothéose en 1990 avec la sortie du MEGA STE. Toujours au format « boîte à pizza », cette unité centrale survitaminée tourne à 16 MHz, utilise 16Ko cache et est équipée d'un FPU 68881 de Motorola plus 2 Mo de mémoire par défaut toujours au format SIMM. Au niveau des extensions, un port d'extension VME situé à l'intérieur du boîtier, un port réseau et surtout un disque dur interne. Le TOS est enfin arrivé à maturité (à la septième version au final !) avec une meilleure gestion du bureau « DESKTOP » et surtout une meilleure visibilité sur la capacité mémoire. [3] Ah ! J'allais oublier, il y a eu deux versions « portables » du ST sous le nom de STACY ou encore ST BOOK mais la piètre capacité des batteries de l'époque, de sa réelle utilité et le prix en ont découragé plus d'un à l'achat. Pourtant ils sont devenus des pièces de musée aujourd'hui !

4 *Le « Jackintosh » à gauche en action !*

3 *Le ST Book d'Atari*

On faisait quoi de plus avec un ST ?

Les plus grosses ventes d'Atari ST ont été effectuées principalement en Europe dans les années 80. Je ne vais pas revenir sur la large bibliothèque de jeux et de logiciels qui étaient disponible à l'époque sur la machine mais il faut se rappeler qu'en France, équipé d'un Minitel, on pouvait se connecter sur des BALs pour communiquer par message ou télécharger des programmes via des logiciels comme Sapristi bien avant l'ère d'Internet. Et puis on ne parle pas du ST sans parler des émulateurs qui ont tourné dessus à l'époque comme le plus célèbre d'entre eux : Aladin qui était l'émulateur Macintosh d'Apple, incroyable de pouvoir utiliser le Finder sur un ST [4]. Emuler le DOS sur un ST et utiliser des disquettes MS DOS, rien de plus facile avec l'émulateur PC DITTO d' Avant-Garde Systems et vous laisser à l'aventure pour installer un Windows 3.1 de Microsoft ou tout simplement utiliser un environnement multitâche programmable avec le Bootstrap OS9/68k. Mais n'oublions pas l'argument premier de vente de la machine : les deux prises MIDI en standard. Brancher son synthétiseur sur l'ordinateur nous permettait à l'époque de découvrir la musique à travers des outils comme Cubase et rejouer les morceaux de « Rendez-vous » de Jean-Michel Jarre.

Le coin des bidouilleurs

Les magazines parlant de l'actualité de l'Atari n'étaient pas légion à cette époque et Atari Magazine et ST Magazine étaient des mensuels reconnus pour leurs parties éditoriales. Mais je confesse que la partie la plus intéressante était le coin de la « Bidouille »; dans cette partie on

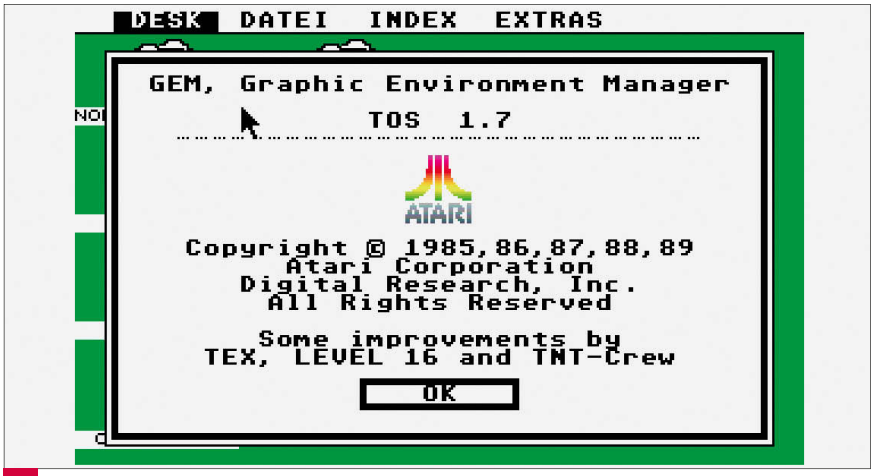
vous expliquait comment upgrader – avec de bonnes connaissances en électronique – votre bécane avec un bon fer à souder et quelques résistances. Les articles étaient écrits par des experts tel que Rodolphe Czuba et vous indiquaient par exemple comment ajouter de la mémoire sur son ST avec des composants 41256, mettre un lecteur de disquette HD, accélérer son microprocesseur ou tout simplement ajouter la puce du Blitter dans son Atari STF. [6]

La scène démo

Non content d'avoir cassé les standards Atari en ayant enlevé les borders haut, bas, droit et gauche de l'écran et affiché des images Spectrum en milliers de couleurs (soit quand même 48 couleurs différentes par ligne en une seule frame), les gens de la démoscène relevaient constamment des défis sur cette machine pour faire les meilleures démonstrations techniques. Pour rappel toutes les limites outrepassées se font au niveau software avec du bon vieux code assembleur. Des groupes de démos Allemand très connus à l'époque ont même été plus loin en réécrivant complètement une version du TOS amélioré pour leurs propres développements. [5]

Et aujourd'hui ?

La communauté est toujours active et très vivace autour de l'Atari ST malgré toutes ses années. Les émulateurs ont pris le pas depuis PacifiST de *Frédéric Gidouin* et nous permettent de nous régaler en nous rappelant de bons



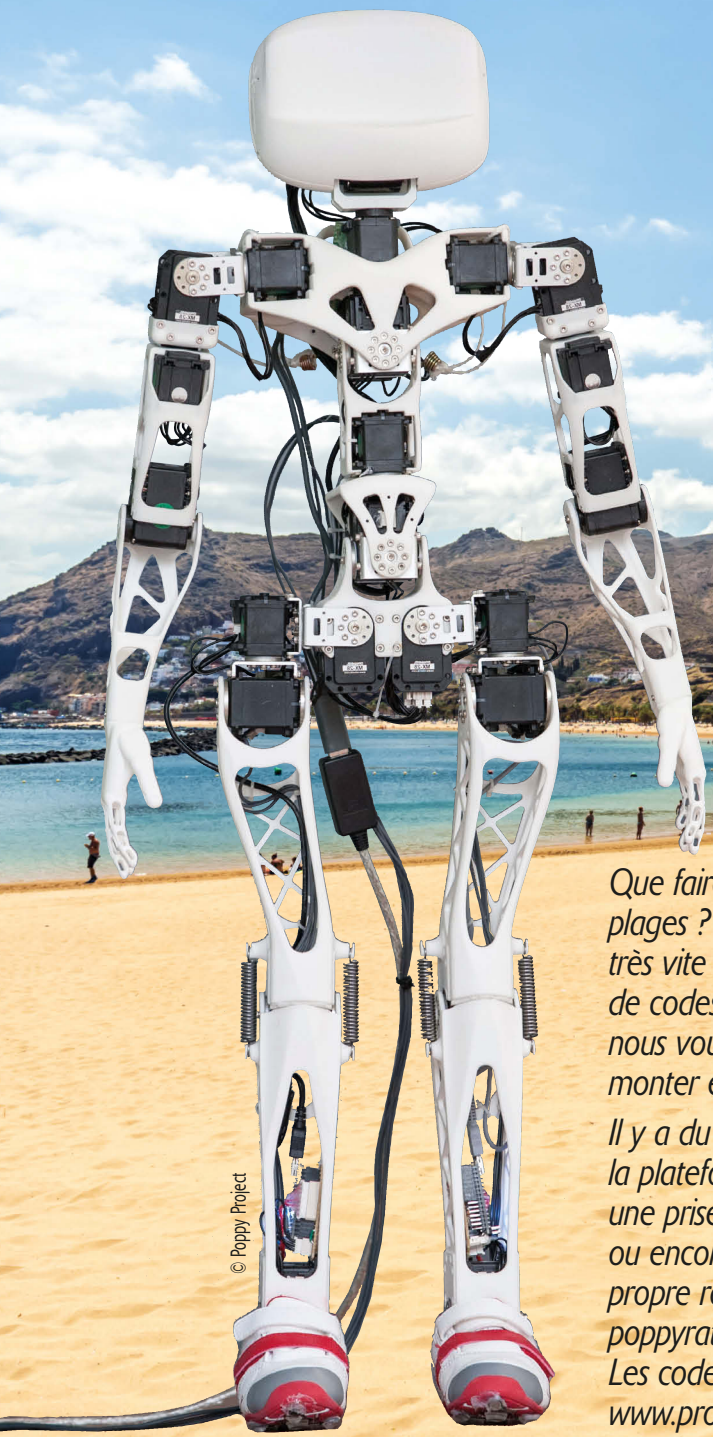
5 *Le Rainbow TOS du groupe The Exception (TEX)*


BIDOUILLES
ATARI STEREO
Musicalement parlant, le ST n'est ni muet, ni manchot. Mais il est inévitable de reconnaître que la plus grande part de sa force sonore revient à l'interface MIDI qui réjouit le cœur de plus d'un musicien synthétivore. Au fait, LE SON NE PASSE PAS PAR LES CABLES MIDI !

6 *Article d'Ange Larriche – Atari 1ST n°10*

vieux souvenirs. La scène démo n'est pas en reste et nous sort des démos de plus en plus éblouissantes avec les technologies d'aujourd'hui comme le CosmosEx de *Miroslav NOHAJ* qui amène le ST à se connecter sur Internet, à utiliser des prises USB et faire office de disque dur avec un accès beaucoup plus rapide. Bien sûr il existe des irréductibles comme moi qui utilisent toujours le Hardware d'origine avec ses bonnes vieilles disquettes. Parfois se replonger dans le passé en ressortant son bon vieux ST de la cave de ses parents peut s'avérer une aventure surtout si la bécane ne démarre plus. Un conseil, il existe des forums ou des experts qui vous répondent sur tout type de problème, n'hésitez pas à consulter le site de Yaronet à cette adresse http://www.yaronet.com/ , et de visiter le forum Atari. •

Mes projets à réaliser cet été



© Poppy Project

Que faire cet été ? Des randonnées ? Un peu de plages ? Lire ? C'est très bien, mais un geek aura très vite envie de toucher un peu de technologies, de codes. Pour ne pas perdre sa programmation, nous vous proposons quelques projets et IoT à monter et à coder durant l'été.

Il y a du choix : un miroir connecté orienté fitness, la plateforme Jeedom (dédiée domotique), créer une prise intelligence avec une Raspberry Pi Zero ou encore monter et programmer son propre robot avec le superbe projet poppyrate.
Les codes sources des projets sont sur www.programmez.com

Enjoy !

La rédaction.

Jeedom : la solution domotique Open Source

Gaëtan Cottrez
Chef de projets chez
Orditech SA
(Tournai, Belgique)
www.deviotyourself.com

Passionné d'informatique dès le plus jeune âge, c'est le développement, plus précisément le web, qui m'a toujours attiré. Analyser un besoin réel pour le développer soi-même dans le but de faire gagner du temps et d'automatiser est quelque chose de très excitant et passionnant pour moi.

Depuis quelques années, la domotique se démocratise de plus en plus dans nos foyers et il y en a pour tous les goûts, que l'on soit un « maker » ou non. Pour centraliser sa domotique, on entend souvent parler de box domotique qui ont chacune leurs avantages/inconvénients. Moi je vous propose d'en découvrir une qui se nomme « Jeedom ».

Jeedom (https://www.jeedom.com/site/fr/) est une solution domotique Open Source en technologie Web (PHP/MySQL/HTML/CSS/JS/NodeJS) créée par 2 français : Loïc et Mathieu. J'insiste sur le fait que c'est bien au départ une solution domotique et non pas une box domotique puisqu'elle peut être installée sur beaucoup d'appareils comme un NAS Synology ou une machine virtuelle mais l'appareil le plus utilisé reste le Raspberry Pi.

Les créateurs ont d'ailleurs créé une image basée sur Raspbian pour faciliter le déploiement de la solution. Plus tard, ils ont créé et commercialisé une box domotique avec Jeedom préinstallé pour la rendre « Plug & Play » à la demande de la communauté n'étant pas très à l'aise avec la partie installation et déploiement. Mais pour vous qui lisez le magazine Programmez! ça ne vous posera pas de problème de la déployer par vous-même j'en suis certain !

Jeedom possède des avantages qu'il n'a rien à envier à ses concurrents :

- Il est totalement autonome et ne nécessite aucun accès à des serveurs extérieurs. Hé oui ! Pas de Cloud ! ;
- Il est « multiprotocole » (Z-Wave, RFXcom, RTS SOMFY, EnOcean, xPL, etc.) ;
- Il est totalement personnalisable grâce à son système de plugins, de widgets, de vues et de design ;
- Il possède un market permettant d'obtenir de nouvelles fonctionnalités (et de nouveaux protocoles) ;
- Il est constitué d'un « core » offrant un panel de fonctionnalités qui permet à l'utilisateur de se l'approprier et de le moduler à sa guise.

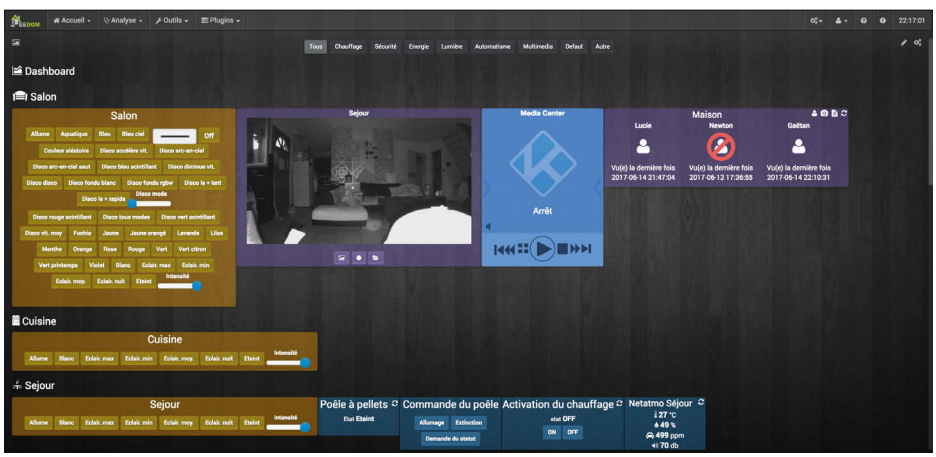
Passons en revue ses principales fonctionnalités.

Le dashboard

C'est la page sur laquelle l'utilisateur arrive lorsqu'il s'est connecté. On y retrouve toutes les données et actions que vous souhaitez afficher. Il faut le voir comme le panneau de contrôle de votre domotique.

Les vues

Ce sont des affichages plus simplifiés que le dashboard. Cela peut être utile lorsque vous possédez une grosse installation domotique et que vous souhaitez un affichage clair et concis.



Les designs

Bien que n'ayant jamais utilisé cette fonctionnalité (je l'avoue), les designs sont très utiles et permettent d'obtenir une représentation graphique en 3D de votre système domotique.

Les objets

Les objets permettent de regrouper un ensemble de données et d'actions pour la cohérence de votre domotique. Pour vous donner un exemple concret, j'ai créé un objet pour chacune de mes pièces : salon, cuisine, chambre, salle de bain, bureau, etc. Cela permet d'obtenir un affichage plus clair sur le regroupement de son équipement sur le dashboard.

Les plugins

Il est possible de rajouter de nombreuses fonctionnalités constituer sous forme de plugins. Jeedom étant de plus en plus connu, il n'est pas rare de voir de nouveaux plugins débarquer sur le market pour les nouveaux appareils et objets connectés populaires qui sortent sur le marché. La plupart du temps ces sociétés qui produisent le plugin car elles sont bien souvent renseignées sur l'engouement de Jeedom mais si ce n'est pas le cas vous pouvez être sûr que la communauté le produira très rapidement.

Les logs

Jeedom possède un système de logs simple mais efficace; tout ce qui concerne le « core » de Jeedom est loggé et il en va de même pour les plugins.

API

Jeedom possède comme toute bonne application qui se respecte une API riche et puissante. En clair tout ce qui est possible de faire par l'interface Web de Jeedom peut-être fait par l'API. C'est très utile pour interagir avec d'autres sys-

tèmes domotiques ou pour tout simplement envoyer/recevoir des informations avec des systèmes externes à la solution.

Les interactions

Les interactions permettent d'ajouter un contrôle supplémentaire à votre maison : le contrôle vocal. En effet, il est possible de créer des interactions personnalisées pour lancer des commandes. Nous pouvons par exemple configurer la phrase suivante « Allumer la lumière du salon » et l'attacher à l'action qui allume la lumière du salon. Ainsi avec son smartphone, il est tout a fait possible de faire de la reconnaissance vocale pour lancer votre instruction directement sur l'API Jeedom, elle-même interprètera votre interaction.

Les scénarios

C'est la partie la plus importante du Jeedom car elle constitue le cerveau de votre maison et l'intérêt d'avoir un système domotique. Les scénarios permettent d'interagir avec le monde réel de manière « intelligente ». Leurs constitutions reposent sur un système de bloc. Un bloc peut contenir une ou plusieurs instructions. Les instructions peuvent être :

- Des actions (lancer telle commande) ou faire une pause dans le script ;
- Des instructions basiques que l'on retrouve dans la plupart des langages de programmation (boucle, si/alors/sinon, commentaires) ;
- Insérer du code spécifique comme du PHP ou du Shell ;
- Définition de variables.

La constitution d'un scénario permet de faire communiquer tout votre petit monde domotique afin d'automatiser des actions et de déclencher des conséquences dans le monde réel. Vous pouvez par exemple demander à votre système domotique de tamiser votre lumière du salon lorsque vous lancez un film sur votre média center (www.maison-et-domotique.com/69724-scenario-jeedom-controlez-ampoules-connectees-mi-light-avec-kodi/).

L'historisation

Il est possible d'historiser quasiment toutes les actions et commandes qui ont obtenu des changements de valeurs. Cela peut être très pratique pour effectuer des courbes pour la température sur une période choisie par exemple.

Le market

Le market permet de faire évoluer votre solution suivant vos besoins. C'est par le market que vous installez les plugins. Une grande majorité des plugins est gratuite et il en existe des payants. Un plugin payant coûte entre 2 et 6 euros en moyenne. Si vous souhaitez réaliser votre propre plugin sachez que Jeedom possède une documentation développeur (https://jeedom.github.io/documentation/phpdoc/index.html) et un plugin nommé « Template » (https://jeedom.com/doc_old/documentation/plugins/template/fr_FR/template) qui sert de base de travail pour la réalisation de votre plugin. Enfin, la communauté est très active sur le forum (https://www.jeedom.com/forum/) et se fera un plaisir de vous accueillir, vous et votre plugin, afin de vous aider.

Et si vous espionniez votre facteur pendant que vous êtes en vacances ?

Je vous propose ce cas concret à reproduire chez vous pour cet été : domotiser sa boîte aux lettres. Le but est de savoir si vous avez reçu du courrier ou non dans la journée en surveillant les 2 ouvertures qui peuvent être utilisées par le facteur à savoir la porte avant pour les gros colis et le volet pour le dépôt du courrier.

Pour cela vous allez avoir besoin :

- Jeedom déjà configuré avec :
  - Le plugin « Arduidom » ;
  - Le plugin « Pushbullet », pour envoyer une notification sur son smartphone que le facteur est passé ;
- 2 détecteurs d'ouverture de porte émettant en radiofréquence 433 Mhz pour la boîte aux lettres ;
- Un récepteur 433 Mhz DIY pour recevoir les émissions de la boîte aux lettres et les transmettre à Jeedom.

Le fonctionnement est facile à comprendre. Lorsque l'un des détecteurs d'ouverture de porte se déclenche, il envoie un code en 433 Mhz. Notre récepteur capte l'information et la transmet à notre plugin « Arduidom » qui l'interprète comme une valeur connue dans notre système. Il faut savoir que le 433 Mhz est un protocole libre utilisé par pas mal d'équipements comme les portes de garage par exemple. Il est donc judicieux de ne se préoccuper que des codes qui nous intéressent. Si « Arduidom » détecte que le code est celui de notre boîte aux lettres, il mettra à jour une valeur qui va déclencher automatiquement un scénario qui va envoyer une notification sur votre smartphone pour vous dire « le facteur est passé ! ».

Si cela vous tente de reproduire ce cas de figure, vous pouvez retrouver tous les détails sur ce lien http://www.maison-et-domotique.com/72247-projet-arduino-domotiser-boite-aux-lettres-connectee/.

Partez l'esprit tranquille avec **Jeedom** !

Christopher Polonio
Lead Developer chez
Orditech SA
(Tournai, Belgique)
www.espritorienteobjet.com
Passionné par les nouvelles technologies qui gravitent autour du Web, du mobile et de l'IoT, j'ai la chance d'exercer un métier que j'adore. Il me faut toujours plus de challenge ! Ma philosophie est d'apporter, quel que soit le degré de complexité d'un projet, des solutions pratiques et ergonomiques aux utilisateurs finaux.

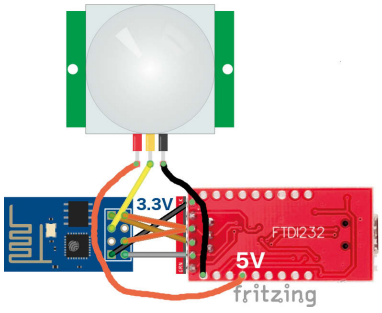
Vous partez en vacances ou en week-end cet été ? Vous n'avez cependant pas beaucoup d'argent à investir dans un système d'alarme complet ou une caméra IP. Je vous propose de concevoir rapidement un petit capteur de présence (PIR) connecté en Wi-Fi à Jeedom. Le système vous alertera instantanément par notification sur votre smartphone de la présence d'une personne. Vous pourrez ainsi prendre les dispositions nécessaires à distance !

Le matériel

- 1 FTDI FT232RL breakout (3.3V / 5V) + câble mini USB + adaptateur secteur USB (2A) ;
- 1 ESP8266-01 (ESP-01) ;
- 1 capteur PIR ;
- Câbles de prototypage mâle-mâle, femelle-femelle et mâle-femelle pour breadboard.

Le schéma de montage

- Le modèle FTDI FT232RL est un circuit intégré permettant de convertir aux normes USB des appareils communiquant en série. Nous l'utiliserons ici pour 2 raisons; la première pour téléverser facilement le code dans notre ESP-01, et la seconde pour alimenter ce dernier ainsi que notre capteur PIR. Ce modèle délivre du 3,3V et du 5V.



- L'ESP8266-01 est un microcontrôleur disposant d'un module Wi-Fi et de 2 pins GPIO nous permettant de recevoir ou d'envoyer un signal numérique, fonctionnant en 3,3V.
- Le capteur PIR (Passive InfraRed) détermine si un être humain ou un animal est entré ou sorti du champ de détection du module. Il fonctionne en 5V.

Le code embarqué

L'ESP-01 fonctionne par défaut avec des commandes AT (serial modem mode). Ces commandes sont verbeuses et ne permettent pas une grande souplesse d'utilisation. Dès lors, grâce à nodemcu-pyflasher (https://github.com/marcelstoer/nodemcu-pyflasher), nous flashons notre ESP-01 avec le Firmware NodeMCU (http://nodemcu.com/index_en.html). Il a l'avantage d'être open-source et de fournir un kit de développement qui aide à la confection de scripts Lua, un langage proche du Python.

Pour flasher avec le Firmware NodeMCU, il est important de se baser sur une image stable et adaptée à l'ESP-01 (la plus légère possible). Pour ce faire, vous pouvez personnaliser votre build (https://nodemcu-build.com/index.php) en ne sélectionnant que les modules nécessaires. Dans notre cas, nous sélectionnons ceux proposés par défaut et ajoutons le module *HTTP*. Grâce au service en ligne, vous recevez par e-mail 2 images (*integer* et *float*). Choisissez la version *float* qui gère tout simplement les valeurs flottantes dans les variables ou calculs que vous utiliserez.

Le branchement adéquat pour le flash se fait en déconnectant le PIR de l'ESP-01 et du FTDI et en reliant la broche GPIO0 à la masse. Une fois votre prototype branché en USB à votre ordinateur, dans nodemcu-pyflasher, choisissez le baud rate 115200, le port série détecté par votre machine et le mode DIO, préféré si l'on veut faire du rendu de pages Web. Après l'opération, débranchez le module et rebranchez-le comme initialement, avec le capteur.

Afin de téléverser le code Lua que nous allons produire dans l'ESP-01 fraîchement flashé avec le Firmware NodeMCU, nous devons utiliser ESPlorer (https://github.com/4refr0nt/ESPlorer). C'est un IDE conçu en Java permettant à la fois de développer en Lua et Python mais aussi de communiquer de façon bi-directionnelle avec un ESP8266. Sélectionnez dans ESPlorer le bon port de communication. Il vous faut créer un fichier nommé *init.lua* avec le code qui suit, adapté à votre configuration réseau et à celle de Jeedom pour ensuite le sauver et l'envoyer vers l'ESP.

```
ssid = "#votre_ssid#"
password = "#votre_mot_de_passe#"
cfg = {ip="#ip_esp#",netmask="#masque_sous_reseau#",gateway="#ip_
passerelle#"}
pir = 3
gpio.mode(pir, gpio.INPUT)

wifi.setmode(wifi.STATION)
wifi.sta.config(ssid,password)
wifi.sta.setip(cfg)

print(wifi.sta.getip())

if not tmr.create():alarm(2000, tmr.ALARM_AUTO, function()
  state = gpio.read(pir)
  http.get("http://#ip_jeedom#/core/api/jeeApi.php?apikey=#cle_api_jeedom
#&type=virtual&id=#id_commande_virtuelle#&value="..state, nil, function(code, data)
    if (code < 0) then
      print("HTTP request failed")
    else
      print(code, data)
    end
  end)
end)
then
  print("timer")
end

if srv~=nil then
```

```
    srv:close()
end
srv=net.createServer(net.TCP)
srv:listen(80,function(conn)
    conn:on("receive", function(client,request)
        local buf = ""
      local _, _, method, path, vars = string.find(request, "([A-Z]+) (.+)?(.+) HTTP")
        if(method == nil)then
            _, _, method, path = string.find(request, "([A-Z]+) (.+) HTTP")
        end
        local _GET = {}
        if (vars ~= nil)then
            for k, v in string.gmatch(vars, "(%w+)=(%w+)&*") do
                _GET[k] = v
            end
        end
        if(state == 1)then
            print("motion detected")
        elseif(state == 0)then
            print("nothing")
        end
        print(state)
        buf = "<?xml version='1.0' encoding='UTF-8'?>"
            buf = buf.."<data>"
            buf = buf..state
            buf = buf.."</data>"
        client:send(buf)
        client:close()
        collectgarbage()
    end)
end)
```

En résumé, que fait ce code ?
Premièrement, il sert à configurer les paramètres IP de la carte Wi-Fi intégrée et à instancier un serveur Web écoutant sur le port 80. Ensuite, il permet de récupérer l'état haut ou bas envoyé par le capteur PIR sur le GPIO0 de l'ESP toutes les 2 secondes, de traiter cette valeur reçue et de l'envoyer à l'API Jeedom en utilisant la méthode GET afin qu'elle puisse être stockée dans une commande virtuelle qui provoquera un scénario. Enfin, il constitue un rendu XML de la donnée pour que l'on puisse contacter l'ESP et obtenir la valeur de "mouvement" à la demande.

Test de fonctionnement

Pour tester votre module, il vous suffit d'accéder à son adresse IP via un navigateur. Vous devriez obtenir une page XML constituée d'une balise *data* et contenant la valeur 1 ou 0 correspondant à la détection ou non de mouvement. Faites le test en rechargeant votre page plusieurs fois. Vous pouvez également observer la console dans ESPlorer.

Ajout du plugin Virtuel dans Jeedom

Ce plugin permet la création de périphériques virtuels (dans notre cas le PIR) et de leurs propriétés. Une fois téléchargé depuis le Market, ce plugin se retrouve dans la catégorie *Programmation*. Il vous suffit d'ajouter un équipement virtuel, lui donner un nom, un parent, l'affecter à une catégorie, l'activer, le rendre visible et d'ajouter une commande virtuelle associée de type numérique que vous nommerez *Mouvement* par exemple. Cette commande nouvellement créée possède un identifiant que vous devez renseigner dans l'URL d'appel à l'API Jeedom (Cfr. Code embarqué).

Utilisation des services Pushbullet

Pushbullet (https://www.pushbullet.com) est un service gratuit de communication entre différents appareils. Il est multiplateforme et s'utilise aussi bien sur mobile/tablette que sur ordinateur. Jeedom l'intègre en tant que plugin. Il est dès lors facile de notifier plusieurs personnes en même temps. On peut également l'utiliser comme un espace de chat et de partage de médias. Le paramétrage du plugin est simple. Vous créez un équipement et renseignez le token de votre compte Pushbullet. Après l'avoir sauvegardé, Jeedom va récupérer tous les devices que vous avez déjà ajoutés sur la plateforme.
NB : Ici, nous fonctionnerons de Jeedom vers Pushbullet mais l'inverse est possible aussi; écrire à l'équipement Jeedom afin qu'il réalise une tâche spécifique.

Création du scénario

Dans la partie *Outils* de Jeedom, vous allez créer un nouveau scénario de type avancé. Donnez-lui un nom et un objet parent. Choisissez comme mode de scénario *Provoqué* et ajoutez comme déclencheur la commande virtuelle créée ici plus haut. Ainsi, lorsque Jeedom détectera que la valeur de la commande a changé, le scénario se déclenchera automatiquement. La logique de ce scénario est très simple. On teste si la valeur de la commande est égale à 1 (état haut = détection de mouvement); dans ce cas, on envoie un push message via Pushbullet en appelant la commande de l'équipement Jeedom correspondant. Ensuite, on temporise 30 secondes afin que le scénario ne se rejoue pas si le capteur détecte un autre mouvement juste après le premier. Vous pouvez rebrancher votre prototype et tester une fois le tout sauvegardé.
NB : Les scénarios ne se lancent donc pas en parallèle par défaut.

Fonctionnement du plugin Script

Cet add-on peut exécuter des scripts (shell, php, ruby…), des requêtes http ou encore récupérer des informations dans du XML ou JSON. Il peut nous être utile pour contacter notre mini serveur Web embarqué dans l'ESP-01 afin d'extraire la donnée renvoyée par le détecteur de mouvement infrarouge à la demande.
Aussi dans la catégorie *Programmation,* ajoutez un équipement script, donnez-lui un nom, un parent, affectez-le éventuellement à une catégorie et ajoutez une commande script de type XML. Sur cette ligne, le champ *option* possède un libellé *URL du fichier XML*. Celui-ci contient le lien vers la machine hébergeant le fichier XML en question (dans notre cas l'IP de notre module ESP-01). Il ne vous reste qu'à sauvegarder l'équipement et vous pouvez l'utiliser dans la partie *Scénarios*.
NB : La solution du composant virtuel est préférée dans notre cas puisque le module envoie la donnée captée par le PIR à Jeedom en totale autonomie. La logique de questionner l'ESP-01 depuis le système centralisé n'est pas optimale car cela sollicite bien plus le service exécutant les scénarios. De plus, une tâche Cron est limitée à 1 minute; on ne peut donc pas être averti en "temps réel".

CONCLUSION

Si vous êtes "maker" dans l'âme, ce petit projet à faible coût peut rapidement être mis en place sous forme de prototype. Par la suite, vous pourrez concevoir une board en soudant les composants proprement et la conditionner dans un support destiné à être branché directement sur une prise murale. Vous pouvez également améliorer le scénario pour que, par exemple, vos lumières connectées s'allument ou clignotent automatiquement lors d'une détection de présence. •

Créer une bande de **LED connectée** avec Constellation pour un éclairage intelligent

• Sébastien Warin
Creative Technologist
http://sebastien.warin.fr
http://myconstellation.io

Découvrons dans cet article comment créer une bande de LED connectée en Wifi à votre Constellation. Le but est de gérer l'éclairage de votre pièce de manière automatique, en fonction de la présence et de la luminosité, de créer des effets lumineux pour vos soirées, ou même de vous avertir de certaines situations ; par exemple lorsque l'on sonne à votre porte, que votre site Web ne répond plus, ou qu'une fenêtre est restée ouverte alors qu'il commence à pleuvoir et bien plus encore selon votre imagination …

Nous avons une cuisine avec un long plan de travail d'environ 5 mètres en bois massif au-dessus duquel sont fixées quatre réglettes lumineuses, chacune composée de trois ampoules halogènes. Cet éclairage indirect qui est réfléchi sur le bois du plan de travail donne une ambiance très chaleureuse. Seulement avec le temps les ampoules claquent les unes après les autres et c'est ainsi que l'an passé, je me suis retrouvé avec à peine 2 ampoules fonctionnelles sur les 12 initiales ; forcement ça éclaire moins bien !

Je suis donc allé dans une grande surface pour acheter de nouvelles ampoules mais à ma grande surprise elles sont très chères : plus de 10 euros pour quatre ampoules soit un achat total de trente euros !! De plus comme il s'agit avant tout d'un éclairage d'ambiance, il est allumé dès que nous sommes dans la pièce de vie (cuisine ouverte) avec une luminosité inférieure à un certain seuil, soit en moyenne 6 heures par jour en hiver, 2 heures en été. Avec des ampoules de 10W, il faut donc dépenser entre 10 et 15 euros d'électricité par an rien que pour cet éclairage.

En réalisant cela j'ai reposé les ampoules dans le rayon et je suis reparti chez moi les mains vides mais avec une nouvelle idée en tête : remplacer cela par un éclairage LED moins couteux à l'achat et à l'usage.

Une bande LED connectée

Mes éclairages halogènes étaient jusqu'à présent pilotés par un micro-module Z-Wave caché dans le boitier d'encastrement de l'interrupteur (un FGD-211 pour être précis) connecté donc en Z-Wave à une box Vera, elle-même connectée à la plateforme Constellation.

La première solution aurait pu consister à remplacer ces réglettes par une simple bande LED mono ou multi-couleurs avec transformateur 220V branché sur ce même micromodule Z-Wave. Comptez environ 10 euros pour un bandeau de 5 mètres avec son transformateur. Le pilotage serait resté en Z-Wave via la Vera. Seulement j'étais séduit par la possibilité de piloter également l'intensité et la couleur des LEDs ou mieux de chaque LED individuellement !

Une autre solution dans cette optique aurait été d'utiliser les bandes de LED de la gamme Hue de Philips. Il aurait fallu retirer le micromodule Z-Wave pour alimenter en continue la bande de LED elle-même connectée par ZigBee au bridge Philips dont je dispose déjà pour mes ampoules du salon. Avec le connecteur Philips Hue pour Constellation, le pilotage de cette bande de LED sera très simple seulement cette LED Strip Philips est très onéreuse : comptez environ 80 euros pour seulement 2 mètres !

C'est pourquoi la solution que j'ai retenue est de le faire soit même (DIY) en utilisant des bandes de LED type WS2801 composées d'environ 32 LED RGB par mètre, toutes pilotables individuellement. Pour les lecteurs de ce magazine, j'avais déjà évoqué ce type de LED en Mai 2013 dans mon article sur S-Light, mon système Ambilight home-made.

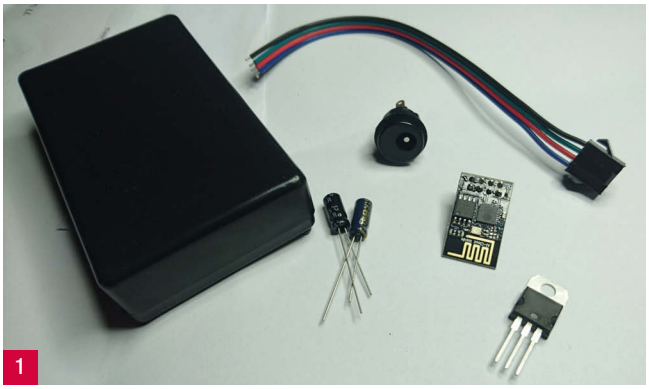


Figure 1 – Le matériel

Comptez environ 30 euros pour une bande de 5 mètres (soit 160 LEDs) que l'on va piloter avec un microcontrôleur ESP8266 aussi présenté dans ce magazine à de nombreuses reprises. Comptez environ 1,6 euros pour le modèle ESP-01 équipé de 4 GPIO (digitales, U-ART, SPI ou I²C), un processeur cadencé à 80mhz, 96KB de ram et surtout disposant d'une interface Wifi. On rajoutera une plaque epoxy pour réaliser notre circuit, un régulateur de tension 3,3v pour alimenter l'ESP, des connecteurs JST pour connecter la bande LED et surtout une alimentation 5v fournissant un ampérage suffisant. A ce sujet, une LED RGB type SMD5050 qui équipe notre bande de LED consomme au maximum 60mA pour du blanc (20mA par couleur : rouge, vert, bleu). Il faudra donc une alimentation capable de fournir 9,6A si on souhaite allumer les 5 mètres en blanc à 100% (255/255/255).

Au final, côté budget, mon nouvel éclairage sera rentabilisé en moins d'un an en m'affranchissant de l'achat des ampoules halogènes avec la possibilité d'aller beaucoup plus loin qu'un simple On/Off.

Réalisation hardware

Pour ce faire on a donc un ESP8266 au format ESP-01, un régulateur de tension 3,3v avec deux condensateurs 10uF, un ou plusieurs connecteurs JST 4 broches, une embase d'alimentation, un boitier et une plaque epoxy. [1]

Pour la connectique, la bande de LED WS2801 sera connectée à notre boitier par un connecteur JST 4 broches. L'alimentation 5v alimentera directement la bande de LED et le régulateur LD33 qui, lui, sort une tension de 3,3v pour l'ESP-01. Les pins CK et SD (clock et data du SPI) pour contrôler la bande de LED seront connectées sur les GPIO 0 et 2 de l'ESP. [2]

Soyez vigilant sur le fait que les premières LED de votre bande risquent de griller si elles transportent la seule source d'énergie de votre bandeau

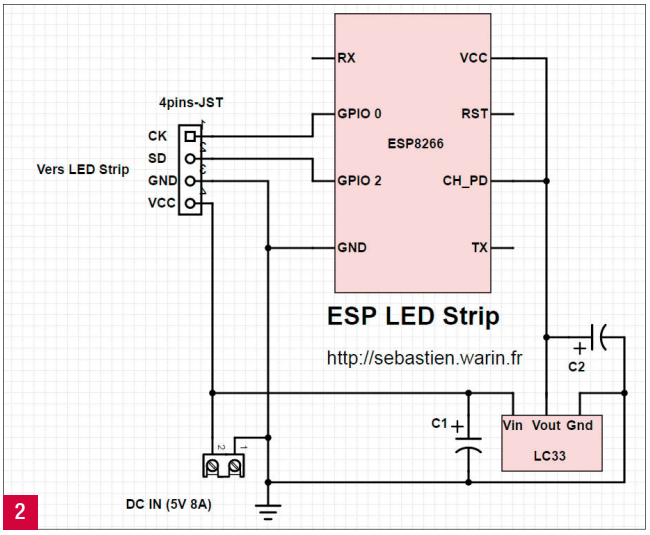


Figure 2 - Schéma

(forte intensité). C'est pourquoi il est conseillé d'alimenter votre bandeau tous les 2/3 mètres. J'ai donc dans mon cas deux connecteurs JST en sortie : le premier pilote par SPI l'intégralité du bandeau et alimente la 1ère moitié, le deuxième quant à lui ne sert qu'à alimenter la 2ème moitié. Une fois réalisé votre boitier n'attend plus que son microprogramme.

Conception logicielle

Les ESP8266 peuvent être programmés avec plusieurs frameworks et plusieurs langages. Nous pouvons par exemple utiliser l'IDE Arduino pour développer nos ESP comme des Arduinos et ça tombe bien car il existe un SDK Constellation pour Arduino ! Constellation est une plateforme d'interconnexion des applications, services et objets connectés. Pour plus d'information rendez-vous sur le portail développeur : https://developer.myconstellation.io.
Vous trouverez sur ce portail toutes les documentations et tutoriels pour déployer votre Constellation et interconnecter vos Arduino/ESP à vos programmes, services, objets, pages Web, etc., que vous développez en .Net, en Python, en Javascript ou autre. On va également installer la librairie d'Adafruit nommée « Adafruit_WS2801 » pour piloter la bande de LED. Au démarrage il suffit d'initialiser cette librairie de la façon suivante :

```
// Init led strip
strip.begin();
strip.show();

// Update the nummber of leds from setting
JsonObject& settings = constellation.getSettings();
strip.updateLength(settings["NumberLeds"].as<int>());
```

Le nombre de LED est un setting défini et centralisé dans Constellation. Si vous changez de bandeau par exemple, nul besoin de reprogrammer votre ESP pour changer cette valeur de configuration, il suffira, depuis la Console Constellation, de mettre à jour ce setting : [4]
Constellation apporte également la notion de MessageCallback permettant d'exposer des méthodes dans un catalogue central, c'est-à-dire que chaque service, application, objet connecté ou page Web et quel que soit le langage utilisé, peut exposer des méthodes dans Constellation qu'on va pouvoir invoquer depuis n'importe où.
Dans notre code Arduino, enregistrons un MessageCallback nommé « SetColor » qui prend en paramètre un objet de type « Color » et qui permet d'allumer chaque LED de notre bandeau de cette couleur.

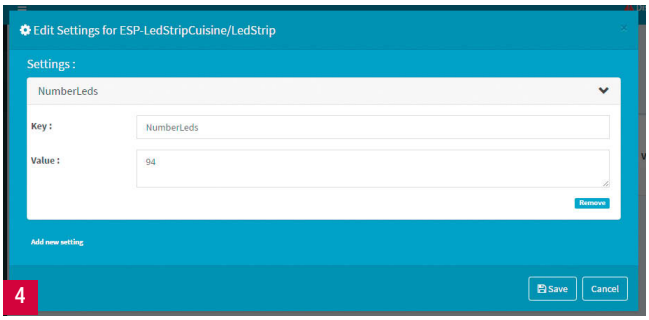


Figure 4 - Gestion des settings dans Constellation

```
constellation.registerMessageCallback("SetColor",
  MessageCallbackDescriptor().setDescription("Sets the color of the LED strip").addParameter
("color", "Color", "The RGB color"),
  [](JsonObject& json) {
  uint32_t color = Color(json["Data"][0]["R"].as<int>(), json["Data"][0]["G"].as<int>(), json
["Data"][0]["B"].as<int>());
    constellation.writeInfo("Setting color to %d", color);
    for (int i=0; i < strip.numPixels(); i++) {
      strip.setPixelColor(i, color);
    }
    strip.show();
  );
```

On peut également décrire notre objet « Color » dans Constellation de la façon suivante :

```
TypeDescriptor color = TypeDescriptor().setDescription("RGB color").addProperty<byte>
("R", "The Red component").addProperty<byte>("G", "The Green component").addProperty
<byte>("B", "The Blue component");
constellation.addMessageCallbackType("Color", color);
```

Une fois notre code téléversé sur l'ESP, on peut explorer le catalogue de Message Callbacks grâce à la Console Constellation et retrouver notre MC « SetColor » : [5].
L'interface permettra de tester chaque MC exposé dans Constellation et même de générer des snippets de code dans différents langages pour invoquer notre méthode « SetColor » depuis un programme C#, Python, une page Javascript, un simple appel http, etc… [6]
On peut donc très facilement enregistrer plusieurs MessageCallbacks pour exposer plusieurs méthodes de notre ESP dans Constellation. Par exemple, on pourrait exposer une méthode pour créer un effet clignotement, idéal pour notifier les utilisateurs. Pour cela enregistrons un MC nommé « Blink » prenant en paramètre la couleur (objet Color), le nombre de clignotements et le temps entre chaque clignotement :

```
constellation.registerMessageCallback("Blink",
  MessageCallbackDescriptor().setDescription("Blinks the LEDs").addParameter("color", "Color",
  "The RGB color").addParameter<short>("wait", "The delay of blink").addParameter<short>
("repeat", "Number of repeats for Blink"),
  [](JsonObject& json) {
    static uint32_t color;
    static short wait;
    color = Color(json["Data"][0]["R"].as<int>(), json["Data"][0]["G"].as<int>(), json["Data"]
[0]["B"].as<int>());
    wait = json["Data"][1].as<short>();
    short repeat = json["Data"][2].as<short>();
    constellation.writeInfo("Blink %d time(s) for %dms", repeat, wait);
    for (int j = 0; j <= repeat * 2; j++) {
      for (int i=0; i < strip.numPixels(); i++) {
        strip.setPixelColor(i, j % 2 == 0 ? color : 0);
      }
```

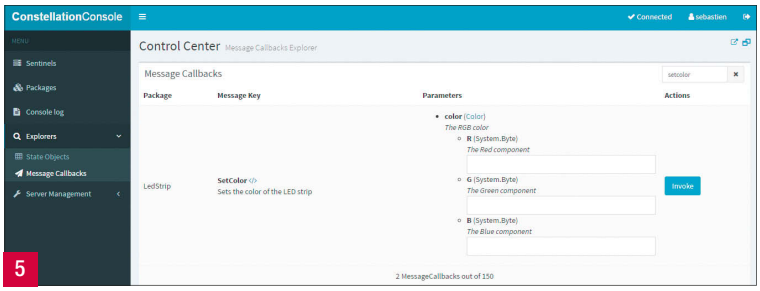


Figure 5 - MessageCallbacks Explorer

Figure 6 - Code Snippets

Figure 7 - Pilotage LED de manière individuelle

```
    strip.show();
    delay(wait);
  }
  // Restore current color
  showCurrentColor();
);
```

Ainsi depuis une page Web par exemple, je peux faire clignoter ma bande de LED en vert 3 fois de suite à une vitesse de 500ms avec la simple ligne Javascript :

```
constellation.server.sendMessage({ Scope: 'Package', Args: ['LedStrip'] }, 'Blink', [ { "R":0, "G":255, "B":0 }, 500, 3 ]);
```

On peut également ajouter d'autre MC pour créer des effets type K2000, d'arc-en-ciel, de « cross-fading » pour passer d'une couleur à une autre, etc… On peut également améliorer notre MC « SetColor » en ajoutant des paramètres optionnels comme un « from » et un « to » pour contrôler individuellement chaque LED ou même marquer une pause dans l'application des couleurs à nos LED pour donner un côté animé ! Par exemple toujours en Javascript, pour supporter la France lors du mondial de handball:

```
constellation.server.sendMessage({ Scope: 'Package', Args: ['LedStrip'] }, 'SetColor', [ { "R":0, "G":0, "B":255 }, 0, 0, 30 ]);
constellation.server.sendMessage({ Scope: 'Package', Args: ['LedStrip'] }, 'SetColor', [ { "R":255, "G":255, "B":255 }, 0, 31, 30 ]);
constellation.server.sendMessage({ Scope: 'Package', Args: ['LedStrip'] }, 'SetColor', [ { "R":255, "G":0, "B":0 }, 0, 61, 30 ]);
```

[7]

Pour finir, à chaque fois que l'on change la couleur de la bande de LED on publie deux StateObjects : « Color » et « State » contenant respectivement la couleur actuelle et l'état (on/off) de la bande de LED via la méthode « pushStateObject » :

```
constellation.pushStateObject("Color", stringFormat("{ 'R':%d, 'G':%d, 'B':%d }", r, g, b), "Color");
constellation.pushStateObject("State", currentColor > 0);
```

Les StateObjects sont des variables publiées par les packages Constellation (applications, services ou objets) permettant de partager des informations aux autres packages d'une Constellation.
Ainsi on pourra connaître, en s'abonnant au StateObject « State » de ce package « LedStrip », si notre éclairage est allumé ou non.

Installation

Maintenant notre puce programmée, il suffit de fixer la bande de LED avec de l'adhésif double face que je consolide avec de petites équerres vissées sous les placards. A proximité de l'arrivée 220v on fixera avec un bon adhésif l'alimentation 5V et notre boitier renfermant l'ESP8266. [9]
Et voilà, le nouvel éclairage connecté de la cuisine est opérationnel !

Intégration dans le smart-home

Dans le dashboard S-Panel

Je vous avais déjà présenté S-Panel il y a deux ans dans ce magazine, un dashbaord domotique permettant d'avoir une visibilité sur l'ensemble des équipements de la maison, de la consommation des différentes ressources énergétiques, en passant par l'état des serveurs, des températures des pièces, contrôle des éclairages, des volets, du chauffage, sonorisation des pièces, médias-centers, des TVs, etc.
Ce dashboard étant une application Web full-client side développée en Angular-JS et Bootstrap. Pour ajouter sur ce dashboard le contrôle de notre éclairage LED, il suffit de s'abonner au StateObject « State » pour connaître son état et d'invoquer le MessageCallback « SetColor » pour définir son état, sa couleur, son intensité.
On placera donc dans un div de notre code HTML, deux inputs de type « radio ». Lors du clic, on appellera la méthode « SetLedStrip » avec la couleur souhaitée (ou 0/0/0 pour éteindre). Notez qu'on utilisera la directive Angular « ng-checked » pour positionner le switch sur l'état courant de la bande de LED (via la variable de scope « LedStripState ») permettant d'avoir une interface parfaitement synchronisée.

```
<div class="switch">
  <input ng-click="SetLedStrip(255, 180, 20)" ng-checked="LedStripState == true" type="radio" class="switch-input" id="switchLedStripOn" >
  <label for="switchLedStripOn" class="switch-label switch-label-off">Ouvert</label>
  <input ng-click="SetLedStrip(0, 0, 0)" ng-checked="LedStripState == false" type="radio" class="switch-input" id="switchLedStripOff">
  <label for="switchLedStripOff" class="switch-label switch-label-on">Fermé</label>
</div>
```

La fonction « SetLedStrip » invoque simplement notre MC « SetColor » avec l'API Constellation pour Angular :

```
$scope.SetLedStrip = function(r, g, b) {
  constellation.sendMessage({ Scope: 'Package', Args: ['LedStrip'] }, 'SetColor', [ { "R": r, "G":
 g, "B": b }, 10 ]);
};
```

Et pour finir, on s'abonne au StateObject de l'état de notre bande de LED en enregistrant simplement un « StateObjectLink » sur le StateObject nommé « State » du package « LedStrip » permettant de mettre dans notre variable de scope Angular l'état de la LED strip en temps réel.

```
constellation.registerStateObjectLink("*", "LedStrip", "State", "*", function (so) {
   $LedStripState = so.Value
   $scope.$apply();
});
```

Et voilà, avec quelques lignes de Javascript et un peu d'HTML notre dashboard Web peut maintenant piloter la bande de LED et est synchronisé en temps réel à son état. [11]
On pourrait également enrichir notre page Web en ajoutant un « Color Picker » afin de pouvoir sélectionner la couleur de l'éclairage ou bien encore ajouter des boutons pour invoquer les autres MessageCallbacks pour créer différents effets !

Dans le cerveau de la maison

Pour créer un StateObjectLink en C#, il suffit d'ajouter un attribut sur une propriété .NET :

```
[StateObjectLink("LedStrip", "State")]
public StateObjectNotifier LedCuisine { get; set; }
```

Ici la propriété .NET « LedCuisine » est liée au StateObject « State » de notre bande de LED en temps réel. Dès que l'état de la Led Strip change, la propriété .NET sera instantanément mise à jour.
Pour invoquer le MessageCallback « SetColor » de notre LED Strip afin d'allumer toutes les LED en rouge en C#, on écrira :

```
PackageHost.CreateMessageProxy("LedStrip").SetColor(new { R:255, G:0, B:180});
```

Ainsi le cerveau de la maison, qui est un package Constellation écrit en C#, peut facilement piloter la bande de LED pour allumer ou éteindre les LEDs de manière automatique en fonction de l'occupation de la maison et de la luminosité. L'occupation de la maison étant connue grâce aux StateObjects du système d'alarme et la luminosité grâce à un autre ESP8266 équipé d'un capteur de Lux qui publie à intervalle régulier un StateObject nommé « Lux » contenant le nombre de lux mesuré.
Ainsi dès lors que la maison est désarmée, qu'il y a du mouvement dans la Salle à manger / Cuisine et qu'il commence à faire sombre, l'éclairage se met en route et lorsqu'il fait jour ou que l'on quitte la maison ou part se coucher à l'étage, l'éclairage s'éteint !

On sonne !

Je vous avais présenté ma sonnette connectée l'an passé dans ce magazine. Pour rappel il s'agit d'un simple ESP8266 connecté à Constellation qui permet de savoir quand on appuie sur le bouton de la sonnette.
Dès lors que l'on sonne à la porte, l'ESP envoie un message nommé « PushButton » dans un groupe Constellation. Il suffit donc de créer une méthode, en C# par exemple, et l'exposer en ajoutant l'attribut « MessageCallback ». Ainsi dès que l'on sonne, la méthode C# ci-dessous sera implicitement invoquée :

```
[MessageCallback]
public void PushButton()
```

Figure 9 - L'ESP8266 et son alimentation

Figure 11 - S-Panel

```
{
   PackageHost.WriteInfo("On sonne à la porte !");
   // Envoie de l'image courant de la caméra sur le smartphone
PackageHost.CreateMessageProxy("PushBullet").PushFile(cameraRue.StreamUri,  "On
 sonne !", PushTargetType.Device, null);
   // Si l'alarme n'est pas armée
   if (!this.IsArmed)
   {
      // Faire clignoter les lampes Hue du salon
      PackageHost.CreateMessageProxy("Hue").ShowAlert(1, 0, 255, 0, 2000);
      // Faire clignoter la bande de LED de la cuisine
       PackageHost.CreateMessageProxy("LedStrip").Blink(new { R = 0, G = 255, B = 0 },
 500, 4); // 4x (2x500) ms
   }
   // Réveiller le miroir de l'entrée et afficher le flux vidéo de la caméra
   theMirror.DoorBellRing();
}
```

Dans le code de cette classe C# il y a également deux StateObjectsLink : l'un vers un StateObject du système d'alarme permettant de savoir si l'alarme est armée ou non (et donc d'en déduire s'il y a quelqu'un ou non dans la maison) et l'autre vers le StateObject représentant la caméra de la porte d'entrée qui contient entre autres l'URI du flux vidéo.
Ainsi quand on sonne, le code C# invoque le MC « PushFile » du package PushBullet de façon à envoyer une notification sur mon smartphone accompagnée de la capture de la caméra au moment où l'on sonne et, dans le cas où l'alarme n'est pas armée, de faire clignoter les lampes Hue du salon et également notre bande de LED installée dans la cuisine. Ce bandeau lumineux peut ainsi servir à notifier un tas d'évènements. On sonne : ça clignote en vert, mon site Web ne répond plus : ça clignote en rouge (si et seulement si le StateObject représentant la liste des clients Wifi actuellement connectés indique que mon smartphone est connecté sur la borne de la Cuisine/Salle à manger !), il pleut alors qu'une fenêtre est restée ouverte : ça clignote bleu, la porte de garage est restée ouverte lorsqu'il n'y a plus de mouvement sur la camera : ça clignote violet, etc.
Toutes ces informations (le site Web ne répond plus, il pleut, une fenêtre est ouverte, activité sur les caméras vidéo, etc.) sont des StateObjects publiés par des différents packages que vous retrouverez dans le catalogue Constellation. •

S-Fit : concevez un miroir connecté orienté fitness

C'est l'été, la saison des maillots de bains, il est grand temps de se prendre en main et de se sculpter un corps de rêve. Pourquoi ne pas utiliser une des nombreuses solutions de tracker d'activités présentes sur le marché ? Ce n'est pas assez drôle pour des makers, nous avons donc décidé de créer notre propre solution fitness axée autour d'un miroir connecté !

1 *Résultat final du miroir connecté à Constellation*

L'équipe au complet, de gauche à droite : Valentin BEQUART, Pierre-Alexandre CHOAIN, Hugo MROCZKOWSKI, Milan FERTIN, David BRICENO-AGUILERA

Nous sommes 5 étudiants en troisième année de Cycle Informatique et Réseaux à l'ISEN et nous avons conçu un nouveau concept de solution fitness basée sur un miroir. Pour réaliser notre projet, nous avions un budget de 0€ mais nous avions surtout une grande motivation pour créer un produit innovant et agréable à utiliser.

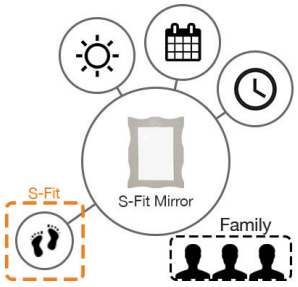
C'est pour cela que nous avons utilisé des produits de récupérations. En effet, nous avons tous dans notre garage un ordinateur portable que nous n'utilisons plus, une ancienne webcam, et quelques planches de contreplaqué. Concernant l'aspect miroir, nous avons utilisé du film sans tain car nous en avions déjà, cependant, pour une dizaine d'euros de plus, vous pourrez utiliser une vitre sans tain. Cette dernière donnera un rendu bien meilleur à votre miroir. Voilà qui devrait suffire pour la partie matérielle de notre projet.

Pour la partie logicielle, nous avons utilisé la plateforme Constellation. Les lecteurs réguliers de ce magazine la connaissent déjà, pour les autres, il s'agit d'une plateforme technique d'orchestration et d'interconnexion des objets, des services et des applications. Elle s'appuie sur des paquets qui peuvent publier et consommer des messages ainsi que sur des fonctions partagées. Concrètement, avec Constellation, en quelques lignes, il devient très simple de connecter des objets (ou applications) entre eux. Ces derniers vont donc dialoguer via Constellation comme le feraient des micro-services. L'avantage d'utiliser cette plateforme pour un tel projet c'est la facilité avec laquelle nous avons pu connecter et déployer les différentes briques de notre miroir. Pour en savoir plus sur cette technologie, vous pouvez vous rendre sur http://www.myconstellation.io/

Pour résumer, en raison d'un coût très faible et d'un développement simplifié, S-Fit est le projet parfait pour vous occuper cet été.

Fonctionnement général

Nous avons tout d'abord pensé S-Fit comme une application dotée d'un podomètre. Cette dernière synchronise les différents profils des membres de la famille en temps réel grâce à Constellation. Vous pouvez ainsi y gérer vos propres objectifs et surveiller votre progression. Vous ne perdrez pas votre motivation grâce à notre système de trophées qui vous donnera envie de repousser vos limites chaque jour. Comme il n'existe pas de meilleure motivation que la compétition, vous pourrez vous comparer à vos proches grâces à des outils d'analyse s'appuyant sur une série de graphiques. [2]



2 *Les sources d'informations du miroir*

Pour vous rappeler vos objectifs chaque matin, nous avons ajouté à S-Fit un miroir compagnon. Ce dernier a demandé beaucoup de réflexion car il s'agit d'un nouvel objet avec lequel il faut interagir de manière naturelle. De plus, il fallait faire de ce dernier un bel objet que l'on puisse retrouver chez soi. Nous avons donc fait le choix d'une interface minimaliste qui affiche seulement les informations pertinentes : la météo, les évènements à venir et votre progression. De ce fait, pas besoin de toucher le miroir et d'y laisser des traces de doigts.

Pour gérer les multiples profils, nous avons également intégré un module de reconnaissance faciale qui permettra au miroir d'afficher des informations personnalisées en fonction de son utilisateur. [1]

Comme vous vous en doutez sûrement, le lien entre l'application et le miroir se fait par l'intermédiaire de la plateforme Constellation. Tout est synchronisé en temps réel et cela fonctionne comme par magie.

Etape 1 : Conception du boîtier

Pour commencer, il faut démonter votre vieil ordinateur, afin d'en récupérer la dalle LCD. On utilise ensuite la référence de cette dernière pour pouvoir se procurer le contrôleur adapté.

Ensuite, il faut concevoir un boîtier capable d'accueillir l'ensemble de votre appareil. Son épaisseur et ses dimensions dépendent donc de votre miroir. Nous ne fournirons donc pas de plans pour rendre votre création unique.

Attention toutefois à prévoir des espaces pour l'aération, l'alimentation et les contrôles de la dalle.

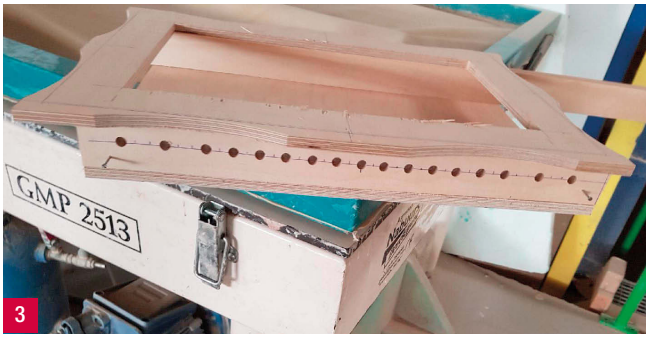
C'est la partie la plus personnelle du projet, c'est le moment de libérer

votre créativité pour mettre en place votre vision de S-Fit. [3]
Nous avons également prévu une trappe d'accès à l'arrière pour pouvoir modifier notre miroir plus tard.
Si vous avez fait le choix du film sans tain, il va falloir le poser. Pour cela, voici les quelques étapes à suivre :
Nettoyer votre dalle à l'aide d'un chiffon doux
Appliquer un peu d'eau savonneuse sur celle-ci
Poser le film sans tain petit à petit en vous aidant d'un grattoir. Attention à ne pas rayer le film avec, c'est très fragile !
Chassez, toujours avec ce grattoir, les dernières bulles d'air
Prenez bien votre temps lors de la pose, c'est une partie très délicate et elle affectera directement l'esthétique de votre miroir.
Vous avez maintenant l'ensemble des pièces qui vont constituer votre miroir. Pour terminer, il ne vous reste plus qu'à tout assembler en faisant attention à bien aligner la dalle et le boîtier. [4]

Etape 2 : Le développement logiciel

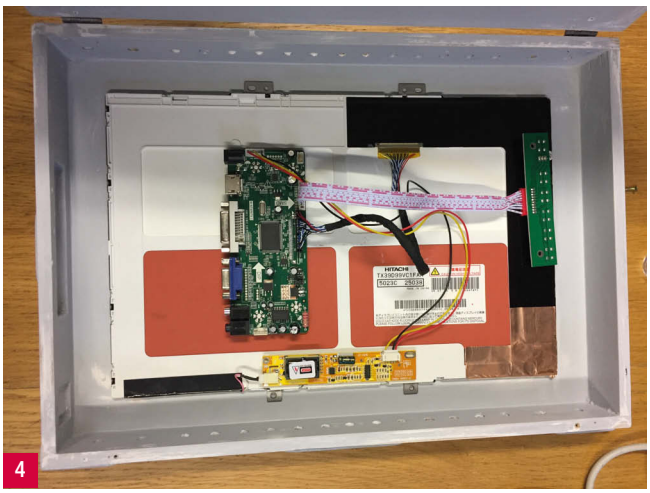
Etape 2.1 : L'interface du miroir

Pour réaliser le miroir, nous avons choisi de concevoir une application Web avec AngularJS. En effet, comme le dit le créateur de Constellation, Sébastien Warin, on peut connecter n'importe quoi avec quelques lignes de code qui vont bien.
Tout d'abord, il est important de rappeler que pour continuer ce tutoriel, il est nécessaire d'avoir une Constellation déployée chez soi. Vous trouverez la plateforme ainsi que les tutoriels de prise en main sur le portail https://developer.myconstellation.io/ [5]
Nous allons donc pouvoir connecter notre application Angular à Constellation :



3

Assemblage du boîtier et vernissage


4

Le miroir assemblé, prêt à être refermé

```
var app = angular.module('Mirror', ['ngConstellation']);
app.controller('MyController', ['$scope',  'constellationConsumer', ($scope,
 constellation) => {

  constellation.initializeClient("maconstellation.local", "masupercle123", "MyMirror");

  constellation.connect();

}]);
```

Il ne reste plus qu'à s'abonner aux StateObjects de Constellation que l'on veut voir sur le miroir. Par exemple, ici, nous allons récupérer la météo dans la ville de Lille :

```
constellation.registerStateObjectLink("*", "ForecastIO", "Lille", "*", (so) => {
        $scope.$apply(() => {
            $scope.temperature = so.Value.currently.temperature;
});
});
```

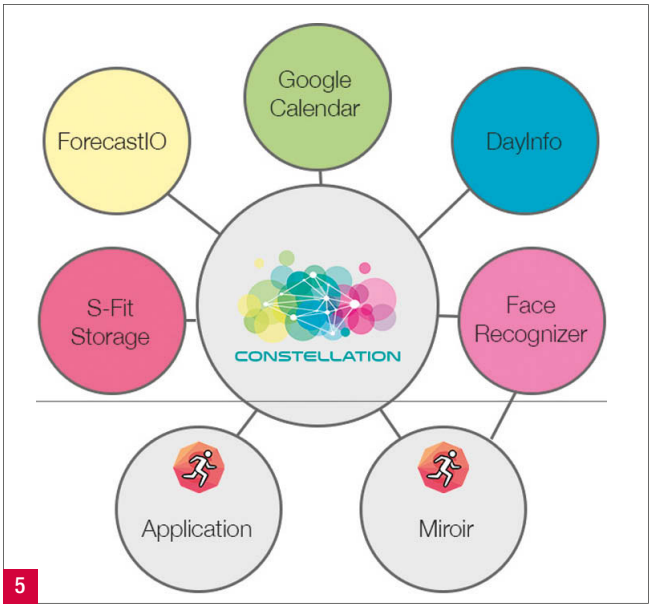
Pour en savoir plus, vous pouvez vous rendre sur le portail dont le lien se trouve plus haut pour y trouver la documentation complète. Vous trouverez d'ailleurs un tutoriel détaillé sur l'utilisation de Constellation en JavaScript. Mais rassurez-vous, ce n'est pas plus compliqué que cela. Il ne manque que quelques lignes d'HTML et de CSS pour donner vie à votre miroir. Si on continue l'exemple de la météo, le code HTML associé pourrait-être le suivant :

```
<p>{{temperature}}</p>
```

On obtiendrait alors une page sur laquelle la température va s'afficher dynamiquement. [6]

Etape 2.2 : La reconnaissance faciale

Comme nous l'expliquions plus haut, nous avons ajouté un module de reconnaissance faciale pour gérer plusieurs profils. Il s'agit d'une partie facultative et plutôt complexe. Pour cela, nous avons développé un



5

Le rôle de Constellation dans la plateforme S-Fit

paquet Constellation qui analyse les images de la webcam avec la librairie EMGU.CV. C'est un portage en C# d'OpenCV. Lors de l'ajout d'un nouvel utilisateur S-Fit, le paquet lance la séquence d'enregistrement d'un visage automatiquement.

Nous avons effectué plusieurs essais sur la quantité d'images à enregistrer, afin d'obtenir l'équilibre idéal entre une reconnaissance optimale et un minimum d'espace utilisé. Pour vous reconnaître, l'algorithme s'appuie sur plusieurs caractéristiques faciales, comme la forme du nez, de la bouche, des yeux, de vos sourcils…

De par le peu d'espace pris par la reconnaissance, vous pouvez aisément enregistrer toute votre famille, afin que le miroir devienne un élément à part entière de votre lieu de vie, et que tout le monde participe à la compétition!

Pour rendre la gestion des utilisateurs agréable, nous avons intégré cette reconnaissance faciale de manière totalement transparente. Lorsqu'un utilisateur s'enregistre dans l'application il doit être face au miroir. Pour vérifier cela, l'utilisateur sera invité à saisir un code à six chiffres qui s'affichera quelques secondes sur le miroir. Une fois l'ensemble des informations saisies dans l'application, le miroir va automatiquement lancer une séquence de capture de 100 clichés du nouvel utilisateur.

Pour la reconnaissance des utilisateurs enregistrés, le paquet de reconnaissance faciale va capturer une image chaque seconde pour vérifier la présence ou non d'un individu connu.

Lorsque deux utilisateurs enregistrés sont face au miroir, ce dernier va se concentrer sur celui qu'il identifie le mieux.

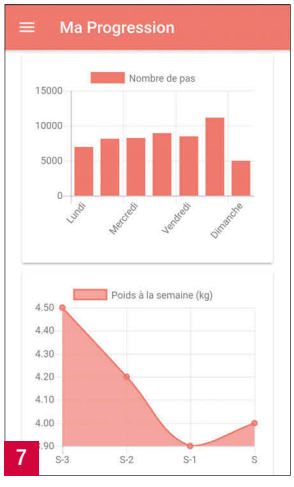
Etape 2.3 : L'application mobile

L'application S-Fit a été conçue avec les frameworks Ionic 3 et Apache Cordova. Ces frameworks permettent d'obtenir une application Web à l'intérieur d'une application native Android ou iOS qui embarque un serveur NodeJS sur le mobile. Comme nous l'avons vu plus haut, l'application qu'affiche le miroir est une page Web, l'application mobile utilise donc les mêmes technologies.

Ainsi, la connexion s'effectuera tout aussi simplement :

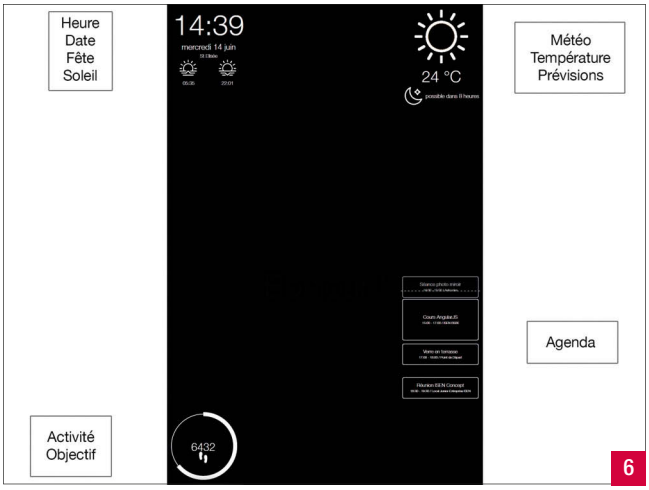
```
var constellation = $.signalR.createConstellationConsumer("maconstellation.local", "masupercle123", "MonApp");
constellation.connection.start();
```

Une application comme la nôtre peut alors consommer des StateObjects mais également envoyer des MessageCallbacks. C'est à dire exécuter des fonctions directement sur la Constellation. C'est particulièrement utile pour incrémenter le compteur de pas. [7]



L'application

Pour le podomètre, nous nous sommes appuyés sur un plugin de Cordova permettant d'accéder aux données de l'accéléromètre du mobile. A l'aide des données fournies par ce plugin, nous avons pu étudier les variations sur les axes x, y et z, dans l'optique de compter les pas. Le développement de cette application ne se résume pas qu'à de la programmation informatique. Nous avons également réalisé des mesures sur plusieurs dizaines de personnes afin d'obtenir un lien entre la morphologie et la longueur des pas.

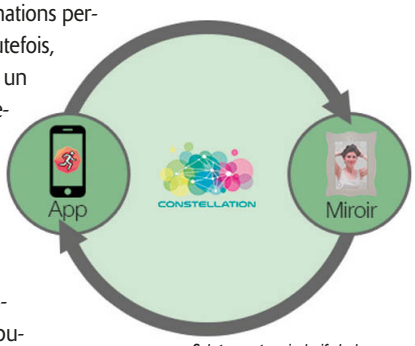


L'interface finale du miroir: le choix du noir et blanc permet un meilleur rendu sur le miroir

```
var slope = 0.64878048 ;
var origin = 44.6744 ;
function getDistance(size, stepCounter) {
  return (size * slope - origin) * stepCounter;
}
```

Avec les informations physiologiques et l'activité de l'utilisateur, on peut donc créer un ensemble de fonctions qui permettent d'étudier son état de santé, en calculant par exemple les calories dépensées chaque jour. C'est grâce à ces données qu'il est possible de créer une application fitness entièrement maîtrisée. On est alors libre d'appliquer les algorithmes souhaités sur les données récupérées.

Trêve d'explications physiques, intéressons-nous maintenant à la synchronisation des données. Lorsqu'un utilisateur lance l'application, il peut choisir son profil. Cette action va établir la connexion avec la Constellation pour récupérer les informations personnelles et l'historique d'activité. Toutefois, s'il n'a pas de profil, il peut en ajouter un s'il est en face du miroir, qui lui affichera alors un code de vérification. Lors du lancement de l'application, cette dernière synchronise instantanément les nouvelles données d'activités avec le serveur Constellation.



8 *Schéma récapitulatif de la synchronisation*

L'utilisation d'S-Fit est totalement transparente et ne demande pas de manipulation particulière de l'utilisateur. En effet, nous avons cherché à fournir un produit simple, accessible et entièrement automatisé. Cette synchronisation est permise par Constellation. [8]

CONCLUSION

Voilà qui conclut les grandes étapes de la réalisation de S-Fit. Comme vous avez pu le voir les possibilités de personnalisation sont très nombreuses. C'est un projet ludique et facile à réaliser. C'est également un bon point de départ pour prendre en main la plateforme Constellation. Nous espérons vraiment qu'il vous a plu et que vous allez réaliser votre propre version.

Nous tenons également à remercier Julie, Adrien et tous ceux qui se sont impliqués de près ou de loin dans la réalisation de ce projet.

Créer une prise connectée avec **Raspberry Pi Zéro**

• Estelle Auberix.
In Omnia Paratus
Consultante MVP Azure

Qui n'a jamais voulu comprendre quel est ou quels sont les appareils qui font le plus gonfler la facture d'électricité ? Voici un article qui va vous permettre de créer votre prise intelligente à partir d'une Raspberry Pi Zéro. Cet article est l'adaptation de plusieurs tutoriels en anglais.

Ce qu'il vous faut

- 1 Raspberry Pi Zéro avec un dongle WiFi ou une Raspberry Pi Zéro W (NDLR : pensez à installer le système Raspbian sur une micro-SD pour utiliser la Pi Zero).
- 1 transformateur de courant type non-invasif 30A (ex. ECS1030 disponible sur Amazon, Sparkfun, SeedStudio…).
- 1 résistance 10Ω afin de pouvoir connecter la Raspberry Pi.
- 1 connecteur jack TRRS 3.5mm pour brancher le transformateur sur la 'planche à pain' (disponible sur Amazon, Sparkfun…).
- 1 convertisseur Analogique vers Numérique MCP3008 avec interface SPI (souvent disponible dans les kits de démarrage pour Raspberry Pi, sinon dispo sur Amazon, Adafruit...)
- 1 PowerSSR Tail Kit (disponible sur powerswitchtail.com). [1]

NB : Cet élément pourra vous servir dans d'autres montages car il s'agit du moyen le plus simple de contrôler un périphérique avec un Arduino, un Raspberry Pi, un Beaglebone ou toute autre carte de ce type.

- 1 fer à souder pour assembler le PowerSSR.
- 1 'planche à pain' (Breadboard).
- Des câbles mâle/femelle.
- 1 appareil électrique de vore choix à contrôler (lampe ou autre).

Comment faire ?

Assembler le matériel

1 – Connecter le transformateur au PoweSwitch Tail [2]

2 – Connecter le convertisseur analogique MCP3008 à la Raspberry Pi Zéro en vous aidant des schémas suivants :
- VDD et VREF vers 3.3V pin de la Raspberry Pi ;
- DGND et AGND vers GND pin de la Raspberry Pi ;
- CLK vers GPIO 11 de la Raspberry Pi ;
- DOUT vers GPIO 9 de la Raspberry Pi ;
- DIN vers GPIO 10 de la Raspberry Pi ;
- CS vers GPIO 8 de la Raspberry Pi ; [3]

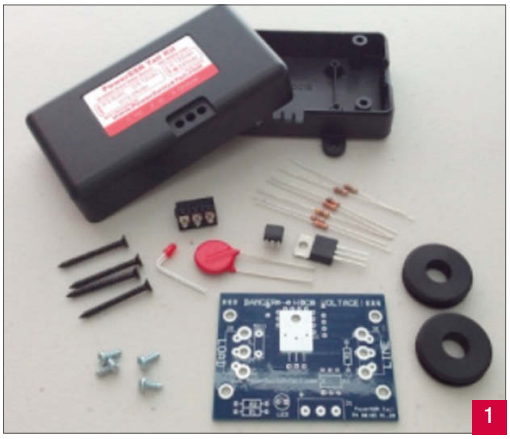
3 – Connecter la prise jack du transformateur au connecteur jack TRRS sur la planche à pain avec la résistance 10 reliant SLEEVE et TIP sur la carte du connecteur.

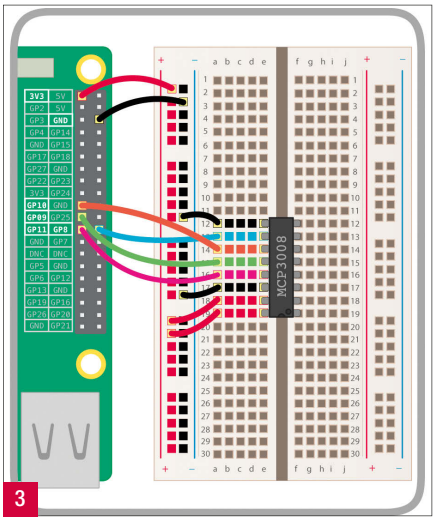
4 – Connecter TIP du connecteur à un GND de la Raspberry Pi et SLEEVE du connecteur au CH5 du MCP3008.

5 – Connecter le PowerSwitch Tail à la Raspberry Pi : Vin+ au GPIO 18 et les deux autres câbles à GND.

6 – Connecter enfin l'appareil électrique que vous souhaitez contrôler à la prise femelle du PowerSwitch Tail.

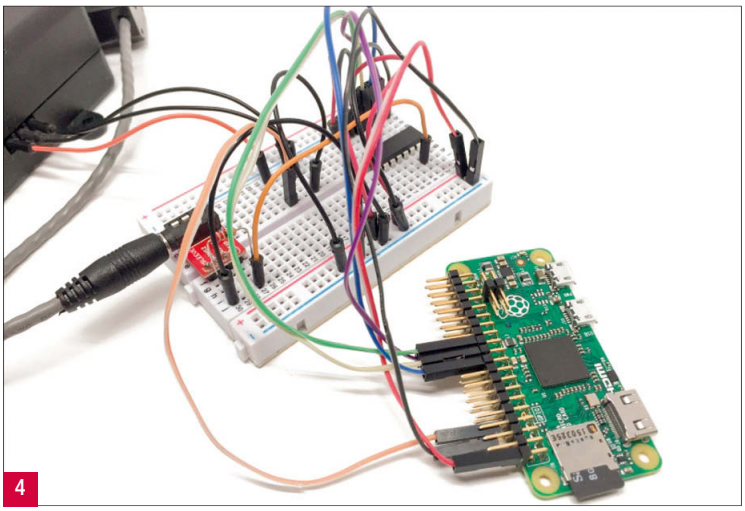
7 – Alimenter votre Raspberry Pi et votre PowerSwitch à la même source d'électricité. [4]


1



3


2


4

Configurer la prise :

1 – Configurer la Raspberry Pi pour quelle se comporte en prise intelligente grâce à Node.js : création du fichier **meter.js**.
Importation des modules nécessaires au projet :

```
var mcpadc = require('mcp-spi-adc');
var express = require('express');
var app = express();
var piREST = require('pi-arest')(app);
```

NB : On utilise le module mcp-spi-adc pour lire les données fournies par le chipset MCP3008.
Définition du canal où est connecté le capteur du transformateur :

```
var channel = 5;
```

Définition de la valeur de la résistance afin de calculer le flux dans le transformateur :

```
var resistance = 10;
```

Configuration du contrôle distant en utilisant le framework aREST. Il faut initialiser le module en donnant un nom (name_project) et un ID de 6 caractères (ID_project) au projet :

```
piREST.set_id('ID_project');
piREST.set_name('name_project');
piREST.set_mode('bcm');
```

Initialisation des valeurs des variables :

```
Current = 0 ;
Power = 0 ;
```

Démarrage du serveur sur le port 80 :

```
var server = app.listen(80, function() {
   console.log('Listening on port %d', server.address().port);
});
```

Mesure des données du capteur grâce au code suivant. La valeur en rouge indique l'intervalle de mesure (ici 500ms) :

```
var sensor = mcpadc.open(channel, {speedHz: 20000}, function (err) {
 if (err) throw err;

 // Measurement interval
 setInterval(function () {

  // Read
  sensor.read(function (err, reading) {
   if (err) throw err;

   // Calculate current
   var measuredVoltage = reading.value * 3.3;
   var measuredCurrent = (measuredVoltage/resistance) * 2000 /1.41;

   // Calculate power
   var power = voltage * measuredCurrent;

   // Assign to aREST
   piREST.variable('power', power.toFixed(2));
   piREST.variable('current', measuredCurrent.toFixed(2));

   // Log output
   console.log("Measured current: " + measuredCurrent.toFixed(2) + 'A');
   console.log("Measured power: " + power.toFixed(2) + 'W');

  });
 }, 500);
});
```

2 – Placer le fichier meter.js ainsi créé dans un dossier de la Raspberry Pi.
3 – Installer les modules nécessaires depuis un terminal via les commandes suivantes :

```
sudo npm install express pi-arest mcp-spi-adcsdfsd
```

4 – Démarrer le projet avec la commande suivante :

```
sudo node meter.js
```

NB : Vous devez immédiatement voir les mesures sur la console avec des valeurs nulles
Measured current: 0.00A
Mesured power: 0.00W
5 – Trouver l'adresse IP de votre carte Raspberry Pi avec la commande :

```
Ifconfig
```

NB : Pour la suite de ce tutoriel, on définit l'adresse IP par 192.168.0.105

6 – Ouvrir une fenêtre de navigateur sur l'ordinateur et inscrire l'URL suivante : http://192.18.0.105/digital/18/1

NB : Vérifier d'être bien connecté sur le même réseau WiFi

Vous devriez alors voir apparaître les valeurs de consommations de l'appareil que vous avez connecté au PowerSwitch Tail.
Measured current: valeur en A
Mesured power: valeur en W

Créer une interface pour la prise intelligente :

1 – Déclarer un dossier 'public' dans le dossier où se trouve le fichier Node.js que l'on vient de créer ('meter.js') afin d'y stocker l'interface Web de la prise : création du fichier **meter_interface.js**.
Importation des modules nécessaires au projet :

```
var mcpadc = require('mcp-spi-adc');
var express = require('express');
var app = express();
```

Utilisation du dossier 'public' :

```
app.use(express.static('public'));
```

Déclaration du pin où est connectée la prise :

```
var outputPin = 18;
```

Définition du chemin principal de l'application pour redirection vers l'interface :

```
app.get('/', function (req, res) {
 res.sendfile(__dirname + '/public/interface.html');
});
```

Déclaration des deux chemins de contrôle de sorties du projet : (l'un pour allumer l'appareil, l'autre pour l'éteindre)

```
app.get('/on', function (req, res) {
```

```
  piREST.digitalWrite(outputPin, 1);

  // Answer
  answer = {
    status: 1
  };
  res.json(answer);

});

app.get('/off', function (req, res) {

  piREST.digitalWrite(outputPin, 0);

  // Answer
  answer = {
    status: 0
  };
  res.json(answer);

});
```

Déclaration de l'instace Pi-aREST :

```
var piREST = require('pi-arest')(app);
```

Définition du canal où est connecté le capteur du transformateur :

```
var channel = 5;
```

Définition de la valeur de la résistance afin de calculer le flux dans le transformateur :

```
var resistance = 10;
```

Configuration du contrôle distant (reprendre les valeurs données dans le fichier meter.js :

```
piREST.set_id('ID_project');
piREST.set_name('name_project');
piREST.set_mode('bcm');
```

Initialisation des valeurs des variables :

```
Current = 0 ;
Power = 0 ;
```

Démarrage du serveur sur le port 80 :

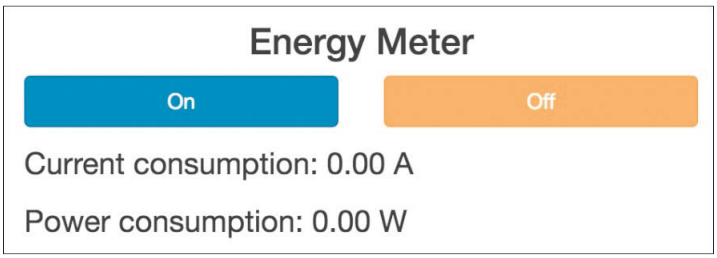
```
var server = app.listen(80, function() {
   console.log('Listening on port %d', server.address().port);
});
```

Mesure des données du capteur grâce au code suivant :
Code complet sur www.programmez.com
Création du fichier HTML qui contiendra les éléments de l'interface et du fichier JavaScript qui fera le lien entre les éléments et Node.js :
Fichier HTML 'interface.html' à mettre dans le dossier 'public' :
Code complet sur www.programmez.com
Fichier CSS 'style.css' à mettre dans le dossier 'public\css' :
Code complet sur www.programmez.com
Fichier JS 'script.js' à mettre dans le dossier 'public\js' :
Code complet sur www.programmez.com

2 – A partir du dossier où se trouve le fichier meter_interface.js, lancer la commande suivante :

```
sudo node meter_interface.js
```

3 – Tester l'interface en consultant l'IP de la Raspberry Pi : http://192.18.0.105/
Vous devriez obtenir un résultat similaire à celui-ci



En cliquant sur le bouton 'On', les valeurs doivent s'afficher en temps réel.

Enregistrer la consommation d'énergie dans le temps :

Afin de visualiser la consommation d'énergie sur un graphique au cours du temps, il faut enregistrer les mesures dans une base de données.
1 – Création du fichier meter_log.js dans le dossier où se trouvent déjà les autres fichiers de l'application :
Importation des modules nécessaires au projet :

```
var mcpadc = require('mcp-spi-adc');
var express = require('express');
var app = express();
```

Utilisation du dossier 'public' :

```
app.use(express.static('public'));
```

Création de la base de données :

```
var Datastore = require('nedb')
  db = new Datastore();
```

Déclaration du pin où est connectée la prise :

```
var outputPin = 18;
```

Définition des différents chemins :

```
app.get('/', function (req, res) {
  res.sendfile(__dirname + '/public/interface.html');
});
app.get('/on', function (req, res) {

  piREST.digitalWrite(outputPin, 1);

  // Answer
  answer = {
    status: 1
  };
  res.json(answer);

});

app.get('/off', function (req, res) {
```

```
  piREST.digitalWrite(outputPin, 0);

  // Answer
  answer = {
    status: 0
  };
  res.json(answer);

});
```

Définition du chemin pour obtenir les données dans la base :

```
app.get('/data', function (req, res) {

  db.find({}, function (err, docs) {

    res.json(docs);

  });

});
```

Déclaration de l'instace Pi-aREST :

```
var piREST = require('pi-arest')(app);
```

Définition du canal où est connecté le capteur du transformateur :

```
var channel = 5;
```

Définition de la valeur de la résistance afin de calculer le flux dans le transformateur :

```
var resistance = 10;
```

Configuration du contrôle distant (reprendre les valeurs données dans le fichier meter.js :

```
piREST.set_id('ID_project');
piREST.set_name('name_project');
piREST.set_mode('bcm');
```

Initialisation des valeurs des variables :

```
Current = 0 ;
Power = 0 ;
```

Démarrage du serveur sur le port 80 :

```
var server = app.listen(80, function() {
   console.log('Listening on port %d', server.address().port);
});
```

Mesure des données en boucle :

```
var sensor = mcpadc.open(channel, {speedHz: 20000}, function (err) {
  if (err) throw err;

  // Measurement interval
  setInterval(function () {

  // Read
  sensor.read(function (err, reading) {
    if (err) throw err;

    // Calculate current
    var measuredVoltage = reading.value * 3.3;
    var measuredCurrent = (measuredVoltage/resistance) * 2000 /1.41;

    // Calculate power
    var power = voltage * measuredCurrent;

    // Assign to aREST
    piREST.variable('power', power.toFixed(2));
    piREST.variable('current', measuredCurrent.toFixed(2));

    // Log
    var data = {
          current: measuredCurrent.toFixed(2),
          power: power.toFixed(2),
          date: new Date()
        };
    db.insert(data, function (err, newDoc) {
      console.log(newDoc);
    });

    // Log output
    console.log("Measured current: " + measuredCurrent.toFixed(2) + 'A');
    console.log("Measured power: " + power.toFixed(2) + 'W');

   });
  }, 500);
});
```

2 – A partir du dossier où se trouve le fichier meter_log.js, lancer la commande suivante :

```
sudo npm install nedb
sudo node meter_log.js
```

Vous devriez voir les résultats des mesures dans la console comme précédemment mais avec une confirmation d'enregistrement dans la base :

```
Measured current: 0.07A
Mesured power: 15.03W
{ current: '0.07',
  power: '15.03',
  date: Mon Jun 19 2017 20:00:00 GMT+0000 (UTC),
  _id: 'EzLoFvbzXK4tRGow' }
```

NB : Pour la démonstration, la valeur de rafraîchissement des données est élevée mais il est recommandé de la diminuer pour éviter de surcharger la base de données.

3 – Consulter les résultats sur l'interface du navigateur : http://192.18.0.105/data
Libre à vous de visualiser ces résultats sur un graphique en temps réel ou dans votre propre application. •

Créer son Makey Makey !

Jean Paul DELCROIX
Animateur multimédia et Fabmanager

• Nicolas THILL *stagiaire en 1ère SEN.*

Connaissez-vous la Makey Makey ? Cette carte électronique servant d'interface entre un PC et tous objets conducteurs, leur permettant de remplacer certaines touches du clavier. Vous n'avez jamais joué du piano sur des bananes ? Ou encore joué à Pacman avec des touches faites en pâte à modeler ?
Plus d'info dans cette vidéo : https://www.youtube.com/watch?v=rfQqh7iCcOU.

Je vous propose donc dans un premier temps de transformer votre Arduino Uno en pseudo Makey Makey ; puis dans un second temps d'adapter le code pour créer une Makey Makey à partir d'une Arduino Leonardo !
Pour réaliser une véritable Makey Makey, il faut utiliser une Arduino Leonardo, qui à la différence de la Uno, peut être considérée par le PC comme une souris ou un clavier. La Uno n'est détectée par l'ordinateur que comme un port série et non comme un périphérique USB.
Alors à quoi peut servir cette pseudo Makey Makey ? Eh bien elle n'est pas d'utilité avec un ordinateur, mais elle peut très facilement servir de lien entre un clou et une led, ou encore une cuillère et un servomoteur, ou bien de la pâte à modeler conductrice et un moteur continu…

Source

De nombreux sites nous annoncent que la Makey Makey et l'Arduino sont compatibles, ou encore que la Makey Makey est une dérivée de l'Arduino, mais même si nous trouvons quelques vidéos de personnes ayant réussi à le faire, ou à créer un shield, on ne trouve rien sur la méthode… La seule chose que nous ayons trouvée c'est cette vidéo en arabe dont mon tuto est une adaptation : https://www.youtube.com/watch?v=7rHi2XcFqgw.
Nous avons essayé de comprendre le principe, de plus une partie du code n'était pas apparente sur la vidéo, il nous a fallu la trouver… J'ai également modifié un peu le code supprimant certaines parties et changeant le nom des variables. Le code d'origine faisait s'allumer 3 leds. Je vous propose d'appuyer sur une banane et d'actionner une led témoin et un servomoteur !!!

Matériel

- Une carte Arduino et son câble USB
- 1 mini servomoteur 9g
- 9 câbles / jumpers
- 1 résistante de 1 MΩ et 1 résistance de 220
- 1 led de 5 mm
- 2 pinces crocodiles

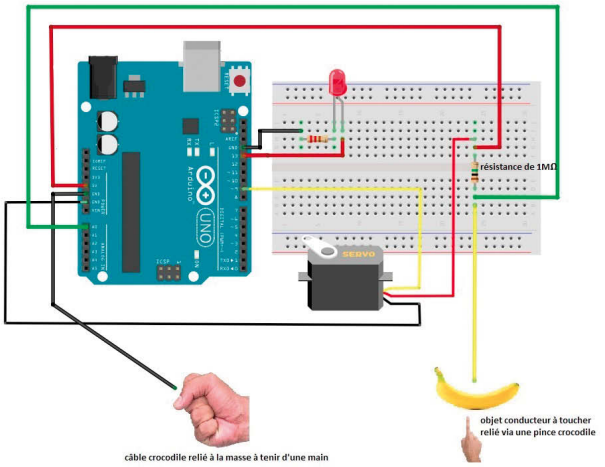
Budget : environ 32€10 sur Gotronic

Etape 1 : le montage

Il s'agit ici en fait de 3 montages distincts, le premier pour la Makey Makey (objet conducteur à toucher), le second pour le servomoteur, et le dernier (optionnel) pour une led témoin :

- Pour obtenir le ***montage Makey Makey***, le ground de l'Arduino doit être branché à une pince croco et tenu à la main.

La broche 5V, doit être connectée à une résistance de 1MΩ, qui elle-même est branchée à la fois à la broche A0 de l'Arduino et à une pince croco pour le relier à l'objet conducteur.



- Le ***servomoteur*** est branché d'un côté au ground, au 5V de l'Arduino, et à la broche 9 de l'Arduino. On peut imaginer remplacer le servomoteur par d'autres composants.
- La ***led témoin*** est branchée d'un côté au ground, via une résistance, ici de 220 , et de l'autre à la broche 13 de l'Arduino. Elle est optionnelle, mais permet de visualiser lorsque le courant passe.

Etape 2 : le code : Principe de base

L'Arduino envoie un courant de 5V qui va diminuer à cause de la résistance de 1MΩ et de la "résistance" rencontrée entre les 2 pinces crocos, donc entre l'objet conducteur et le corps de la personne touchant cet objet. Ce qui sera donc variable en fonction d'une part des personnes et des objets reliant les 2 pinces crocos.
Dans notre code, on appellera cette résistance entre les 2 pinces "résistivité".
Si cette dernière est inférieure au seuil qu'on définit (c'est à dire qu'il y a suffisamment de courant qui passe) alors la led s'allumera et le servomoteur se positionnera à l'angle défini.
Par exemple : les différentes personnes ayant servi de cobaye avaient une résistivité variant entre 2200 et 3700.
Dans le cas d'un dessin fait à la mine de crayon gris (graphite) la valeur de "résistivité" était un peu moins de 5000, une banane également testée tournait entre 5000 et 6000. De plus, cette résistivité à tendance à diminuer avec le temps.

Etape 3 : Déclaration des variables et des constantes

1) Pour la fonction Makey Makey

```
int sensorPin = A0;
int sensorValue = 0;
```

```
int values[10];
int k=0;
int resistivite=0;
```

Explication : on crée une variable de type "int" nommée "sensorPin" qui est affectée à la broche analogique A0 de l'Arduino.
On crée une variable de type "int" nommée "sensorValue" permettant d'y stocker la valeur que la carte détectera, elle est au départ initialisée à 0.
On crée un tableau nommé "values" contenant 10 cellules pour y stocker des variables de type "int". Les cellules sont pour l'instant vides.
On crée une variable de type "int" nommée "k", initialisée à 0, et qui va nous permettre de calculer la "résistivité".
On crée une variable de type "int" nommée "résistivité" qui symbolise en quelque sorte la résistivité électrique. Plus elle est élevée, moins le courant passe.

2) Led et servomoteur

```
int ledPin = 13;
```

```
#include <Servo.h>
Servo monservo;
```

Etape 4 : Void Setup

```
Serial.begin(9600);
```

```
pinMode(ledPin, OUTPUT);
```

```
monservo.attach(9);
```

Etape 5 : Void Loop

1) Pour la fonction makey

```
sensorValue = analogRead(sensorPin);
  for (k=0;k<10; k++){
    values[k]     = values[k+1]; }
    values[9]     = sensorValue;
    resistivite = 0;

for (k=0; k<10; k++) {
resistivite = resistivite + values [k]; }

 Serial.println(resistivite);
```

Explication : on définit la valeur de la variable "sensorValue" comme étant ce que l'on lit sur la broche analogique où sensorPin est affectée (donc ici A0).
On crée une fonction for, où la variable "k" est initialisée à la valeur 0, et où si elle est inférieure à 10, elle est incrémentée (augmentée) de 1 en 1.
On y stocke dans le tableau "values" les valeurs de la variable "k", chaque cellule du tableau prenant la valeur suivante de "k".
La dernière cellule du tableau, la cellule 9, prendra la valeur de la variable "sensorValue, donc de ce que la carte détecte sur la broche A0.
On initialise la variable "résistivité" à 0.
On relance la boucle for pour déterminer la valeur de la variable "résistivité".
La variable "résistivité" aura comme valeur "résistivité" + la valeur présente dans les cellules du tableau "values".
On affiche dans le moniteur série la valeur de la variable "résistivité". Plus la valeur est basse, plus l'objet est conducteur. Lorsqu'on fait toucher les 2 pinces crocos, le courant passe normalement, il n'y a pas de résistance la valeur de la variable "résistivité" est à 0.
Dans mon cas, lorsque rien ne touche les pinces crocos, donc seul l'air ambiant fait passer l'électricité entre les 2 pinces, le moniteur série affiche une valeur d'environ 10 200.
Si je touche les 2 pinces crocos, donc seul mon corps sert de conducteur ; "résistivité" est à environ 3600, pour d'autres personnes l'ayant testée, cela varie entre 2200 et 3700.
Dans le cas d'un dessin fait à la mine de crayon gris (graphite) la valeur de "résistivité" est un peu moins de 5000, une banane également testée tourne entre 5000 et 6000. Il faudra donc peut être modifier la valeur dans la condition if en fonction de l'objet à « cliquer ».

2) Pour la led et le servomoteur

```
if (resistivite < 5000){
    digitalWrite(ledPin, HIGH);
    }

  if (resistivite < 5000){
    monservo.write(90);
    delay(500);
    monservo.write(0);
    }

  else{
digitalWrite(ledPin, LOW);
    }
```

Explication : si la valeur de la variable "résistivité" est inférieure à 5000 (cette valeur peut être changée pour améliorer la sensibilité), alors on allume la led.
De même, si la valeur de la variable "résistive" est inférieure à 5000, on positionne le servomoteur à l'angle choisit ici 90°.
On crée un délai d'une demi seconde avant de refaire la boucle. Attention durant ce délai la led reste allumée.
On repositionne le servomoteur à l'angle choisit ici à 0°.
Sinon, c'est à dire que la valeur de la variable "résistive" est supérieure à 5000, on éteint la led. Pas besoin de commande pour le servomoteur, puisqu'il est déjà repositionné.

RÉALISATION D'UNE MAKEY MAKEY À PARTIR D'UNE ARDUINO LEONARDO

Pour ce cas de figure, il nous faut donc une carte Arduino Leonardo qui pourra émuler les touches du clavier et la souris. Pour rappel également seules 6 touches peuvent être émulées puisque la carte ne possède que 6 entrées analogiques.

Etape 1 : le montage (visualisation sur Fritzing)

Le montage est identique au précédent, si ce n'est qu'il ne se compose que de 2 montages distincts, le premier pour la Makey Makey (objet conducteur à toucher) que l'on reproduit 6 fois pour pouvoir connecter 6 objets conducteurs différents, le second (optionnel) pour des leds témoins :
• pour la ***Makey Makey***, même câblage. Pour rappel, le ground de

l'Arduino doit être branché à une pince croco et tenu à la main. La broche 5V, doit être connectée à une résistance de 1M , qui elle-même est branchée à la fois à la broche A0 de l'Arduino et à une pince croco pour le relier à l'objet conducteur. L'opération étant répétée pour brancher les différents objets sur les 6 broches analogiques de la Leonardo, allant de A0 à A5.

- pour les ***leds témoins,*** on branche chaque Led d'un côté au ground, via une résistance, ici de 220 , et de l'autre à une broche digitale de l'Arduino, par exemple de la PIN 8 à la PIN 13. Ce montage est toujours optionnel, mais permet à la fois de visualiser lorsque le courant passe, et en y affectant des leds de couleurs différentes, de pouvoir s'y retrouver plus facilement.

Etape 2 : Principe de base de code

Le principe de base de ce montage reste le même, mais les réglages doivent cependant être plus fins, car le taux de résistivité fluctue sans cesse, et la carte fait de nombreux relevés lors de la phase d'appui sur l'objet conducteur.

En d'autres termes, durant le temps que l'on passe à appuyer sur notre banane, même s'il ne dure qu'une seconde, la carte fera des dizaines de relevés de la résistivité.

Cela n'est pas gênant en soi, lorsqu'il s'agit de faire tourner un servomoteur comme dans notre premier montage, puisque la fonction « delay » liée au servomoteur empêche toute autre action. Le servomoteur tournant pour atteindre l'angle voulu et restant en position pendant toute la durée du « delay ». Aucune autre action ne peut être entreprise.

Mais lorsqu'il s'agit d'émuler une touche du clavier ou un clic cela peut s'avérer problématique. En effet, nous ne pouvons pas associer de « delay » à une touche pressée, car celle-ci exécuterait un nombre conséquent de « pressions » durant toute la durée du delay. Par exemple, si en appuyant sur une banane cela représente la pression de la touche espace ; associé un delay de 1 seconde reviendrait à appuyer sur cette touche pendant une seconde, ce qui créerai un nombre conséquent d'espaces.

De même, une autre problématique entre en ligne de compte : une fois l'appui sur la banane effectué, en relâchant celle-ci, la résistivité ne remonte pas directement à son niveau ambiant (par exemple à 10230 pour mon cas), ce qui peut également être considéré par l'Arduino comme un nouvel appui de la touche espace.

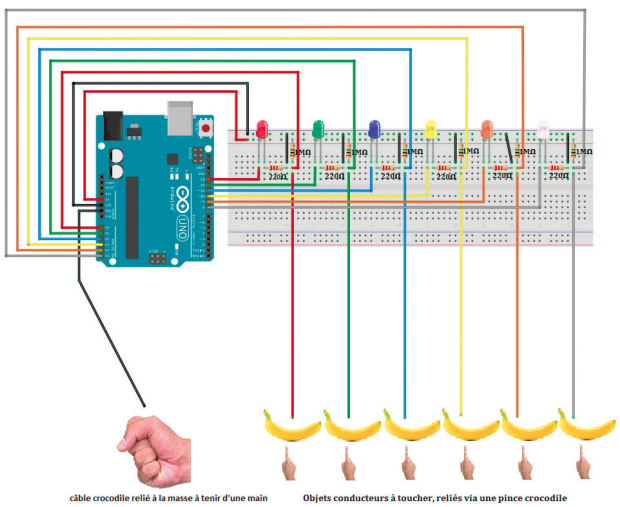
Il est donc important d'une part de rajouter une temporisation dans le loop, lorsque l'Arduino calcule la résistivité. Dans mon cas j'ai choisi un lapse de temps de 1500 millisecondes, afin de ne pas avoir des relevés en cascade, mais uniquement tous les 1500 millisecondes.

D'autre part, il vous faudra choisir un seuil de résistivité assez proche de la résistivité ambiante. En effet, l'appui sur notre objet conducteur doit être comme celle d'une touche : rapide et non maintenue, pour que lorsqu'on appuie sur notre banane la résistivité passe directement en dessous du seuil choisi et qu'elle remonte immédiatement après relâchement pour atteindre sa valeur par défaut qui est celle de l'air ambiant.

Pour affiner ces réglages il vous faudra vous aider du moniteur série du logiciel Arduino IDE, pour connaitre d'une part la résistivité ambiante, donc celle qui s'affichera lorsque vous ne cliquez pas sur votre objet, mais également connaître la résistivité rencontrée lorsque vous touchez l'objet conducteur, afin de définir dans votre code le seuil optimal.

Attention ! Cette opération peut s'avérer fastidieuse, puisque la résistivité ambiante dépend de nombreux paramètres (humidité, température ambiante …). De même la résistivité obtenue lorsqu'une personne clique sur l'objet conducteur est également variable d'une part en fonction de la personne, mais également de l'objet conducteur. Et dans ce domaine une banane n'est pas équivalente à une autre !

Entrons maintenant dans le code.



câble crocodile relié à la masse à tenir d'une main — Objets conducteurs à toucher, reliés via une pince crocodile

Etape 3 : Déclaration des variables et des constantes

1) Pour la fonction Makey Makey

Voir code complet sur www.programmez.com

Explication : aucun changement par rapport à notre précédent montage, nous utilisons le même principe que nous déclinons en 5 fois, puisque la makey makey possède 5 fonctions. Pour plus de facilité chaque objet conducteur est associée à une led de couleur différente.

Pour rappel, nous créons des variables de type int, que l'on nomme sensor_Pin suivies de la couleur de la led et que l'on associe à une des entrées analogiques de la carte Arduino. Ainsi l'objet branché à la PIN A0 sera relié à la led rouge ; celui sur la PIN A1, à la led verte ; la A2 à la led Bleue ; la A3 à la led Jaune ; la A4 à la led Orange et finalement la A5 à la led Blanche.

Sur la même idée nous créons également des variables de type int nommées sensorValue suivies également de la couleur de la led, que l'on initialise à 0, qui nous permettront d'y stocker la valeur que la carte détectera.

S'ensuit pour chaque objet la création d'un tableau de 10 cellules pour y stocker les variables, et la création de variables de type "int" nous permettant de calculer la "résistivité" et dont les noms seront représentés par des lettres, que j'ai choisies arbitrairement allant de k à p.

On finit par la création de variable de type "int" nommée "résistivité et le nom de la couleur" qui symbolise en quelque sorte la résistivité électrique. Plus elle sera élevée, moins le courant passera.

2) Pour les leds

```
int ledPin_Rouge = 13;
int ledPin_Verte = 12;
int ledPin_Bleue = 11;
int ledPin_Jaune = 10;
int ledPin_Orange = 9;
int ledPin_Blanche = 8;
```

Explication : on crée une suite de variables de type "int" nommée "ledpin suivie de la couleur de la led" et affectée à l'une des broches de l'Arduino, pour cela on utilisera les PIN 8 à 13 de la carte.

3) Pour le clavier et la souris

```
#include <Keyboard.h>
#include <Mouse.h>
```

Explication : afin d'émuler un clavier et la souris nous faisons appel à deux bibliothèques.

Etape 4 : Void Setup

1) Pour la fonction makey

```
Serial.begin(9600);
```

Explication : on définit le taux de transfert de données à 9600 bauds.

2) Pour les leds

```
pinMode(ledPin_Rouge, OUTPUT);
pinMode(ledPin_Verte, OUTPUT);
pinMode(ledPin_Bleue, OUTPUT);
pinMode(ledPin_Jaune, OUTPUT);
pinMode(ledPin_Orange, OUTPUT);
pinMode(ledPin_Blanche, OUTPUT);
```

Explication : on définit les broches connectées à la variable "ledPin suivie de la couleur de la led" comme étant des sorties.

3) Pour le clavier et la souris

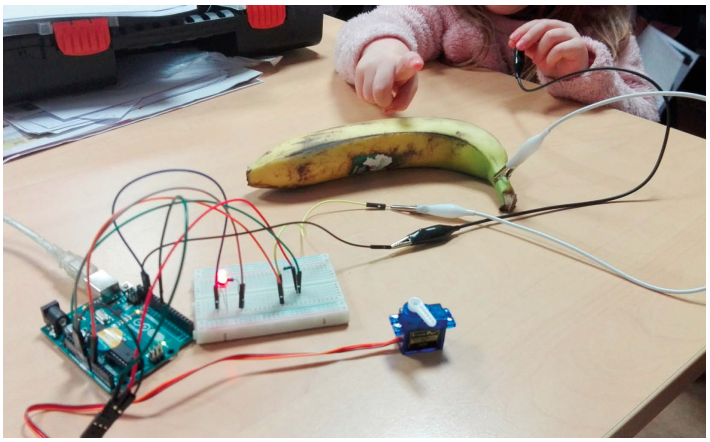
```
Mouse.begin();
 Keyboard.begin();
}
```

Explication : on initialise la communication avec le clavier et la souris. Et on n'oublie pas de fermer l'accolade du void setup.
https://www.arduino.cc/en/Reference/KeyboardModifiers
https://www.arduino.cc/en/Reference/MouseClick

Etape 5 : Void Loop

1) Pour la fonction makey

Voir code complet sur www.programmez.com
Explication : Encore une fois on se retrouve avec 6 blocs, chacun représentant un objet connecté symbolisé par une led de couleur différente. Dans chaque bloc, on définit comme pour le premier code les "sensorValue" et des variables nommées par des lettres allant par exemple de K à p pour nous permettre de calculer la "résistivité".



On affiche dans le moniteur série la valeur de la variable "résistivité". Plus la valeur est basse, plus l'objet sera conducteur. Pour rappel, lorsqu'on fait toucher les 2 pinces crocos, le courant passe normalement, il n'y a pas de résistance ; la valeur de la variable "résistivité" est à 0.
Dans mon cas, l'air ambiant donne une valeur d'environ 10 200 à 10 300, un corps humain varie entre 2200 et 3700, de la mine de crayon gris (graphite) est à un peu moins de 5000, et une banane tournera entre 5000 et 6000. Il faudra donc peut être modifier la valeur dans la condition if en fonction de l'objet à « cliquer ».
Finalement on temporise par une fonction « delay » ici à 1500 millisecondes, mais dont vous devrez certainement modifier la valeur en fonction de vos besoins. Pour rappel, ce lapse de temps est nécessaire pour éviter que l'Arduino ne fasse une cascade de relevés et n'enchaîne ce qu'elle considérera comme de multiples appuis.

2) Pour les leds témoins

Voir code complet sur www.programmez.com
Explication : si la valeur de la variable "résistivité" liée à chaque couleur est inférieure à 5000 (cette valeur peut être changée pour améliorer la sensibilité), alors on allume la led de couleur correspondante.
Sinon, c'est à dire que la valeur de la variable "resistive" est supérieure à 5000, on éteint la led.

3) Pour les touches du clavier et le clic gauche de la souris

Voir code complet sur www.programmez.com
Explication : si la valeur de la variable "résistivité" liée à chaque couleur est inférieure à 5000, alors on fait appel à la commande Keyboard.press(chiffre) ; de la bibliothèque Keyboard pour émuler la touche correspondante au chiffre saisi dans les parenthèses. Par exemple 218 correspond à la flèche du haut, alors que le 178 correspond à la barre espace. On utilise ensuite la commande Keyboard.releaseAll() ; pour simuler la fin l'émulation de toutes les touches.
De même pour émuler le clic gauche de la souris on utilise la commande Mouse.click() ; de la bibliothèque Mouse, que l'on arrête par la commande Mouse.release() ;
Et on n'oublie pas de fermer l'accolade du loop.

4) Et si on émulait d'autres touches ?

Le principe reste le même il suffit juste de choisir le chiffre correspondant à la touche. Par exemple :

```
128 pour Ctrl gauche
129 pour Shift gauche
130 pour Alt gauche
131 pour Touche Windows
132 pour Ctrl droit
133 pour Shift droit    etc.
```

Adapter, modifier et vous l'approprier !

Ce projet est une base améliorable en fonction de vos besoins et de vos envies. Vous pourrez vous en inspirer et les modifier.

D'autres exemples sur mon site https://ardwinner.jimdo.com/arduino/ •

Le **Robot à fabriquer** et à « Programmez » soi-même : Poppyrate !

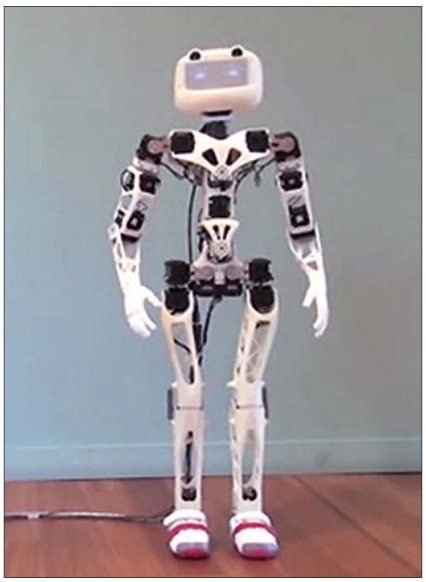
• Aymeric Weinbach
@aymericw et Paul Mugnier

Ils tondent nos pelouses et aspirent parfois la poussière de nos chambres. Ces petits automates qui autrefois n'appartenaient qu'à la fiction ou rampaient au fond de nos piscines se dressent maintenant sur leurs deux jambes. Les robots marchent et communiquent plus ou moins bien. Si bien qu'ils apparaissent petit à petit dans notre quotidien : nous les rencontrons parfois dans les rayons de nos supermarchés ou dans nos usines. Habitués à leur seule présence dans les bandes dessinées et les films, une défiance à leur égard s'est installée. D'autant plus que leur arrivée a vite été perçue comme une menace pour l'emploi … Dommage ! La robotique est peut-être le domaine le plus rassembleur et le plus porteur de notre époque. A condition de le rendre accessible à tout le monde ! Designers, artistes, ingénieurs, infirmières, informaticiens, psychologues, sauveteurs, etc.

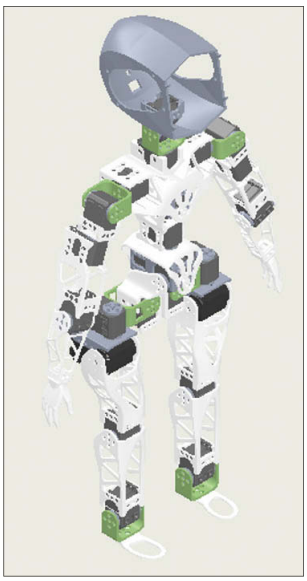
Alors que l'on entend que la technologie divise la société, toutes les professions sont ainsi appelées à se rencontrer pour fabriquer ou exploiter cet outil fantastique. C'est donc dans ce souci d'accessibilité, d'apprentissage et de diversité qu'est née notre initiative : permettre à chaque personne souhaitant apprendre la robotique de lui donner un outil modulable dans l'accomplissement d'un projet personnel ou professionnel. Ce projet était ambitieux par rapport à nos capacités mais grâce à la découverte du projet Poppy, sur lequel nous nous sommes appuyé nous avons réussi à offrir un robot d'éducation peu onéreux.

De Poppy à Poppyrate

Le projet Poppy humanoïde est un projet de recherche de l'INRIA, dont la particularité majeure est d'avoir un « soft » et un « hard » open source. Il est aussi imprimable en 3D et ses composants se trouvent sur « étagère » sauf quelques cartes électroniques faites sur mesure. Cette volonté de démocratisation fut donc presque atteinte. Néanmoins l'ensemble des pièces pour constituer un Poppy coûte cher. On peut acheter soi-même ses pièces ou acheter un kit pour un prix avoisinant les 9000 €. Un coût qui ne le met qu'à la portée des laboratoires de recherche. Le projet Poppyrate vise donc à créer un Poppy à un tarif abordable pour rendre accessible le robot humanoïde à un public beaucoup plus large.



Robot Open-Source Poppy de l'INRIA



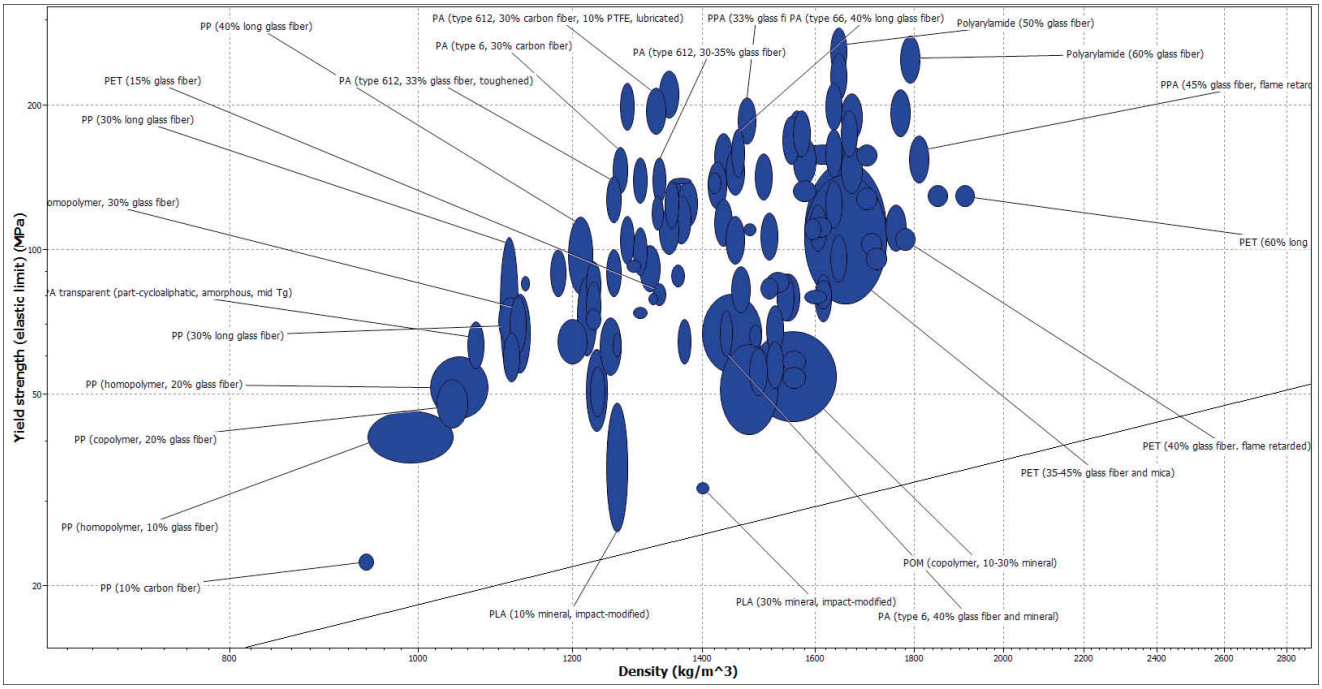
Modèle Solidworks de Poppyrate

Avec Paul Mugnier, élève-ingénieur en mécanique, nous nous sommes attaqués à ce projet. Le coût élevé du Poppy provient essentiellement de ses moteurs. Ils sont au nombre de 25 et sont des servomoteurs haut de gamme très couteux : les AX32 et AX64 de Dynamixel. En décidant de réduire le gabarit du Poppy et en mettant en compétition plusieurs architectures, une configuration optimale pour le Poppyrate se détache: 2 Dynamixel AX-12A – 23 Dynamixel XL-320. En s'appuyant sur toutes les informations disponibles sur le Poppy, nous avons dessiné un nouveau squelette accueillant de nouveaux servomoteurs alimentés par un nouveau schéma électronique. Ce dernier reste très proche du schéma original du Poppy à la différence qu'il comporte moins de composants importés des USA (Sparkfun, Adafruit, …) et possède 3 hacheurs de plus pour adapter le courant et la tension entre les AX-12A (placés dans les hanches), les XL-320, les cartes électroniques et la prise secteur.

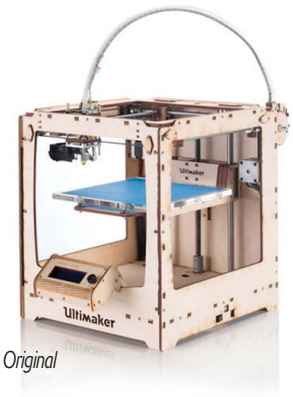
Avec comme base de départ les fichiers Solidworks nous avons redessiné un nouveau squelette pour notre Poppyrate. Ce dernier a été dimensionné et réduit par rapport à celui de Poppy pour s'adapter à de nouveaux servomoteurs (AX-12 + XL-320) plus petits, moins puissants et à leurs nouveaux points d'ancrage. Outre son adaptabilité aux servomoteurs, le squelette a également été modifié pour être plus facile à imprimer. Le squelette de Poppy s'imprime assez difficilement sur une imprimante FDM classique comme l'Ultimaker, la Replicator ou n'importe quelle RepRap. Ainsi les pièces ont besoin d'une multitude de supports qui rendent l'impression plus longue et le résultat pas évident à nettoyer par la suite. Toutes les pièces ont ainsi été modifiées pour être simples à imprimer et quelques supports indispensables ont été ajoutés directement dans les fichiers 3D ce qui permet d'avoir des supports plus simples et plus facilement destructibles par la suite.

Premières impressions 3D du squelette

Les premiers tests ont eu lieu sur mon imprimante 3D, une « Ultimaker Original », premier modèle d'Ultimaker en bois distribuée en kit à monter soi-même et équipée d'une tête d'im-



Contrainte à la rupture de matériaux 3D en fonction de leur densité



Utlimaker Original

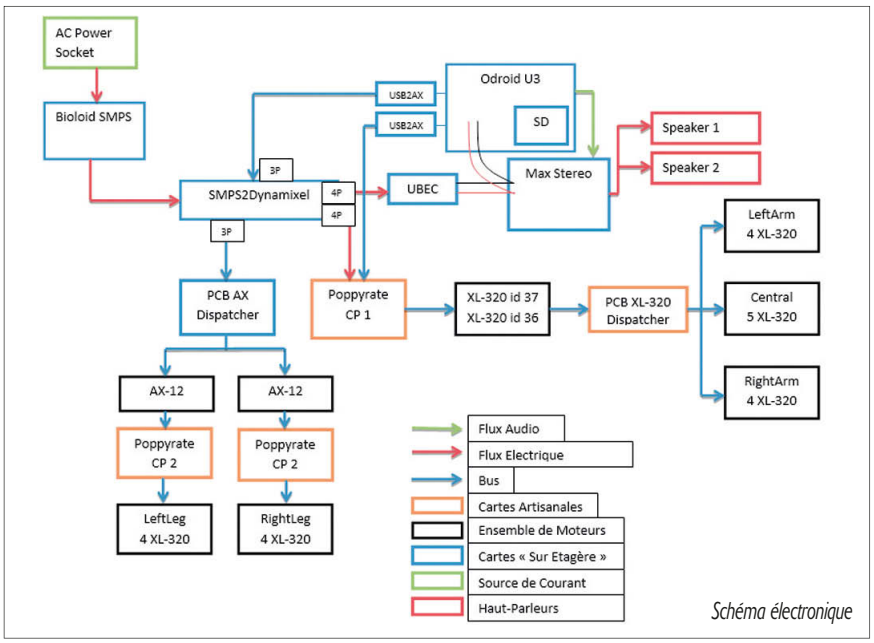


Schéma électronique

pression E3D permettant l'impression de plusieurs autres matériaux.

Les premières impressions ont surtout servi à ajuster les dessins des pièces par rapport aux servos et toutes leurs problématiques associées (dimensions, connectiques, points d'ancrage …). Elles ont été réalisées en PLA blanc avec une épaisseur de couche de 0.2 mm et une vitesse d'impression privilégiée à la qualité lors de ces tests.

Comme le PLA n'est pas un matériau très solide de les pièces sensibles se devaient d'être imprimées dans une matière plus solide. Après des courtes études de résistance des matériaux, trois critères de sélections se sont détachés : Solidité à la Flexion (Contrainte à la rupture/Densité), Rigidité à Flexion (Module de Young/Densité) et Traction/Compression. De nombreux matériaux mêlant polymères et fibres de verre ou de carbone sont apparus mais le fuseau des recherches s'est énormément réduit lorsque nous nous sommes renseignés sur leur « Imprimabilité 3D ».

Le choix s'est alors arrêté sur le filament XT CF 20, polymère développé par le néerlandais ColorFabb renforcé avec 20% de fibre de carbone, qui rend le filament particulièrement abrasif et qui a pour effet d'user prématurément les buses d'imprimante 3D. Pour éviter tout problème une buse d'impression en acier inoxydable a donc été ajoutée. Comparé au PLA, Le XT-CF20 possède une densité 14% supérieure mais une rigidité 3 fois plus importante et un gain de solidité de 50%. Ces bonnes qualités mécaniques nous permettent de rentrer dans un cercle vertueux : en réduisant le volume du squelette pour les mêmes standards mécaniques nous réduisons sa masse globale, augmentons sa taille et limitons l'impact du prix du matériau sur le prix global.

Un plateau chauffant à également été acheté pour rendre ces impressions plus faciles, mais les premiers tests réalisés la semaine dernière sur le XT CF 20 sans plateau chauffant ont montrés qu'il n'y avait aucun problème de déformation sur les petites pièces.

Maintenant sur les impressions 3D c'est à vous de jouer ! Le projet étant opensource si vous voulez imprimer votre propre Poppyrate envoyez-nous un email à contact@poppyrate.com pour

avoir un accès en avant-première aux fichiers d'impression.

Une Source d'Energie Adaptée

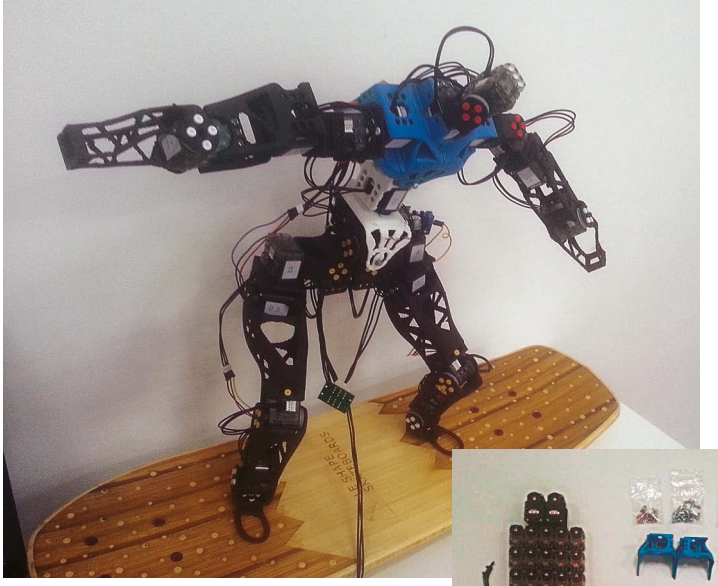
Il a fallu adapter l'électronique de puissance du Poppy à celle réclamée par Poppyrate. Les moteurs XL320 et les AX12 n'ayant pas la même puissance, pour le projet ont donc été développées des cartes faites maison, mais malheureusement il n'est pas toujours évident de faire ses propres cartes et les cartes développées étaient vraiment peu fiables. Mais depuis cette première version, l'équipe du projet Poppy a développé une carte qui fait exactement ce que faisait cette carte «custom ». La carte PIXL permet d'alimenter les XL320 et de transmettre la data à un Raspberry pi. Et elle est en cours d'intégration dans la nouvelle version de Poppyrate. Pour l'intelligence embarquée, Poppy dans sa version originale qui a servi pour Poppyrate tournait avec une carte Odroid U3. Cette carte avait été choisie car à taille équivalente elle est plus puissante qu'une Raspberry Pi, et pour Poppy elle est placée dans la tête pour plus d'anthropomorphisme. Pour Poppyrate à l'origine nous avons utilisé la même carte, mais Poppyrate étant beaucoup plus petit, la carte ne peut tenir dans la tête. Dans la version en cours on teste donc l'intégration d'une Raspberry pi zero, plus petite mais aussi moins puissante.

Le montage

Pour l'impression 3d nous avons utilisé CURA comme «slicer » logiciel OpenSource fourni par Ultimaker. Une fois que vous avez toutes les pièces imprimées en 3D ainsi que l'électronique nécessaire vous êtes prêt pour monter votre premier Poppyrate. Le montage est assez intuitif, vous aurez juste besoin d'un tournevis ainsi que d'un OLLO Tool fourni avec les rivets du même nom produit par le cosntructeur des XL 320 Et ensuite dernier conseil brancher la connectique après le montage des moteurs avec le squelette. Un guide de montage détaillé est en cours de réalisation.

La programmation du robot

Le projet Poppy, c'est aussi une contribution open source logicielle avec Pypot. Pypot est un framework pour python 2.7 qui va permettre de programmer aisément les moteurs Dynamixel, ainsi que de faire différents niveaux d'abstraction sur des ensembles de moteurs pour en faire des « creatures » programmables. Pour Poppyrate nous avons recréé une structure Pypot pour pouvoir aisément programmer Poppyrate. Pypot



Une application possible : Poppyrate qui fait du skate !

Ensemble des pièces nécessaires pour monter un Poppyrate (hors électronique)

nécessite les packages scientifiques classiques scipy et numpy et est disponible via le package manager Pip ou easy_install. Pypot est aussi compatible avec le simulateur V-REP ce qui permet de programmer et de tester le comportement de son code sur un robot virtuel afin de garantir la sécurité du robot et de visualiser rapidement la cinématique de votre application. Pour Poppyrate le modèle V-REP n'existe pas encore mais est en cours de réalisation.

Hello World Poppyrate

```
Import pypot
```

Pour importer la librairie Pypot

```
poppyrate = pypot.robot.from_json('configuration\
poppyrate_config.json')
```

Pypot permet de décrire un robot avec un fichier json. Dans ce fichier json on décrit 3 choses :

- Les controllers qui correspondent aux bus de moteurs en série;
- Les moteurs pour décrire leur possibilité de mouvement (un genou par exemple ne peut s'ouvrir que sur un certain angle);
- Les abstractions « groupes de moteurs », par exemple un bras.

Cette ligne permettra de charger la config correspondant à Poppyrate.

```
for m in poppyrate.motors:
    m.compliant = False
    m.goal_position = 0
```

Met tous les moteurs en position initiale (tel que défini dans le json de config)

```
poppy.r_shoulder_y.goto_position(-90, 2, wait=False)
poppy.r_shoulder_x.goto_position(15, 2, wait=False)
poppy.l_shoulder_y.goto_position(-90, 2, wait=False)
poppy.l_shoulder_x.goto_position(-15, 2, wait=True)
```

Poppyrate va lever les bras

Applications et modularité

En fonction des applications que vous souhaitez faire vous pouvez embarquer tout type d'équipement à bord de la tête comme une caméra ou des capteurs infra-rouge. Notre objectif est quasiment atteint car nous obtenons un prix global de 1500€, il reste maintenant à lui faire explorer tous les horizons possibles.
En ce qui nous concerne pour la partie matérielle nous y intégrons actuellement une caméra et des haut-parleurs avec un micro.

Rendre « intelligent » le robot avec le Cloud

Pour le côté logiciel le but est actuellement de rendre « intelligent » le robot en utilisant les « Cognitive Services » de Microsoft Azure.
La caméra avec l'utilisation des services de vision permettra la reconnaissance de visages et des émotions. Le micro et l'utilisation des services de synthèse vocale et de reconnaissance vocale des cognitive services vont permettre une commande vocale. Le but ultime étant de réaliser un robot social avec le Cloud.

Un projet ouvert à vos contributions

Comme le projet Poppy, Poppyrate a l'esprit à être un projet open-source. Si vous voulez rejoindre l'aventure, si un des sujets dont parle l'article vous motive, rejoignez l'aventure Poppyrate: contactez-nous : contact@poppyrate.com

Liens

Pour Poppyrate : https://www.poppyrate.com contactez nous directement pour un accès au repository beta contact@poppyrate.com
Pour Pypot https://github.com/poppy-project/pypot et http://poppy-project.github.io/pypot/
Pour Poppy https://www.poppy-project.org/en/ et https://github.com/poppy-project/poppy-humanoid

USAGE AVANCÉ

Une utilisation avancée du robot serait de le rendre « intelligent » avec des services cloud d'IA tels que les Cognitive Services de Microsoft. Pour faire la suite de cet exemple vous devez vous inscrire préalablement et obtenir des clés d'API pour cognitive services sur le portail Azure https://azure.microsoft.com/fr-fr/try/cognitive-services/ Il y a un crédit gratuit pour un certain nombre d'appel à l'API. Comme pour le reste du code du robot les exemples sont en Python 2.7

```
import httplib, urllib, base64

headers = {
  # Request headers.
  'Content-Type': 'application/json',

  # Mettez Ici la clef  d'api récupéré
  'Ocp-Apim-Subscription-Key': ######',
}
```

Pour que vous puissiez faire l'exemple aussi sans le robot vous pouvez utiliser des images stockées sur Internet

```
# Mettez ici l'url vers la photo que vous avez prise avec la caméra
body = "{'url':'...'}
#Pensez à mettre le point d'api correspondant à la clef d'API pour laquelle vous vous etes enregistré

try:
  conn = httplib.HTTPSConnection('westeurope.api.cognitive.microsoft.com')
  conn.request("POST", "/vision/v1.0/analyze?%s" % params, body, headers)
  response = conn.getresponse()
  data = response.read()
  print(data)
  conn.close()
except Exception as e:
  print("[Errno {0}] {1}".format(e.errno, e.strerror))
```

En retour vous allez récupérer un JSON qui ressemblera à ça :

```
{
 "categories": [
  {
   "name": "abstract_",
   "score": 0.00390625
  },
  {
   "name": "people_",
   "score": 0.83984375,
   "detail": {
    "celebrities": [
     {
      "name": "Satya Nadella",
      "faceRectangle": {
       "left": 597,
       "top": 162,
       "width": 248,
       "height": 248
      },
      "confidence": 0.999028444
     }
    ]
   }
  }
 ],
```

voir code complet sur www.programmez.com

Vous pouvez ainsi récupérer beaucoup d'infos sur l'environnement direct grâce à l'analyse des photos prises par la caméra. Il est aussi possible avec « cognitive service » de restreindre la reconnaissance à des « Domain Specific Model » , de faire de la reconnaissance d'émotions sur des visages afin de faire réagir le robot différemment en fonction des émotions de l'utilisateur. Ou d'aller plus loin de faire vos propres jeux de données privé. Il est aussi possible d'utiliser « Cognitive Services « pour la reconnaissance de caractères, vous pouvez ainsi apprendre à «lire » au robot :

```
import httplib, urllib, base64

headers = {
  # Request headers.
  'Content-Type': 'application/json',

    # Mettez Ici la clef  d'api récupéré
  'Ocp-Apim-Subscription-Key': '13hc77781f7e4b19b5fcdd72a8df7156',
}

params = urllib.urlencode({
  # Vous allez mettre ici des paramètres correspondant à la langue ou à l'orientation en mettant unk il détectera automatiquement la lanque de l'utilisateur ainsi que l'orientation de la photo avec detectorientation à true
  'language': 'unk',
  'detectOrientation ': 'true',
})

# Ici mettez l'url vers la photo prise
body = "{'url':'...'}"

try:
#Pensez à mettre le point d'api correspondant à la clef d'API pour laquelle vous vous etes enregistré
  conn = httplib.HTTPSConnection('westeurope.api.cognitive.microsoft.com')
  conn.request("POST", "/vision/v1.0/ocr?%s" % params, body, headers)
  response = conn.getresponse()
  data = response.read()
  print(data)
  conn.close()
except Exception as e:
  print("[Errno {0}] {1}".format(e.errno, e.strerror))
```

De la même façon vous allez récuperer un json correspondant :
voir code complet sur www.programmez.com

Cet exemple fonctionnera avec des textes imprimés mais il est aussi possible de faire de la reconnaissance d'écriture manuscrite. •

Pourquoi Xamarin ?

Maxime EGLEM
Développeur Xamarin
Chez **Cellenza**

cellenza
DOESITBETTER | Conseil - Expertise Microsoft & méthodes agiles

Dans la mesure où les technologies mobiles sont susceptibles de transformer profondément les mœurs pour donner naissance à tout moment à des formes d'organisations nouvelles, il est nécessaire pour Cellenza de partager notre expérience concernant la décision prise depuis plus de 4 ans dans ce secteur.

Faire des applications mobiles n'a jamais été facile. Il y a une telle diversité entre les plateformes et les téléphones qu'il fallait trouver une solution afin d'optimiser les temps de développement et d'intégration sans oublier le coût de maintenance, tout en évitant la création de multiples bases de code afin de ne pas perdre en performance. En effet, avec des codes différents il faut écrire chaque évolution plusieurs fois. Cela multiplie aussi le risque d'avoir des bugs au sein des applications.

Chaque langage de programmation est unique même si beaucoup se ressemblent et demandent des compétences dédiées. L'écriture de code dans différents langages demande des personnes qualifiées sur chaque langage. Plus le nombre d'intervenants et de langages augmente, plus l'organisation du projet devient difficile, plus il faut maintenir une bonne cohérence de l'équipe. Tout ceci a un coût non négligeable.

Les sociétés comme Apple, Google et Microsoft souhaitent se différencier en trouvant un moyen de rendre leurs logiciels ou produits incontournables. Pour ce faire, chaque société investit pour créer un produit unique. Apple travaille depuis 2007 sur des produits fonctionnant avec iOS en Objective-C et Swift quelques années plus tard. Google décide de contrer Apple avec Android se basant sur du Java, et maintenant Kotlin. Et Microsoft décide aussi de se battre pour proposer d'autres produits sous Windows 10 en .Net. Trois grandes entreprises avec des savoir-faire et des visions différentes. Certaines plateformes s'effondrent tandis que d'autres émergent. Cette différence de vision influe donc sur les sociétés souhaitant promouvoir leurs applications sur ces multiples plateformes. Les logiciels permettant de développer des applications sur Android, iOS ou Windows 10 sont complètement différents les uns des autres et demandent beaucoup plus de ressources.

Le tournant

En 2011 beaucoup de changements ont eu lieu. Les créateurs de Mono décident alors de créer très rapidement une startup, Xamarin, permettant de résoudre les critères cités. Mais ce ne sont pas les seuls. La société à but non lucratif Apache Software Foundation décide elle aussi de trouver une idée pour combler les problématiques connues en créant Apache Cordova.

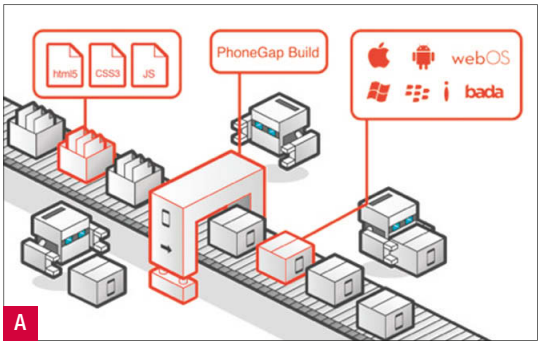
Deux solutions permettant de résoudre certaines problématiques, mais basées sur des idées complètement différentes. Les solutions Open-Source ont toujours été appréciées par la communauté des développeurs. Et Cordova devient vite plus connu et accessible que son concurrent Xamarin. Pourquoi alors avons-nous choisi de mettre nos ressources au sein de Xamarin, une solution un peu moins connue à l'époque et de plus payante ? Quel type de projets nous a permis de nous démarquer ? Comment savions-nous que cette technologie allait être rachetée par Microsoft ? Je vous rassure nous ne le savions pas, mais certains aspects on fait pencher la balance.

L'approche hybride

Le principe de Cordova est d'opter pour des technologies Web déjà connues pour fonctionner sur de multiples plateformes. C'est ce qui permet de créer des applications Hybrides. Cette solution est l'une des premières vraies réponses apportées pour répondre à nos problématiques. Une application hybride est donc une application basée sur du HTML adaptatif, CSS et JavaScript. Chaque plateforme native fournit un composant permettant d'afficher un navigateur Web au sein de l'application appelée WebView. Cette Webview est intégrée dans un conteneur permettant d'accéder aux API natives. L'appellation Hybride est due à l'utilisation de scripts Web hébergés dans une application. [A]

C'est donc un mélange entre un site Web et une application native. Le développement hybride permet de fortement rationaliser les coûts de développement de la couche métier et surtout de la couche graphique. Une WebView, via le conteneur, peut avoir accès aux API natives de la plateforme, mais ne permet pas d'accéder aux composants graphiques natifs de cette dernière. Ce qui veut dire que l'utilisateur peut se retrouver perdu en utilisant une application qui ne possède pas les normes graphiques mises en œuvre par les constructeurs. Le fait d'avoir un conteneur pour afficher du code non natif rend l'application moins performante de manière notable pour l'utilisateur. De plus, l'avantage d'utiliser les composants graphiques natifs est d'avoir une application continuellement mise à jour automatiquement. Une des mises à jour graphique marquante fut par exemple la sortie d'iOS 7 par Apple.

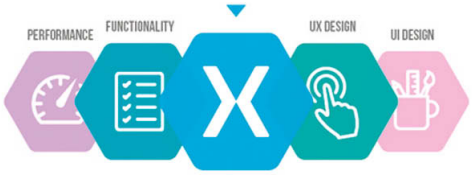
L'approche hybride n'est pas une solution miracle et possède de nombreuses problématiques. Même si parfois, elle est intéressante financièrement à très court terme. Cela peut par exemple sembler justifié de se concentrer sur cette solution pour de petites applications, mais reste délicat pour posséder une application performante avec un code propre et maintenable.



A

L'approche Xamarin

La mission de Xamarin est de rendre le développement mobile rapide, facile, en ayant un langage de programmation moderne et en utilisant des outils puissants tout en créant une expérience utilisateur parfaite.



Le développeur commence par créer une base de code commune. Elle contient notamment la logique métier, le stockage en base de données, les appels réseaux, les éléments d'interface communs. Ce projet peut être facilement encadré par des tests unitaires car son code est indépendant de tout système spécifique. Ensuite, un projet est créé par plateforme cible. Il contient l'interface graphique, la navigation et les composants propres à chaque SDK. Ainsi, on peut tirer parti des spécificités propres à Android ou iOS sans réduire l'expérience utilisateur.

Xamarin est un Framework basé sur le Framework .NET permettant de créer deux grands types d'applications. Xamarin Natif, une solution native bénéficiant de toute la puissance offerte par la plateforme et Xamarin Forms, une solution dont l'interface est partagée un peu comme du Cordova, mais sans les problématiques de la WebView.

Avec le support des PCL (Portable Class Librairies) ou des Shared Projects, Xamarin va un cran plus loin dans la réutilisation de code. Ce code commun regroupe une interface commune puis des implémentations propres à chaque plateforme.

Pouvoir partager le code métier ainsi que les tests unitaires en utilisant toute la puissance native de la plateforme est un énorme gain de temps. En février 2016, Microsoft confirme notre choix et investissement sur ce Framework en annonçant le rachat de Xamarin.

Le choix

Choisir une technologie peut avoir un impact important dans des entreprises de services. Après plusieurs années à créer tous types d'applications B2B, B2C et B2E nous nous sommes forgé une expérience qui nous permet de partager notre savoir-faire. Chaque projet est différent et possède des problématiques propres. Il est un devoir de bien savoir conseiller le client sur la solution Xamarin à utiliser. Prendre une solution qui nous permet de créer une application native n'enlève pas les problématiques de performance et de mémoire que les appareils des utilisateurs peuvent avoir. Savoir comment fonctionne chaque système d'exploitation et comment éviter ces difficultés est important pour créer des applications performantes.

Créer une application prend du temps, nous l'avons bien compris. Mais ne pas savoir la gérer une fois publiée en fait perdre encore plus. Xamarin a tout d'abord développé des outils, comme Xamarin Insight et Xamarin Test Cloud, permettant en outre d'avoir un suivi en continu de l'application même lorsque celle-ci se trouve installée chez l'utilisateur. Microsoft après le rachat a continué d'améliorer ces outils et les a intégrés dans d'autres tel que Hockey App et Mobile Center.

CONCLUSION

Xamarin est pour nous une plateforme qui a fait ses preuves et vous comprendrez par la suite pourquoi. Xamarin est une solution en constante évolution permettant de simplifier au mieux la vie des développeurs et ce pour de nombreuses années !

Les annonces de Microsoft chaque année convergent vers un avenir serein. •

Retour sur quelques années de Xamarin

John Thiriet
Technical Manager Mobilité @Cellenza
Microsoft MVP & Xamarin MVP
@johnthiriet

cellenza
DOESITBETTER | Conseil - Expertise Microsoft & méthodes agiles

Lorsque nous étions petits, nous avons tous tenté de construire des châteaux de sable à la plage. Les premiers essais en général non concluants se font réduire en miette parce que construits trop près de la mer. Ou bien le sable n'est pas assez humide pour maintenir la structure en état. Bref au début, on n'arrive à rien. Si on utilise l'analogie des châteaux de sables avec Xamarin, on peut dire une chose, c'est qu'on en a raté un bon nombre avant de pouvoir aujourd'hui vous partager ce qu'il faut faire et ne pas faire. [1]

Rien de tel que d'essayer et d'utiliser longuement un outil, dans des contextes et avec des objectifs variés pour apprendre à le maîtriser. Vous l'aurez donc probablement compris, c'est de notre expérience avec Xamarin que je vais vous parler dans cet article. Nos premiers châteaux de sables datent de 2014 et nous n'étions pas encore familier avec les outils à notre disposition. Nous avions auparavant utilisé les outils habituels et recommandés tels que pelles, râteaux et seaux. Cependant, à chaque fois nous devions construire nos châteaux en double voire triple pour qu'ils soient disponibles à tous les utilisateurs de la plage. C'est donc avec résolution que nous nous sommes activés à l'apprentissage de cet outil qui commençait à vraiment faire parler de lui. Tous les ouvriers en bâtiment plagistes en parlaient, il fallait s'y mettre.

Je vous rassure, je ne me suis pas perdu dans cet article; bien que l'appel du sable se fasse sentir, la plage représente ici le marché mobile et chaque château est une version de l'application pour chaque plateforme.


1

La première étape

Comme beaucoup de développeurs .NET, c'est Xamarin Forms qui fut de prime abord l'objet de toute notre attention, bien que conscients que développer en Xamarin Forms ne soustrait pas à la connaissance des plateformes sous-jacentes.

C'est également en Xamarin Forms qui nous avons réalisé notre premier projet Xamarin, en régie, pour un client. Ce client souhaitait une application Xamarin fonctionnant sur iOS, Android et Windows Phone. Il est assez rare de devoir cibler les trois plateformes mais ça tombe bien, Xamarin est fait pour ! L'autre particularité de ce projet est sa cible, les employés de la société. Il ne devait donc pas être déployé dans les stores. Ce fut donc une expérience riche sur des problématiques liées à Xamarin Forms, à des composants natifs comme la prise de photo, des notifications push, ou la cartographie mais aussi aux problématiques de déploiements en entreprise.

Et voilà, notre premier château est maintenant construit. Mais plutôt que de faire comme tout le monde avec des châteaux grand publics, nous avons construit un château privé et caché à la vue de tous.

La seconde étape

Notre seconde expérience consista en une intervention en régie sur un projet grand public. Le projet était déjà en place et avait démarré et les choix d'architecture était actés, notamment l'utilisation de MvvmCross. Ce projet était développé en majorité par des équipes internes extrêmement compétentes en .NET mais surtout côté Web. La montée en compétence de ces équipes n'a pas été uniforme et certains choix d'architectures, tels que l'utilisation de MvvmCross ont été remis en cause en cours de développement. De fait, l'inexpérience globale de toutes les équipes à l'utilisation de MvvmCross n'a pas non plus aidé. Si on peut retenir une chose de ce projet c'est qu'à trop chercher à contourner un framework tout en cherchant à le maintenir en place on finit par perdre plus de temps qu'a en gagner.

Notre deuxième château est en place, mais la construction de ce dernier a été laborieuse et il a fallu de nombreux coups de pelles pour arriver à le faire tenir debout.

La troisième étape

La suite de notre expérience en Xamarin a été le développement d'une preuve de concept (POC), en régie, en Xamarin iOS natif pour un jeu de type Quizz. A l'opposé de l'application


2

précédente, le design ici était extrêmement riche. Beaucoup d'animations, du son, des timers, etc. Une vraie expérience très riche utilisant énormément d'API natives.

Notre troisième château est maintenant en place, magnifiquement fini et architecturé mais disponible qu'à un seul bout de la plage. [2]

La quatrième étape

Par la suite nous avons mené de front deux projets grand public, à destination des stores, un en régie et un en forfait agile au Studio Cellenza. Les deux ciblent iOS et Android et ont des problématiques similaires.

Le projet en régie, a un très fort besoin de performance, a nécessité l'audit, la reprise et l'optimisation d'un code existant utilisant MvvmCross. La mauvaise expérience précédente avec ce framework s'est encore une fois confirmée. Une grosse partie du travail d'optimisation de l'application consista en la suppression brique par brique de ce dernier pour s'approcher le plus près possible du fonctionnement natif de la plateforme.

Le projet au Studio est quant à lui un nouveau développement mélangeant Azure et Xamarin. La particularité principale de ce dernier est le besoin d'intégrer des composants de reconnaissance d'image, développés en langage natif par le client. Nous avons donc dû créer des bindings sur ces derniers. C'est aussi le premier projet qui nous a permis de mettre concrètement et pleinement en pratique les problématiques DevOps avec Xamarin. De la configuration du gestionnaire de code source, au déploiement sur les stores en passant par la compilation ou les déploiements en beta, nous avons eu la chance de rencontrer tous les écueils possibles et ainsi apprendre énormément. Ce quatrième château, constitué de deux ailes se ressemblant mais avec des différences notables est construit, et les plagistes dans leur immense majorité sont maintenant admiratifs. C'est le moment pour le clou du spectacle !

La cinquième étape

Forts de toutes nos expériences, nous avons pu aborder notre deuxième projet Xamarin au Studio sereinement en sachant tout ce que nous pourrions faire. Ce projet grand public, à fort besoin de performance et de qualité, ainsi qu'à fort partage de code a été mené sans heurts notables. Nous avons perfectionné notre chaîne DevOps, codé énormément de tests unitaires avec près de 90% de couverture sur les parties de code partagé. Ce projet nous a permis de nous assurer que notre approche du développement Xamarin natif est bonne et permet une forte satisfaction client.

Notre dernier château est jalousé par les plages voisines et nous tentons de transmettre ce que nous avons appris aux vacanciers intéressés.

CONCLUSION

Xamarin n'est pas un outil miracle dans le sens où il fait tout, tout seul, et sans efforts de la part du développeur. On n'a jamais vu un marteau frapper directement sur un clou tout seul sans intervention d'un être humain (et je vous vois venir avec la gravité, etc, Oui un marteau peut tomber sur un clou mais ce n'est pas frapper). Il faut qu'un être humain utilise le marteau et se tape les doigts quelques fois avant d'apprendre à bien s'en servir.

Xamarin est un merveilleux outil et ceci aussi nos expériences le confirment.

La prochaine étape ? [3] •


3

Xamarin **natif** ou Xamarin **Forms** ?

Antony Canut
Développeur Xamarin
@Cellenza

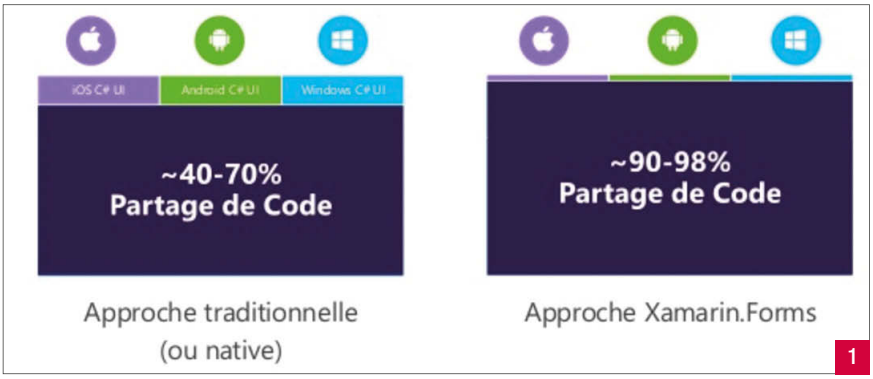
cellenza
DOESITBETTER | Conseil - Expertise Microsoft & méthodes agiles

Au moment d'entamer le développement d'une application cross-platform avec Xamarin, la première question que l'on se pose est toujours la suivante : faut-il choisir Xamarin.Forms ou Xamarin Natif ?

Le mauvais réflexe [1]

Pour beaucoup d'entreprises, la réponse semble évidente après quelques recherches sur le Web nous informant que Xamarin.Forms propose un partage de code allant jusqu'à 100%.

Cependant la réalité est en général très différente, et le choix technique d'utiliser une forme ou une autre de Xamarin ne dépend pas vraiment de cette publicité annoncée par Xamarin/Microsoft. Un ensemble de critères et de conditions sont à connaître pour développer son projet avec la bonne technologie.



1

La cible

Il est d'abord important de savoir s'il s'agit d'une application pour le grand public. Une application de ce genre devra plutôt viser du Xamarin Natif pour éviter différents écueils qui pourraient survenir lors du développement cross-platform. Lors du développement d'une application de type Business to Business, l'emploi de Xamarin.Forms peut être envisagé ici mais soumis à quelques conditions plus techniques ci-après détaillées.

Les fonctionnalités

En second nous abordons le sujet qui porte sur la prédisposition de l'application à utiliser des fonctionnalités natives telles que l'appareil photo ou le GPS. Ce genre de fonctionnalité peut généralement nécessiter un développement plus important sur Xamarin.Forms si aucune solution tierce n'existe, il faut donc se renseigner sur les existants avant le début du projet. Cela peut également réduire l'expérience utilisateur suivant son implémentation et au pire des cas rencontrer des problèmes insolubles sur l'une des deux plateformes. Un exemple concret de ces limitations : avec deux téléphones, l'un Android, l'autre iOS, ils pourront tous les deux récupérer la liste des réseaux wifi à portée mais seul l'appareil Android pourra se connecter automatiquement à un réseau sans action de l'utilisateur. Sur une petite application comprenant seulement une ou deux fonctionnalités natives, cela ne posera pas forcément de difficultés mais sur un projet plus imposant, cela ne donnera pas quelque chose de facilement exploitable et une approche native sera davantage conseillée pour adapter le projet au cas par cas plutôt que de faire tout en même temps et perdre du temps sur ce qui n'est pas possible de réaliser.

Design de l'application "Flaty's" en Xamarin.Forms sur Windows Phone / iOS / Android

Le design

Ensuite il faudra savoir si l'on désire avoir un design très similaire voire identique sur chaque plateforme. Contrairement aux pensées communes, il est beaucoup plus compliqué de faire un design similaire en Xamarin.Forms; cela inclura l'écriture de beaucoup de code lié au design et ceci sans utiliser les storyboards sur iOS ou les xml/axml sur Android. Le rendu final est d'ailleurs différent sur chaque plateforme et il faudra rajouter des features comme celle de type "OnPlatform" pour forcer une application à avoir des contrôles aux mêmes endroits ou à la même taille. Il existe d'ailleurs de nombreuses techniques qui permettent de faire cela. On retrouve parmi elles les renderers, les effects et de nombreux autres qui agiront à des endroits différents du code ou du design pour permettre à celui-ci de se transformer au bon moment, lors de la compilation, de l'exécution et sur certaines conditions. Sur ce genre de cas, cela ne servira donc à rien de choisir le développement sur Xamarin.Forms car tout sera à redévelopper. [2]

Enfin si votre application contient beaucoup de

visuels spécifiques à l'application, on se retrouvera comme dans le précédent cas à redévelopper chaque contrôle et chaque design dans du code spécifique à chacune des plateformes sans le support des designers natifs.

Finalement on choisit quoi ?

En somme, dans un projet qui dispose d'une très forte dépendance au design, dont le marché réside dans la masse d'utilisateurs publics, ou qui doit s'appuyer sur de nombreuses fonctionnalités native, une application doit éviter l'utilisation de Xamarin.Forms. Pour les autres cas, celle-ci peut alors être envisagée mais là encore cela dépendra du projet et de l'équipe. Un gros projet se tournera ainsi naturellement vers du natif. Par ailleurs, une équipe inexpérimentée dans le développement mobile qui souhaiterait ne pas faire de développement Xamarin natif sera obligée d'apprendre iOS, Android et Xamarin.Forms pour espérer produire quelque chose de viable, de plus, la formation d'une telle équipe sera très couteuse au projet.

Nos choix

Nos projets sont de bons exemples de ce choix. Sur trois projets, nous avons utilisé plusieurs manières de développer.

Le premier cas

Le premier projet s'adresse au grand public, avec des contrôles créés avec les outils natifs (ObjectiveC/Java) et intégrés sous la forme d'un Framework. Notre application fonctionne en s'appuyant sur ce dernier. De plus, le design est toujours identique sur chaque plateforme tout en réalisant des livrables sur les deux en même temps. Avec autant de contraintes, le choix d'utiliser un développement Xamarin Natif a donc été largement adopté par l'équipe et l'application a pu ainsi être développée avec cette méthode.

Le second cas

Dans un deuxième cas, un client a souhaité développer un projet se connectant à une caméra externe. Les contraintes de design n'existaient pas, le résultat devait être livré très rapidement par le développeur seul en charge du projet. Initialement, l'application devait pouvoir être déployée sur un Android et un iPhone, mais pas pour le grand public. En tant que POC (proof of concept), Xamarin.Forms s'est trouvé très pratique pour le développement du projet, et ce fut d'ailleurs un succès lors de sa livraison puisque celui-ci a pu délivrer tout ce qui était attendu. Le design lui-même a pu évoluer lors d'une reprise du projet. Et le résultat attendu n'étant pas d'une trop grande précision, l'avantage de la couche graphique partagée s'est encore une fois trouvé appréciable.

Le dernier cas

Le dernier projet que je peux citer est le cas d'une entreprise qui a souhaité créer une application pour ses clients. Tout d'abord, cette entreprise a souhaité investir dans un développement iOS mais avec une promesse de développer l'application Android par la suite. Dans cette optique, Xamarin représentait un cas avantageux pour gagner du temps sur le développement tout en gardant la même équipe projet lors de la conception du second produit. Cela permet également de mutualiser les tests unitaires de l'application sur toute la partie de code partagé, et ainsi obtenir à cet endroit une couverture du code supérieure à 85%.

Initialement, seul le design iOS était prévu, mais la pratique prouve que pour fidéliser ses utilisateurs, un design et une expérience se doivent d'être similaires pour éviter des déceptions lors d'un changement de smartphone. L'identité de la marque est ainsi toujours préservée en restant cohérent quel que soit le support utilisé, qu'il s'agisse d'un site Web ou d'autre chose. Par ailleurs, l'application demandait quelques forts usages natifs que tous les téléphones n'avaient pas tel que le TouchID dont ne disposent que les derniers iPhones et quelques Android récents qui disposent eux-mêmes d'une variante tout aussi efficace. Au final, les directives du projet ont amené le leader technique à imposer le choix de Xamarin natif pour rendre le développement de l'application plus simple et plus efficace.

CONCLUSION

Sur tous les projets que vous pourriez engager, la prise en compte des besoins de l'application, de ses critères en termes d'ergonomie, de design, de la proportion de code que l'on souhaite partager, des tests que l'on souhaite mettre en place etc., est indispensable.

Chacun de ces critères va permettre de décider quelle technologie utiliser mais ce ne seront pas les seules conditions. L'équipe et sa composition compte aussi beaucoup. En effet, si vous donnez votre projet à une équipe contenant quelques membres expérimentés en mobilité, le choix de Xamarin.Forms reste un choix viable.

Dans le cas contraire, il faudra penser qu'un développeur devra être formé au développement iOS, Android et Xamarin.Forms. Cette phase d'apprentissage est longue, et nombreux sont les projets qui peuvent échouer par sa faute. Cependant dans une équipe viable, les bons ingénieurs pourront former les autres plus rapidement.

Pratiquer un développement natif dans un premier temps pourra permettre à un développeur de se former petit à petit sur chaque plateforme, et par la suite pourquoi pas, s'autoriser à développer du Xamarin.Forms, si bien sûr le projet le permet. C'est d'ailleurs souvent le cas de POCs qui peuvent servir à éprouver la technologie et connaître les limites et les possibilités. Quelques projets Business to Business se tourneront également vers cette approche, le design n'étant bien souvent pas une priorité, préférant se concentrer sur les fonctionnalités du produit et sur le temps de développement que cela peut faire gagner. •

Codons !

John Thiriet
Technical Manager Mobilité @Cellenza
Microsoft MVP & Xamarin MVP
@johnthiriet

cellenza
DOESITBETTER | Conseil - Expertise Microsoft & méthodes agiles

Les vacances sont là, on passe du temps sur la plage, on profite du soleil etc., mais surtout, la nouvelle saison de Games of Thrones arrive mi-juillet ! Alors quoi de mieux que de développer une petite application Xamarin pour en profiter ? Non, pas la peine de répondre à cette question je sais que vous pensez comme moi. Alors c'est parti, on va développer un Quizz sur le thème de Games of Thrones !

Native UI ou Forms ?

Comme à chaque fois, la question que l'on va se poser c'est : "Je choisis une UI native ou Forms" ? Vous l'aurez compris ou vous l'apprendrez peut-être ici mais ayant l'habitude de faire des applications B2C, je préfère bien souvent une UI Native. Cependant la réponse à cette question ici est très simple, j'ai deux soirs pour écrire cet article et développer une application, alors en avant Xamarin Forms !

Mise en place de la solution

Pour cette application je vais travailler avec Visual Studio for Mac déjà installé sur ma machine. Cependant, libre à vous de travailler avec Visual Studio 2017 sur Windows.

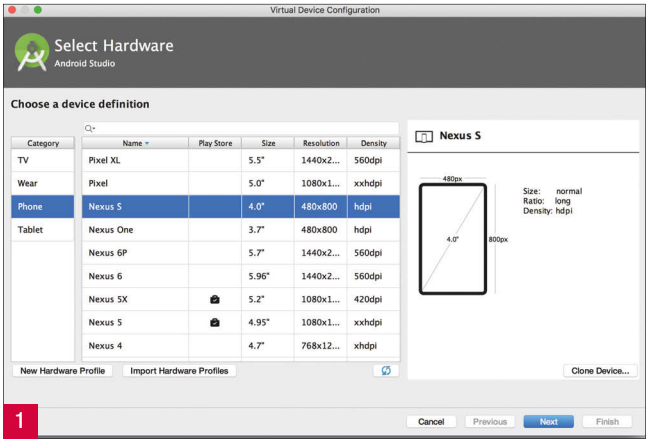
En ce qui concerne le développement iOS ça va donc être très simple pour moi. Pour Android je vais utiliser les émulateurs officiels, mais je vais directement passer par Android Studio pour les créer, l'interface est plus simple. [1]

La première règle lorsque l'on commence un projet c'est de prévoir un contrôleur de code source. Ici, puisque les sources de cet article seront disponibles à vous tous, chers lecteurs, je vais héberger le code sur GitHub et vais donc créer un nouveau repository public du nom de GotQuizz. [2]

Une fois le repository créé, il ne reste qu'à le cloner sur notre machine locale. Je vais donc utiliser l'application GitHub Desktop déjà installée et configurée avec mon compte pour l'opération de clone. Il ne reste maintenant plus qu'à créer notre solution Xamarin dans ce dossier.

Dans Visual Studio on va donc créer un nouveau projet multiplateforme de type Blank Forms App. On va nommer cette application GotQuizz et utiliser une Portable Class Library et des fichiers XAML pour l'interface graphique. [3]

Une bonne idée c'est également de prévoir un fichier gitignore pour Xamarin. Lors du processus de création, on peut préciser à Visual Studio que l'on en souhaite un. Cela nous évitera d'envoyer dans GitHub des fichiers inutiles générés par la compilation.



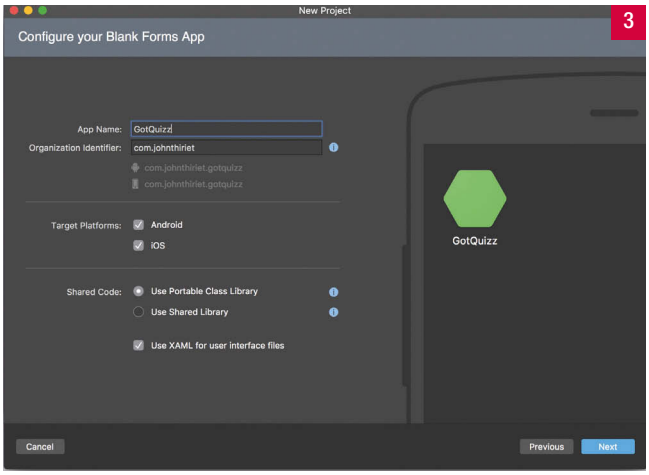
L'API

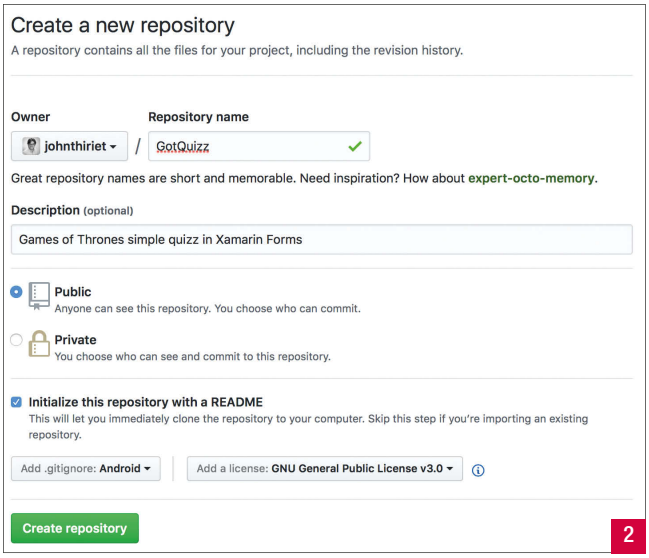
Pour alimenter notre Quizz nous allons nous baser sur une API spécialisée sur Games of Thrones disponible ici : https://api.got.show/doc/. Elle comprend une documentation exhaustive de toutes les informations remontées.

Cependant pour ce quizz et vu le temps limité à ma disposition j'ai choisi quelque chose d'assez simple. On va poser comme question : "Ce personnage a-t-il déjà été à cet endroit" ?

Et pour cela il suffira d'un seul appel : https://api.got.show/api/characters/locations

On commence par créer, dans notre code partagé, un dossier Models pour y créer nos objets métiers mais, pour se simplifier la tâche, on utilise le site http://json2csharp.com/ pour les générer. On y copie l'url précéden-





te et on colle le résultat dans une classe du dossier Models. J'ai appelé cette classe CharactersLocation.

```
public class CharactersLocation
 {
   public string _id { get; set; }
   public string slug { get; set; }
   public string name { get; set; }
   public List<string> locations { get; set; }
 }
```

On crée ensuite, toujours dans notre code partagé, un dossier **Services** où on ajoute une classe nommée **GotApiService**. Dans cette classe on écrit une méthode nommée **GetCharacters LocationsAsync** qui va devoir faire l'appel au service et renvoyer une **Task<CharactersLocation[]>**.
Il nous faut maintenant faire l'appel au webservice et parser le json du résultat. Il faut donc bien penser à ajouter JSON.NET au projet.

```
public async Task<CharactersLocation[]> GetCharactersLocationAsync()
{
   using (var client = new System.Net.Http.HttpClient())
   {
      var json = await client.GetStringAsync("https://api.got.show/api/characters/locations");
      return JsonConvert.DeserializeObject<CharactersLocation[]>(json);
   }
}
```

Le ViewModel

Chargement des données

L'accès à l'API étant codé, il faut à présent l'utiliser et c'est le rôle du ViewModel. On va donc le créer et l'appeler **GotQuizzViewModel**. Toujours pour se simplifier la vie on va utiliser MvvmLight afin de pouvoir profiter de son **ViewModelBase**. Pensez à bien ajouter le paquet Nuget MvvmLightLibs et non le paquet MvvmLight. En effet ce dernier rajoute des fichiers dont on n'a pas besoin.
Notre ViewModel créé, on va ajouter les champs qui vont contenir la liste complète des personnes et des lieux où ils ont été, ainsi que la liste complète des lieux. On va aussi créer deux méthodes, **StartAsync** et **Start**. La première va contenir le code d'appel au service, la seconde va juste exécuter la première de manière sécurisée car oui elle sera en async void, async void c'est le mal et on ne veut pas remonter cela au niveau de la vue.

```
private IReadOnlyList<string> _locations;
private IReadOnlyList<CharactersLocation> _charactersLocations;

public async void Start()
{
   try
   {
      await StartAsync();
   }
   catch (Exception ex)
   {
      Debug.WriteLine(ex);
   }
}

private async Task StartAsync()
{
   var api = new Services.GotApiService();
   var result = await api.GetCharactersLocationAsync();

   _charactersLocations = result;
   _locations = _charactersLocations
            .SelectMany(x => x.locations)
            .Distinct()
            .ToList();

   StartQuizz();
}
```

Le Quizz

Puisqu'on a les données maintenant prêtes, il est temps de faire le quizz à proprement dit. Pour ce faire on ajoute deux classes à l'application dont le nom est explicite quant à leur utilité respective.

```
public class QuizzQuestion
{
   public string Question { get; set; }
   public bool Answer { get; set; }
}

public class QuizzState
{
   public int NumberOfQuestions { get; set; }
   public int AnsweredQuestions { get; set; }
   public int CorrectAnswers { get; set; }
}
```

On va également ajouter une classe pour générer aléatoirement une question à la demande en combinant la liste des personnages et celle des lieux.

```
public class QuestionGenerator
{
   private Random _random = new Random();

   public QuizzQuestion GenerateQuestion(
      IReadOnlyList<string> locations,
      IReadOnlyList<CharactersLocation> charactersLocations)
   {
      var character = GetRandomCharacter(charactersLocations);
      var location = GetRandomLocation(character, locations);

      return new QuizzQuestion
      {
         Question = $"Has {character.name} ever been to {location} ?",
         Answer = character.locations.Contains(location)
      };
   }
}
```

Je passe volontairement sur le détail de l'implémentation du générateur mais vous avez l'idée. Le code source complet est disponible sur GitHub si cette partie vous intéresse. L'objectif principal étant bien évidemment ici de récupérer une question à poser à l'utilisateur ainsi que la bonne réponse.

L'assemblage

Nous sommes à présent capable de générer des questions aléatoires en fonction de données du service. Il semble donc logique de vouloir les afficher. Mais comme nous n'avons pas de temps à perdre en compilation et en test, on ne va s'occuper que du ViewModel pour l'instant. « Afficher des données » quand on est dans un ViewModel cela revient à dire « créer des propriétés utilisant INotifyPropertyChanged ».
On va donc dès à présent démarrer le quizz, en générant des questions et exposant les propriétés nécessaires à l'affichage ultérieur.

```
private QuizzQuestion _currentQuestion;
private QuizzState _quizz;
public string CurrentQuestionLabel => _currentQuestion?.Question;
public string CorrectAnswersLabel => $"{_quizz?.CorrectAnswers ?? 0} correct answers";
private QuestionGenerator _generator = new QuestionGenerator();

private void StartQuizz()
{
  _currentQuestion = null;
  _quizz = new QuizzState();

  NextQuestion();
}

private void NextQuestion()
{
  _currentQuestion = _generator.GenerateQuestion(_locations, _charactersLocations);
  RaisePropertyChanged(nameof(CurrentQuestionLabel));
  RaisePropertyChanged(nameof(CorrectAnswersLabel));
}
```

La question étant affichée il ne reste plus qu'à fournir la possibilité d'y répondre et pour cela rien de plus simple que deux boutons "Oui" et "Non" ce qui, en termes de ViewModel, se traduit par des commandes.

```
public RelayCommand Yes => _yes;
public RelayCommand No => _no;

private RelayCommand _yes;
private RelayCommand _no;

public GotQuizzViewModel()
{
  _yes = new RelayCommand(() => AnswerQuestion(true));
  _no = new RelayCommand(() => AnswerQuestion(false));
}

private void AnswerQuestion(bool answer)
{
  if (answer == _currentQuestion.Answer)
    _quizz.CorrectAnswers++;

  NextQuestion();
}
```

Et voilà ! En fait on a fini le Quizz à proprement parler et tout ce qu'il reste maintenant à faire c'est de l'affichage.

L'affichage

L'avantage de Xamarin.Forms c'est qu'il est très rapide de prototyper un design et c'est exactement ce que l'on va faire ici. L'application n'a qu'un seul écran et n'a pas de design particulier pour l'instant. On a juste besoin de 4 éléments :

- Un label pour afficher la question,
- Un label pour afficher le nombre de bonnes réponses,
- Un bouton oui,
- Un bouton non.

```
<?xml version="1.0" encoding="utf-8"?>
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms" xmlns:x="http://schemas.
microsoft.com/winfx/2009/xaml" xmlns:local="clr-namespace:GotQuizz" x:Class="GotQuizz.
GotQuizzPage">
  <Grid Padding="10">
    <Grid.Margin>
      <OnPlatform x:TypeArguments="Thickness" iOS="0,20,0,0"/>
    </Grid.Margin>
    <Grid.RowDefinitions>
      <RowDefinition Height="30" />
      <RowDefinition Height="*" />
      <RowDefinition Height="50" />
    </Grid.RowDefinitions>
    <Grid.ColumnDefinitions>
      <ColumnDefinition Width="*" />
      <ColumnDefinition Width="*" />
    </Grid.ColumnDefinitions>
    <Label Grid.ColumnSpan="2" Text="{Binding CorrectAnswersLabel}" HorizontalOptions
="End" />
    <Label Grid.Row="1" Grid.ColumnSpan="2" Text="{Binding CurrentQuestionLabel}"
HorizontalTextAlignment="Center" VerticalTextAlignment="Center"/>
    <Button Grid.Row="2" Text="Yes" Command="{Binding Yes}" />
    <Button Grid.Row="2" Grid.Column="1" Text="No" Command="{Binding No}"/>
  </Grid>
</ContentPage>
```

Ne reste maintenant l'étape finale, associer le ViewModel avec la vue, et pour cela direction le code-behind.

```
public partial class GotQuizzPage : ContentPage
{
  private GotQuizzViewModel _vm = new GotQuizzViewModel();

  public GotQuizzPage()
  {
    InitializeComponent();
    BindingContext = _vm;
  }

  protected override void OnAppearing()
  {
    base.OnAppearing();

    _vm.Start();
  }
}
```

L'application est maintenant parfaitement fonctionnelle, il ne reste qu'à la déployer. [4]

Le déploiement

Pour le déploiement, puisque mon projet est sur GitHub et que ce dernier est déjà associé à mon compte Mobile Center, le choix est vite fait et je me dirige donc vers http://mobile.azure.com pour configurer le déploiement en bêta.

Et c'est à ce moment-là que je me rends compte que je n'ai rien configuré dans mon portail Apple pour le déploiement iOS. Qu'à cela ne tienne, on a Fastlane qui va nous simplifier la tâche : https://developer.xamarin.com/guides/ios/deployment,_testing,_and_metrics/provisioning/fastlane/

Après avoir installé et configuré Fastlane, si ce n'était pas déjà le cas, il n'y a plus qu'à lancer la petite commande magique qui va tout créer : **fastlane match adhoc**.

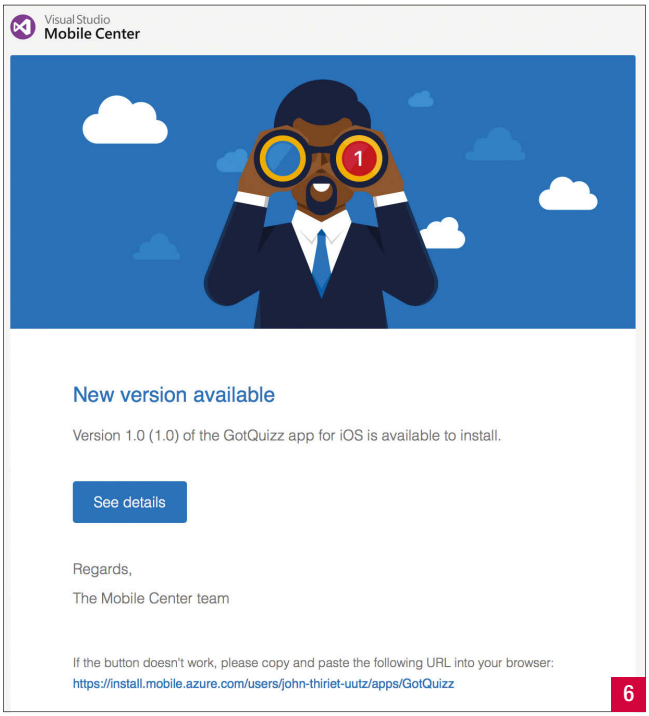
Le certificat et le provisioning profile ainsi créés, il ne reste plus qu'à les récupérer pour les envoyer sur Mobile Center.

Dans Mobile Center il nous faut ajouter une application iOS, pour ensuite se connecter à GitHub et configurer une branche (ici master). C'est l'affaire d'une minute. [5]

L'usine est prête et une build peut être d'ores-et-déjà lancée !

La partie iOS maintenant configurée, on répète les étapes de manière nettement moins compliquée avec Android car nous n'avons pas besoin des certificats et provisioning profiles.

Au final on recevra un joli petit mail de Mobile Center nous informant que l'on peut télécharger l'application. [6 et 7]



6

CONCLUSION

Grâce à cet article très technique, on a vu qu'il était possible, en deux soirs, de partir de rien et de faire une application Xamarin, graphiquement pauvre, certes, mais fonctionnelle, fonctionnant sur iOS et Android, buildée et déployée automatiquement en bêta par une usine de build !

Quand on voit à quel point cela pouvait être dur avant et à quel point c'est devenu simple… Comme diraient nos chers amis : « It's amazing ! ».

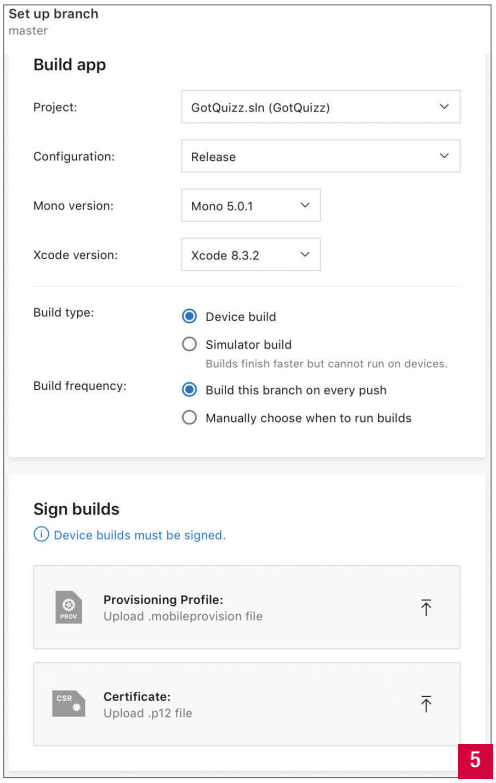
Alors qu'attendez-vous pour vous y mettre aussi ?

Au fait, pour télécharger la solution ça se passe sur Github !

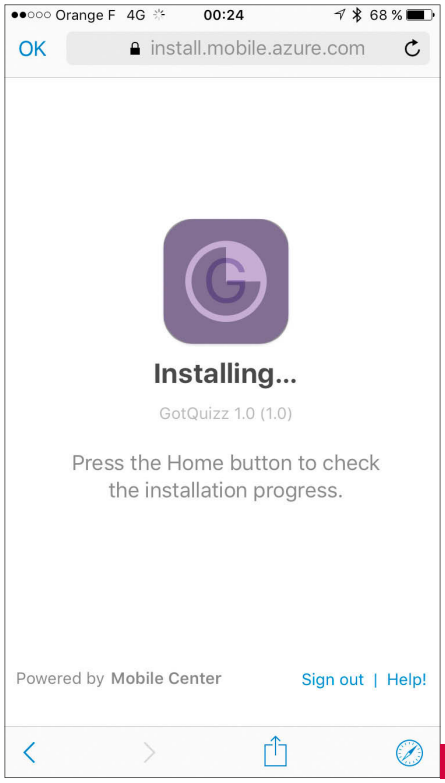
https://github.com/johnthiriet/GotQuizz [8]

8

4



5



7

Monitorer les **threads** avec la Suite Elastic

Armel Chever
Architecte - **Sopra Steria**
J'ai participé depuis un an avec beaucoup d'intérêt à la mise en place de solutions de monitoring basées sur la Suite Elastic chez des clients, avec des retours très positifs.

Cet article va vous présenter comment nous avons pu apporter une réponse à un problème classique qui a donné des maux de tête à bon nombre de développeurs : la maîtrise de la mémoire en production ! Nous vous présenterons comment on peut utiliser la Suite Elastic - Elastic, Kibana, Beats et Logstash - pour monitorer en temps réel la JVM (Java Virtual Machine) d'une application JEE (Java Enterprise Edition) et gagner ainsi drastiquement en efficacité sur les analyses d'incidents.
Dans cet article nous nous positionnerons dans un contexte Java/JEE mais les principes sont réutilisables avec d'autres technologies.

LE PROBLÈME

Lorsqu'une application de dimension importante est confrontée à des problèmes de gestion mémoire en production, il peut être très difficile de remonter à l'origine des dysfonctionnements. En effet, les anomalies liées aux contextes de production sont souvent les plus difficiles à analyser : milliers d'utilisateurs, cas d'utilisations complexes, patrimoine logiciel important… Lors d'une saturation mémoire (les redoutées « OutOfMemory Error » dans le monde Java), la plateforme de production va être complètement indisponible pendant un certain temps et nécessitera un redémarrage complet. L'impact sur les utilisateurs est donc immédiat et brutal ! Place alors aux plans d'urgence et autres « task forces » pour éteindre l'incendie et éviter que l'incident ne se reproduise !
Voici un petit scénario à titre d'exemple qui rappellera peut-être des souvenirs à certains lecteurs de cet article !

DevOps : Chef, mauvaise nouvelle. OutOfMemoryError rencontrée sur le 3ème nœud en milieu de journée. On a dû redémarrer la plateforme en pleine activité.

Chef : Hmm… Etendue des dégâts ?

DevOps : Environ 250 utilisateurs déconnectés et 20 minutes d'indisponibilité. La perte sèche de production évaluée à 5 jours de travail. Les données qui n'avaient pas été enregistrée à temps sont en cours de ressaisie.

Chef : Et je suppose qu'une fois de plus nous n'avons pas trouvé l'origine du problème ?

DevOps : La procédure a été appliquée : heap dump et thread dump ont été générés immédiatement et transmis aux équipes techniques pour analyse.

[Quelques jours plus tard]

Devops : Chef, les premiers résultats de l'analyse des dumps sont arrivés. Cela viendrait d'une requête sur la table GROS_VOLUME qui a brusquement chargé trop d'objets en mémoire.

Chef : Parfait ! Plus qu'à corriger l'anomalie !

Devops : Il y a un hic. Cette table est constamment utilisée par des centaines d'utilisateurs et pour des dizaines de cas d'utilisation différents. Les experts n'arrivent toujours pas à trouver l'origine exacte du problème.

Chef : Que faire ?

Devops : J'ai peut-être une solution Chef… :)

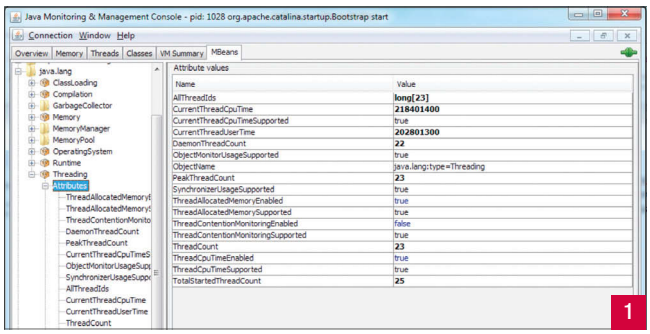


Figure 1 – Consultation du statut d'un thread via la JConsole

LA SOLUTION

L'idée de départ

L'idée est de réaliser un monitoring en temps réel des threads en collectant les données de la JVM via JMX (Java Management Extension). En remontant les informations sur les threads dans la Suite Elastic nous pourrons faire des tableaux de bords dans Kibana qui permettront de comprendre et solutionner les problèmes de mémoire beaucoup plus rapidement qu'en analysant des dumps de JVM post-mortem.

Le principe

Pour monitorer des threads, nous avons voulu nous appuyer sur 2 points clés :

- Pouvoir suivre visuellement un indicateur mémoire associé de chaque thread en temps réel,
- Disposer d'un lien de traçabilité entre les informations sur les threads et les logs applicatifs.

Ainsi il devient possible de remonter immédiatement à l'origine métier d'une mauvaise gestion de la mémoire.

MISE EN ŒUVRE

Créer un MBeans custom

Bref rappel sur JMX

JMX est une API Java qui permet de monitorer une JVM. Elle permet de collecter des métriques au travers de services appelés « MBeans ». L'API fournit de nombreux MBeans par défaut mais il est également possible de développer ses propres MBeans. (C'est ce que nous allons faire dans cet article.)

Les MBeans d'une JVM peuvent être appelés par un client JMX, par exemple une JConsole. Concernant les threads, il existe un MBean qui permet de consulter les informations sur un thread donné : *com.sun.management.ThreadMXBean* :

- Consommation mémoire ;
- Consommation CPU ;
- Statut : RUNNING, BLOCKED, WAITING… [1]

Création du MBean

Malheureusement le *ThreadMXBean* fourni par défaut ne correspond pas tout à fait à notre besoin : il expose les informations d'un thread donné alors que nous voulons récupérer un statut global sur l'ensemble des threads à un instant t. Le développement spécifique que nous allons faire est relativement simple : notre MBean interroge *ThreadMXBean* n fois pour récupérer les informations sur l'ensemble des threads qui nous intéressent et il formate en JSON l'ensemble des données collectées pour permettre une exploitation avec la Suite Elastic.

MBean interface : ThreadsMonitoringMXBean.java

```
package threadmon.monitoring;

import com.fasterxml.jackson.core.JsonProcessingException;

/**
 * Interface du MBean pour récuperer les infos des threads
 *
 * @author armel.chever
 *
 */
public interface ThreadsMonitoringMXBean {

   /**
    * Retourne un json contenant les infos des threads de la JVM
    * @return  String un json contenant les infos des threads de la JVM
    */
   public String getThreadsStatus() throws JsonProcessingException;

}
```

MBean implémentation: ThreadsMonitoring.java

Code complet sur www.programmez.com

POJO pour représenter l'état d'un thread : ThreadState.java

Code complet sur www.programmez.com

POJO pour représenter l'état global d'un ensemble de threads : ThreadsStatus.java

```
package threadmon.monitoring;

import java.util.ArrayList;
import java.util.List;

/**
 * @author armel.chever
 *
 */
public class ThreadsStatus {

   protected List<ThreadState> threadsStatus = new ArrayList<ThreadState>();

   public List<ThreadState> getThreadsStatus() {
      return threadsStatus;
   }

   public void setThreadsStatus(List<ThreadState> threadsStatus) {
      this.threadsStatus = threadsStatus;
   }
}
```

Nous pouvons à présent déployer notre MBean et lui attribuer un nom. Pour cela nous allons créer une application JEE minimale. (J'ai choisi d'utiliser Spring Boot pour cet exemple.)

```
@Configuration
@EnableAutoConfiguration
@ComponentScan
public class Application {

   public static void main(String[] args) throws Exception {

   SpringApplication.run(Application.class, args);

   MBeanServer mbs = ManagementFactory.getPlatformMBeanServer();
   ObjectName name = new ObjectName("monitoring.threadmon.mbean:type
=ThreadsMonitoring");
   ThreadsMonitoring mbean = new ThreadsMonitoring();
   mbs.registerMBean(mbean, name);

   }
}
```

Pour activer l'accès distant à JMX, la JVM doit être lancée avec les options suivantes :

- Dcom.sun.management.jmxremote
- Dcom.sun.management.jmxremote.port=11000
- Dcom.sun.management.jmxremote.authenticate=false
- Dcom.sun.management.jmxremote.ssl=false

Une fois l'application lancée, on peut tester le bon déploiement du MBean sur le port 11000 avec la JConsole et vérifier que les statuts des threads sont bien exposés en JSON : [2]

Voici les données exposées par le MBean:

```
{
  "threadsStatus":[
    {
      "threadId":"32",
      "threadName":"http-nio-8080-exec-3",
      "threadState":"WAITING",
      "allocatedBytes":118680
    },
    {
      "threadId":"31",
      "threadName":"http-nio-8080-exec-2",
      "threadState":"WAITING",
      "allocatedBytes":123496
    },
    {
      "threadId":"30",
```

```
      "threadName":"http-nio-8080-exec-1",
      "threadState":"WAITING",
      "allocatedBytes":7063928
    }
  ]
}
```

Collecte des infos dans la Suite Elastic

La collecte des données s'effectue par le plugin JMX de Logstash. Ce plugin est pour l'instant communautaire, il n'est pas officiellement maintenu par Elastic dans les versions de base de Logstash, il faut donc le télécharger et l'installer séparément (*"bin/logstash-plugin install logstash-input-jmx"*).

Input Logstash

La configuration d'un appel périodique (ici toutes les 5 secondes) au MBean s'effectue comme suit :

01-input-jmx-thread.conf :

```
input {
  jmx {
    path => "D:\projects\Lab\elastic\logstash-2.3.4\conf\jmxsources"
    polling_frequency => 5
    type => "thread"
    nb_thread => 4
  }
}
```

Puis dans le répertoire « D:\projects\Lab\elastic\logstash-2.3.4\conf\jmx-sources » :

jmx_sources.conf :

```
{
  "host" : "localhost",
  "port" : 11000,
  "queries" : [
   {
    "object_name" : "monitoring.threadmon.mbean:type=ThreadsMonitoring",
    "attributes" : [ "ThreadsStatus"],
    "object_alias" : "Threads"
   }
  ]
}
```

4.2.2. Filter Logstash

On utilisera Logstash pour découper le flux JSON. La particularité de ce filtre est que pour un flux JSON reçu il y aura n documents Elasticsearch créés. En effet le flux en entrée contient les données pour un ensemble de threads, or pour pouvoir faire des analyses dans Kibana il nous faut absolument un seul document par thread.

```
filter {

  if [type] == "thread" {

        # Recuperation du json
          json {
                source => "metric_value_string"
                target => "jsondata"
          }

        # Eclatement du flux JSON evenements (1 evenement par thread)
        split {
                field => "[jsondata][threadsStatus]"
        }

        # Renommage des champs pour la lisibilte
        mutate {
                rename => { "[jsondata][threadsStatus][threadId]" => "thread_id" }
                rename => { "[jsondata][threadsStatus][threadName]" => "thread_name" }
                rename => { "[jsondata][threadsStatus][threadState]" => "thread_state" }
                rename => { "[jsondata][threadsStatus][allocatedBytes]" => "allocated_bytes" }
        }

        # Conversion de la mémoire en integer
        mutate {
                convert => { "allocated_bytes" => "integer" }
        }
}
}
```

Tableaux de bords Kibana

Une fois l'application déployée et la Suite Elastic lancée, voici un tableau de bord simple qui permet d'afficher les demandes d'allocation mémoire par thread pour notre application Spring Boot : [3]

En cas d'interblocages ou de problèmes de performance sur une application en production, une autre possibilité intéressante est de suivre les statuts des threads pour repérer ceux qui bloquent les autres : [4]

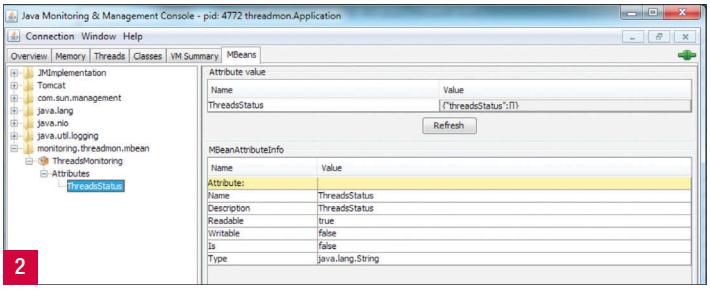


Figure 2 – Déploiement du MBean ThreadsMonitoring

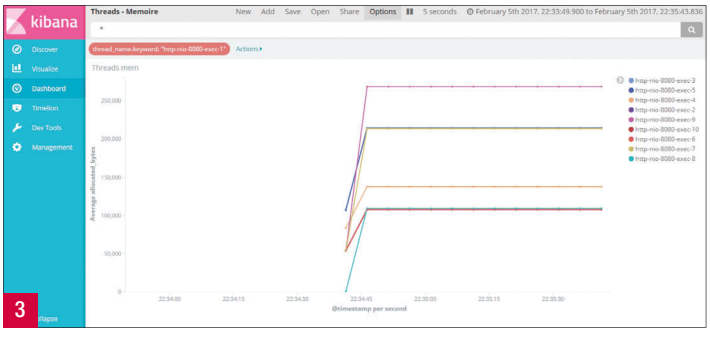


Figure 3 – Tableau de bord de suivi mémoire des threads

Traçabilité avec les logs applicatifs

Enfin pour que la solution soit complète, il convient d'ajouter l'identifiant du thread courant dans vos logs applicatifs. Ainsi, nous pourrons établir un lien de traçabilité entre les tableaux de bord Kibana ci-dessus et les logs « traditionnels » remontés dans la Suite Elastic.

Pour cette étape vous avez le choix des armes. Les frameworks de loging permettent de le faire simplement par configuration (Logback, Log4J), vous pouvez aussi utiliser de l'AOP.

DÉMONSTRATION

L'application test

Pour la démonstration, nous allons déployer un service REST appelable à l'URL http://localhost:8080/service?thousands= qui prend un unique paramètre appelé « thousands ».

Le rôle de ce service est de charger des objets Java par milliers, autant de milliers que le chiffre renseigné dans le paramètre thousands. Ex : thousands = 3 à 3000 objets chargés en mémoire. L'objectif du test est de lancer en parallèle des appels à ce service REST, nous utiliserons pour cela JMeter. Ayant configuré la JVM avec 50 Mo maximum, le OutOfMemoryError devrait vite être atteint.

Lancement du test JMeter

Le test effectué avec JMeter consiste à jouer en parallèle des appels au service REST en faisant varier les valeurs du paramètre thousands. Les valeurs à jouer sont placées dans un fichier .csv dans lequel nous avons volontairement mis des valeurs très différentes. Voici le résultat après quelques secondes d'exécution : [5]. Comme prévu, dès que la mémoire heap a atteint 50 Mo une erreur OutOfMemoryError est apparue.

Nous allons maintenant voir comment analyser un tel crash avec l'outil JMX/la Suite Elastic mis en place.

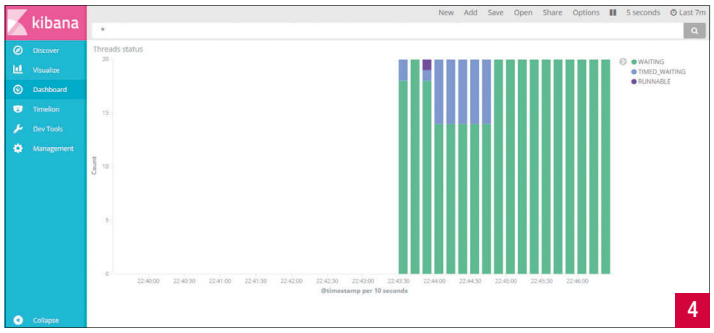


Figure 4 – Tableau de bord de suivi des statuts de threads

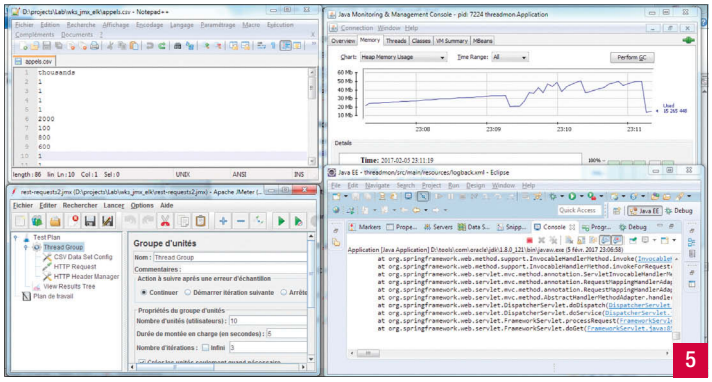
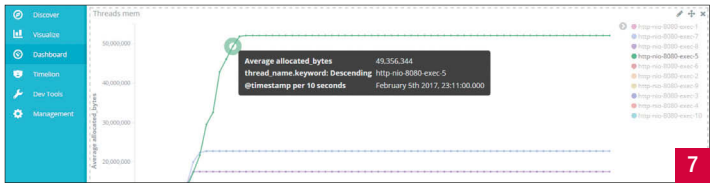


Figure 5 – Lancement du test avec JMeter



Analyse rapide du OutOfMemoryError

Consultons le tableau de bord sur la mémoire des threads. On constate immédiatement que les différents threads du serveur d'application effectuent des demandes d'allocation mémoire au moment où le tir JMeter est lancé. L'un d'entre eux notamment semble demander davantage de mémoire. [6]

En approchant la souris, on repère immédiatement le nom du thread à l'origine de l'augmentation mémoire : http-nio-8080-exec-5 [7]

En un clic on épingle ce filtre dans Kibana et on retourne dans la vue Discover pour consulter les logs applicatifs associés. Voici ce qui apparaît : [8]

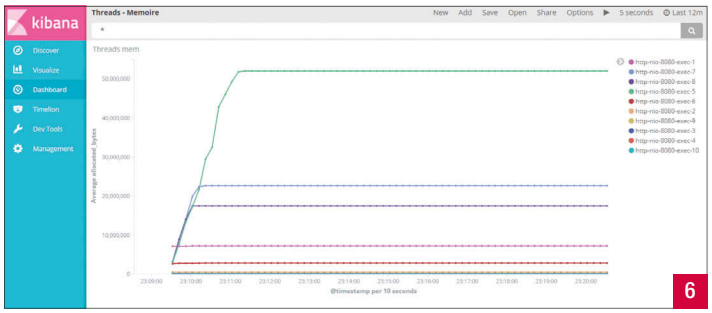
On peut alors filtrer, parmi tous les logs applicatifs, uniquement ceux émis par le thread consommateur de mémoire. Ainsi on peut retrouver très facilement le cas métier à l'origine du dysfonctionnement.

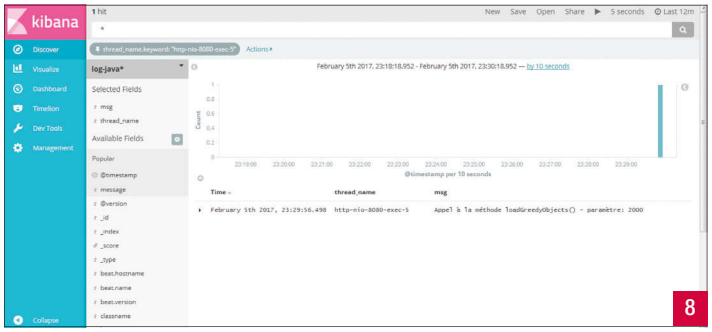
Dans notre cas, sans surprise, il s'agit de l'appel à notre service avec le paramètre à 2000, soit la plus grande valeur présente dans le fichier .csv pris par JMeter.

CONCLUSION

Dans cet article nous vous avons montré comment effectuer un monitoring en temps réel de la JVM et des threads en particulier. Cette fonctionnalité est relativement peu coûteuse à mettre en place et peut permettre un gain de temps considérable en termes d'analyse de problèmes mémoire en production. Cette implémentation a permis de résoudre plusieurs problèmes de production sur un SI fortement sollicité et d'améliorer sa disponibilité par la détection immédiate de l'origine fonctionnelle des mauvaises gestions de la mémoire (éditions volumineuses, exports de données mal optimisés, memory leaks...) et donc leur correction rapide.

Les principes mis en œuvre dans ce monitoring peuvent être réutilisés pour superviser d'autres métriques, par exemple les Locks en BDD, les erreurs SQL, les pools de connexion (jdbc, http), les pools de threads parmi d'autres. •





Créer une API Hypermedia avec Threerest et Node.js en moins de 10 minutes

Partie 1

Vincent Piard
Expert Technique
SQLI Nantes

Wilfried Moulin
Expert Technique
SQLI Nantes.

Une API (Application Programming Interface) est un ensemble de normes qui permet l'échange de données entre des systèmes hétérogènes. En clair, il s'agit d'un code qui remplit un service. Il peut donc être sollicité par n'importe quel acteur (humain ou machine). Donc pour une API donnée, on aura un fournisseur de service autrement dit l'API et le consommateur du service, un client Web par exemple. La beauté des API réside dans le fait que le consommateur ne connaît pas l'implémentation du service. De ce fait, un client PHP peut consommer une API en PHP, en Java ou bien encore en Go. La seule contrainte pour l'API est de rendre correctement son service.

Afin de pouvoir communiquer le plus simplement possible, HTTP est utilisé pour la communication. L'accès aux APIs se fait donc à travers des liens. Voici quelques exemples d'URL d'API :

https://maps.googleapis.com/maps/api/place/nearbysearch/json?location=37.865025,-119.538308&radius=20&key=ABCDEF

http://twitter.com/statuses/

http://gdata.youtube.com/feeds/

http://api.flickr.com/services/rest/

Une API peut être publique, semi-publique ou privée. Il existe de nombreux catalogues d'API, et ce pour n'importe quel domaine. Cela va de la consultation en temps réel des horaires de bus (http://data.nantes.fr/donnees/detail/info-trafic-tan-temps-reel/) à la récupération d'informations sur des livres (https://developers.google.com/books/), en passant par la validation d'email (https://market.mashape.com/pozzad/email-validator-1). Il existe même une API pour parler comme Yoda (http://www.yodaspeak.co.uk/). Il existe une API pour tout et n'importe quoi. La seule limite pour faire une API est votre imagination (et votre temps !). Petite précision, nous allons uniquement aborder ici les API REST. Alors let's code !

NodeJS chez les développeurs

Node.js est une plateforme en JavaScript. Il permet de coder en JavaScript côté serveur. L'installation de Node.js (https://nodejs.org/en/) se fait sans problème particulier. Nous allons omettre volontairement certains détails d'implémentation concernant plus Node.js que les API. L'idée étant ici de coder une API et non d'apprendre Node.js.

Pour commencer, il faut créer un répertoire vide (par exemple bdtheque_simple) puis y ajouter un fichier vide, nommé app.js par exemple. Pour le reste de l'article, le répertoire de base du projet sera donc dénommé bdtheque_simple, vous pouvez naturellement l'appeler comme vous le souhaitez. Le code suivant va créer notre serveur HTTP avec Node.js

```
var http = require('http');
http.createServer(function (req, res) {
  res.writeHead(200, {'Content-Type': 'application/json'});
  res.end(JSON.stringify({message:"Mon premier serveur Node.js !!!"}));
}).listen(8080);
console.log('Le serveur est accessible ici : http://localhost:8080/');
```

Petite explication de texte : l'instruction createServer va enclencher la création du serveur et permet d'ajouter un listener sur la réception d'une requête (ici l'écriture d'une simple ligne). L'instruction finit en précisant le port d'écoute (8080) avec l'instruction listen.

Pour lancer votre serveur, vous devez ensuite ouvrir un shell Node.Js . Puis vous positionner à la racine de votre projet et rentrer la commande suivante dans le shell de Node.Js :

```
node app.js
```

Vous pouvez alors tester votre serveur à l'adresse http://localhost:8080, vous obtenez le résultat suivant : [1]

Autre façon de tester vos requêtes HTTP : HTTPie. HTTPie est un curl avancé à destination des humains ; cela permet de faire des requêtes de façon plus simple et plus intuitive. C'est un outil extrêmement pratique et qui va vous rendre de grands services. [2]

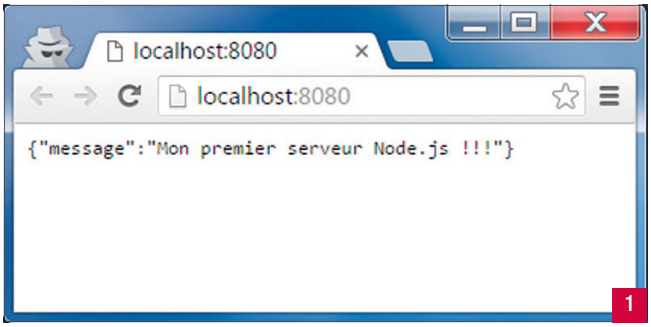
L'installation de HTTPie se fait sur cette page (https://github.com/jkbrzt/httpie). Vous verrez les différentes syntaxes de cet outil au cours de l'article.

Rendu à ce point vous avez donc installé Node.js, créé votre premier serveur et testé son fonctionnement. Il est donc temps d'introduire notre premier framework.

Express et compagnie

Express est un framework qui permet d'accélérer le développement d'application Web. En d'autres termes, Express va nous permettre de structurer notre projet tout en nous apportant des outils puissants et précieux pour un gain en productivité.

Pour ajouter Express à votre projet, nous allons utiliser npm (Node



1

Package Manager). Il s'agit d'un utilitaire qui va vous permettre de gérer nos différents modules Node. Faites-nous confiance : une fois que vous y aurez pris goût vous ne pourrez plus vous en passer. Npm est présent dans la distribution de Node.js donc vous ne le savez pas, mais vous avez déjà installé npm. Pour fonctionner npm a besoin d'un fichier package.json qui doit être présent à la racine de votre projet. Pour cela, il suffit de taper la commande suivante et de répondre aux questions :

```
npm init
```

Voici le résultat du npm init :

```
{
  "name": "bdtheque",
  "description": "mon application de demo pour une gestion des BDs",
  "version": "0.0.1",
  "main": "app.js"
}
```

Si vous voulez ajouter une dépendance au package.json il faut taper en ligne de commandes :

```
npm install express --save
```

Ceci va permettre d'installer express au sein de votre de projet. Le --save va inscrire la dépendance au sein du fichier package.json. Le fichier va donc être réécrit de façon automatique par npm de la manière suivante :

```
{
  "name": "bdtheque",
  "description": "mon application de demo pour une gestion des BDs",
  "version": "0.0.1",
  "private": true,
  "dependencies": {
    "express": "^4.13.4"
  }
}
```

Vous remarquerez ici l'ajout de l'entrée dependencies. L'accent circonflexe sur la version permet de préciser à npm de se mettre à jour pour toutes les versions majeures à partir de 4.x.x, dès qu'on relancera l'installation des dépendances (commande 'npm install' sur notre projet) mais il ne s'upgradera pas sur la version 5.x.x.
Le projet sera nommé bdtheque_express. La création d'un serveur en Express diffère légèrement de la manière précédemment vue, mais le principe reste identique. Ouvrez votre fichier app.js et supprimez ce que vous avez écrit précédemment et remplacez le par le code suivant :

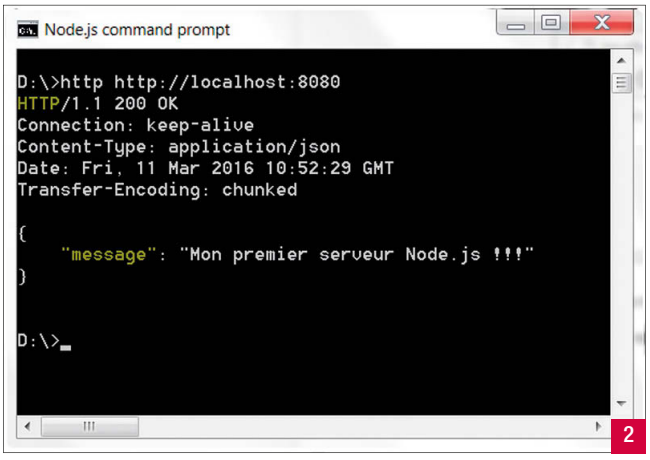
```
//Import du framework Express
var express = require('express');
// Instanciation du framework
var app = express();

// Création de la route par défaut, celle présente à la racine.
app.get('/', function(req, res) {
  res.send({message:"Mon second serveur Node.js !!!"});
});

// Déclaration du port d'écoute du serveur.
app.listen(8080);
console.log('Le serveur est accessible ici : http://localhost:8080/');
```

Comme précédemment, vous pouvez tester votre serveur express (n'oubliez pas de lancer votre serveur avec la commande node apps.js). [3]
Vous vous retrouvez donc avec exactement le même résultat que précédemment. La seule différence est que l'application en cours d'exécution est de type Express et cela va nous rendre grandement service dans la prochaine partie.

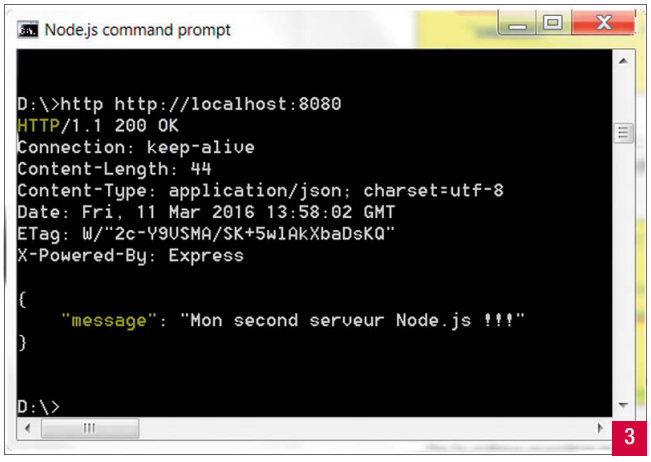
La Database d'or

Maintenant que nous avons un serveur opérationnel, nous allons vouloir lui ajouter des opérations. Dans notre exemple, nous partons sur une gestion d'une BDthèque. Donc le premier service à implémenter est de connaître le nom de tous les auteurs des BD contenus dans notre collection de BD. Afin de pouvoir tester notre API, nous allons nous baser sur une base de données stockée au format JSON. Nous allons donc copier le fichier database.json dans le répertoire database à la racine de notre projet. La base de données est regroupée en liste d'auteurs, de séries et de titres.
Un auteur participe à une ou plusieurs séries. Une série contient un ou plusieurs titres. Un titre appartient donc à une série et à un auteur. Pour utiliser notre base de données au sein de notre serveur, il suffit d'utiliser require pour notre fichier Json (notez que l'ajout de l'extension du fichier est optionnel) :

```
var db = require('./database/database');
```



2



3

Tour de Gaule de notre API

Le projet sera nommé bdtheque_routes. Pour accéder à notre ressource authors (liste des auteurs présents dans la base), il est nécessaire de créer une route. Une route représente le chemin pour atteindre notre ressource. Plus simplement, il s'agit de l'url permettant d'interroger notre ressource.

Trois paramètres sont nécessaires pour obtenir notre ressource :

- Une méthode contenant le code à exécuter ;
- Une URL spécifique ;
- Un verbe http indiquant l'action à effectuer ;

Selon l'architecture REST, le verbe http permet de connaître l'action à effectuer côté serveur. Un DELETE est assez explicite ainsi qu'un GET. La fameuse différence entre POST et PUT s'explique assez simplement. Si vous savez sur quelle ressource précisément agir, que cela soit pour un remplacement, une modification ou même une création, il faut utiliser le PUT. Pour les autres cas le POST est là. Il faut aussi signaler que le PUT est idempotent contrairement au POST. Toutes les requêtes PUT vont renvoyer la même réponse de la part du serveur ce qui n'est pas le cas d'un POST.

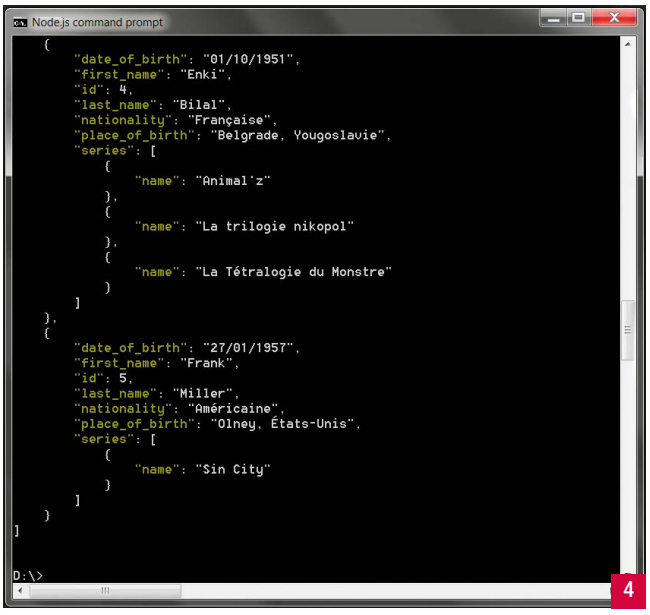
L'URL permet d'avoir l'emplacement exact de la ressource. La méthode implémente la représentation de la ressource que nous voulons renvoyer. Voici comment cela peut fonctionner.

La création d'une route en Express se résume au bout de code suivant :

```
// Création de la route pour consulter la liste des auteurs.
app.get('/authors', function(req, res) {
  res.json(db['authors']);
});
```

Le premier paramètre de la méthode GET est l'URL à saisir pour atteindre la ressource. Si vous souhaitez faire du post ou du put il vous faudra faire un app.post ou un app.put. Attention à bien respecter la sémantique des verbes HTTP. Le REST repose sur un ensemble de normes, il est facile techniquement de passer outre, mais vous ôterez tout intérêt à votre API. Surtout si celle-ci est publique.

Le corps de la méthode se contente de récupérer la partie authors du fichier json et de la renvoyer au client. [4]



4

Vous pouvez donc obtenir la liste complète des auteurs. Sur la capture, vous apercevez les auteurs d'id 4 et 5 à savoir Enki Bilal et Franck Miller (tous les exemples donnés ici sont issus de ma BDthèque personnelle).

Bien sûr si vous ne voulez récupérer qu'un seul auteur en fonction de son id, le code sera légèrement différent. Pour cela il suffit d'ajouter une nouvelle route à Express :

```
// Création de la route pour consulter un auteur à partir de son id.
app.get('/authors/:id', function(req, res) {
  var id = req.params.id;
  var result = searchParams(db, 'authors', 'id', id);
  if (result) {
    res.json(result);
  } else {
    // Si la liste est vide on renvoie un 200 avec un message d'explication
    res.json(errors['empty_list_authors']);
  }
});
```

La méthode searchParams est purement utilitaire et renvoie la partie de la base de données qui nous intéresse. La partie cruciale est le passage de l'id par le chemin de l'API et la récupération par l'intermédiaire du code :

```
var id = req.params.id
```

A noter que la base de données errors contient le type d'erreur possible et renvoie le flux Json correspondant. [5]

Nous avons donc vu ici comment récupérer l'ensemble des auteurs présents en base puis comment récupérer un auteur de façon unitaire. Mais pour cela vous avez vu que nous avons dû saisir les routes pour chacun et ainsi parcourir du code afin de connaître les routes à utiliser. Il est maintenant grand temps d'introduire un peu d'hypermédia dans notre API.

5

Le domaine d'HATEOAS

La maturité d'une API correspond à la manière dont les contraintes de l'architecture REST sont respectées. Cette maturité peut être représentée par un modèle en quatre niveaux. Il s'agit du modèle de maturité de Richardson.

L'idée de cet article n'est pas de vous expliquer ce modèle (pourtant crucial pour comprendre les API) mais disons qu'en résumé les quatre niveaux correspondent à ceci :

- Le protocole d'échange (en général HTTP) sert juste de tunnel. Il fait passer les requêtes et les réponses à travers ce tunnel sans utiliser ce protocole pour indiquer l'état de l'application. C'est le niveau 0 ;
- La notion de ressource doit être introduite au sein de l'API. En d'autres termes, il faut rationaliser son API. C'est le niveau 1 ;
- L'emploi des verbes HTTP et des codes retours doit se faire de façon judicieuse. Chaque verbe et chaque code retour a une signification précise et doit être utilisé intelligemment. C'est le niveau 2 ;
- L'introduction de l'Hypermedia apporte souplesse et puissance à son API. C'est le niveau 3.

Ce découpage est résumé par l'image suivante : [6]

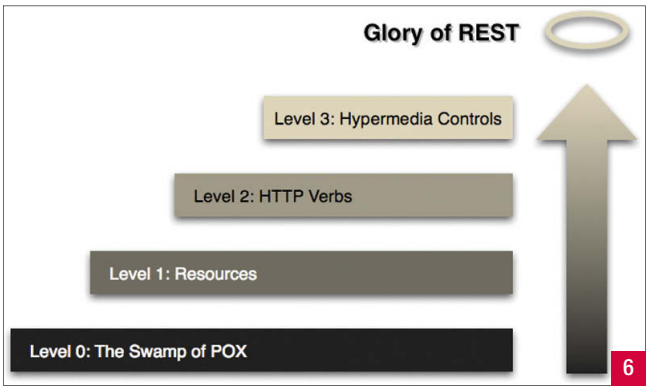
Le niveau 0 de Richardson est à proscrire pour vos API dans la mesure où l'on n'utilise pas le protocole de façon complète. De plus, l'absence d'utilisation de verbe et de découpage en ressource, rend votre API difficilement utilisable par vos clients. Actuellement en 2016, votre API doit au minimum respecter le niveau 2 de Richardson. Mais ici dans cet article nous allons viser le niveau 3 et comprendre ce qu'implique l'acronyme HATEOAS (*Hypertext As The Engine Of Application State*).

Ce mot compliqué recouvre une notion toute simple, mais extrêmement puissante. Il s'agit d'amener le concept de lien au sein de votre API. Et là vous avez complété de vous-même ; on ajoute donc des liens hypermédias dans nos réponses. Pour comprendre la puissance d'un tel concept, il suffit d'imaginer une page Web sans aucun lien hypertexte. Vous êtes bien embêté pour naviguer. Grâce à ces liens vous allez pouvoir maintenant naviguer dans votre API comme sur un site Web.

L'importance de l'Hypermédia est de rendre une API découvrable et navigable de façon autonome. Cela va donc rendre votre API plus flexible dans la mesure où le client sera en capacité à partir d'un lien unique de naviguer dans toutes vos APIs.

Il s'agit là d'un tour très bref de la théorie autour des API REST. La notion d'hypermédia est assez vaste pour y consacrer un livre en entier. Mais pour l'article, nous allons faire court. Vous retrouvez un ensemble de liens en fin d'article, qui récapitulent les normes et bonnes pratiques autour de REST. Mais l'idée ici est de coder notre API. Alors, maintenant codons !

Suite dans le prochain numéro



Tout savoir sur le langage **Kotlin**

David Wursteisen
SOAT

"One More Thing", c'est par la célèbre phrase de Steve Jobs que Stephanie Saad Cuthbertson, Product Manager pour Android, conclut son intervention lors de la conférence Google IO 2017 qui s'est tenue le 17 mai 2017 à Mountain View. Elle annonce alors l'officialisation du support du langage Kotlin pour la plateforme Android. S'ensuit une explosion de joie dans l'amphithéâtre et sur les réseaux sociaux. Pourquoi tant d'effervescence pour un produit dont Google n'est même pas l'auteur ?

Jetbrains, éditeur logiciel Tchèque, spécialisé dans les outils logiciels, est face à un dilemme : ses logiciels sont développés en Java mais ce langage ne leur semble plus adapté face à leurs envies d'expansion : Java est considéré comme trop verbeux et par ses contraintes, trop limitant. JetBrains a besoin d'un langage plus productif. Les langages alternatifs de l'époque n'ayant pas les fonctionnalités recherchées par les développeurs de chez JetBrains, ils prennent le pari de créer leur propre langage, qu'ils présenteront pour la première fois en Juin 2011 lors du JVM Language Submit : Kotlin.

Kotlin est un langage qui compile pour la JVM et est compatible avec Java. Dans ce contexte, vos bibliothèques Java sont utilisables directement depuis Kotlin et vous pouvez exécuter vos programmes écrits en Kotlin sur la JVM. Kotlin peut alors être vu comme une alternative directe du langage Java. Certains voient même en Kotlin un concurrent direct du langage d'Oracle.

La JVM n'est pas l'unique plateforme que Kotlin cible. Ce langage peut compiler pour produire du code Javascript. Votre code s'exécutant alors dans votre navigateur ou dans un moteur Javascript. Dernièrement, JetBrains a annoncé la possibilité de compiler Kotlin en "natif", rendant la JVM facultative dans l'exécution de votre code. Cela annonce la volonté de compiler pour d'autres plateformes et dont on se doute qu'iOS fera sûrement partie.

L'annonce de Google n'a pas été que Kotlin puisse maintenant compiler et s'exécuter sur la plateforme Android; c'était déjà le cas. Mais c'est bien une garantie par Google, qui est derrière Android, que le langage continuera à pouvoir être exécuté sur sa plateforme mobile. Les développeurs Android se voient donc rassurés : Kotlin est un langage fait pour durer sur leur plateforme préférée.

Nouvelle syntaxe

Vous pouvez tester les exemples suivant par vous-même directement depuis http://try.kotlinlang.org. Le site propose un éditeur en ligne capable d'exécuter du code Kotlin.

Kotlin se veut compact, expressif et simple : pour cela, JetBrains s'est inspiré de ce qui marche dans d'autres langages déjà existants. Si vous êtes utilisateurs de différents langages tels que C# ou Scala, vous reconnaitrez sans doute quelques caractéristiques propres à ces langages. Pour être plus concis, Kotlin vous évite l'utilisation redondante de certains mots clés ou caractères : dites au revoir au mot clé new ou encore aux points virgules.

Hello Kotlin !

```
fun main(args: Array<String>) {
    val language: String = "Kotlin"
    println("Bonjour $language!")
}
```

var et val sont deux nouveaux mots clés introduits par Kotlin. var est un raccourci pour variable : comme son nom l'indique, votre variable est modifiable. val quant à lui est un raccourci pour value : vous ne pourrez pas changer la référence. Utiliser le mot clé val est un petit peu comme utiliser le mot clé final sur un champ ou une variable en Java.

Val versus Var

```
fun main(args: Array<String>) {
    // Déclaration d'une variable
    var variable: String = "Never gonna give you up, "
    variable = "never gonna let you down"

    // Déclaration d'une valeur
    val value: String = "Never gonna run around and desert you"
    // le code ci-dessous ne compilera pas :
    // value = "rick rolled"

    println(variable)
    println(value)
}
```

Un autre point marquant entre Java et Kotlin est la déclaration de paramètre et de variable. Ici, le nom précède le type, contrairement en Java.

```
// Méthode Java
public int method(String arg) {
    // ...
}
// Méthode Kotlin
fun method(arg: String): Int {
    // ...
}
```

Pour rendre le code plus compact, lors de la déclaration d'une variable, le typage est optionnel ! Kotlin peut déduire le type de cette variable, via le mécanisme d'inférence de type.

Inférence de type

```
fun helloWorld(): String {
    return "hello world"
}

fun main(args: Array<String>) {
    // Typage explicite de type Int
    val entier: Int = 12345
```

```
    // Inférence de type de type Int
    var entier2 = 67890
    // Inférence de type à partir du type de retour d'une fonction (ici `String`)
    val str = helloWorld()
}
```

Contrairement à Java, Kotlin n'impose pas une classe publique par fichier. Java impose qu'une classe soit dans un fichier ayant le même nom que cette classe. Sans cette restriction, vous pouvez facilement, au sein d'un même fichier, faire cohabiter une interface et ses implémentations.
Autrement, la structuration d'une classe en Kotlin est proche de ce qui se fait en Java : on peut y déclarer des propriétés et des méthodes.

Exemple de déclaration d'une classe Kotlin

```
class MyFirstClass {

    // Déclaration d'une propriété
    val prefix: String = "Current value: "

    // Déclaration d'une propriété avec inférence de type
    var counter = 1

    // Déclaration d'une méthode ayant le type de retour String
    fun inc(): String {
        counter++
        return prefix + counter
    }
}

fun main(args: Array<String>) {
    val result = MyFirstClass().inc()
    println(result)
}
```

Java 8 apporte le support des fonctions anonymes (lambdas). On retrouve ces lambdas en Kotlin, avec une syntaxe toutefois légèrement différente. Si une lambda prend un paramètre, il n'est pas obligatoire de le spécifier : ce paramètre sera directement utilisable à travers une variable disponible nommée it. L'utilisation de method reference est également possible : le développeur Java 8 ne sera donc pas perdu !

Lambda

```
val printStr = { str: String -> print(str) }
listOf("hello", "programmez", "!")
        .map({it.toUpperCase()})
        .map(String::toLowerCase)
        .forEach(printStr)
```

Kotlin reste proche de Java en termes de syntaxe : seuls les éléments pouvant être considérés comme superflus sont supprimés. Cette syntaxe allégée est en avance sur son temps : Java 10 veut aussi s'alléger en commençant par reprendre le concept d'inférence de type, avec un mécanisme très proche de Kotlin.
Le langage s'efforce de garder la même philosophie que Java en restant simple d'utilisation. Mais la différence entre Java et Kotlin n'est pas qu'uniquement une affaire de syntaxe mais bien de fonctionnalités. Fonctionnalités qui seront vues par la suite.

Nouveaux types

Au-delà de ces changements syntaxiques, Kotlin apporte de nouveaux types. Contrairement à Java, ici, tout est *vraiment* objet. De base, on manipulera des Double, des Float, des Long, des Int, des Byte ou encore des String. Kotlin n'utilise pas de types primitifs (double, float, … propres à Java).
En plus de ces types, dits de base, Kotlin vient avec les types Any, Unit, Nothing. Mais à quoi servent-ils ? Dans la hiérarchie de type, tous les types héritent de Any. Any joue le même rôle que Object en Java. Quant à Unit, c'est un type - et un objet - représentant l'absence de résultat. C'est d'ailleurs ce type qui est par défaut retourné par toutes les fonctions que vous déclarez. Unit peut être grossièrement rapproché de void en Java.

Type Unit

```
fun neo(): Unit {
    println("Why do my eyes hurt?")
    // Unit est implicitement retourné
}

fun morpheus(): Unit {
    println("You've never used them before.")
    return Unit // utilisation de Unit explicite
}
```

Le dernier type, Nothing, est utilisé dans les cas où *aucune* valeur n'est retournée, par exemple quand une méthode lance systématiquement une exception ou exécute une boucle infinie.

Type Nothing

```
fun TODO(msg: String): Nothing {
    throw NotImplementedError(msg)
}

fun fibonacci(n: Int): Int {
    TODO("implement me")
}
```

Kotlin propose une déclinaison de tous ces types de base : les types *nullables*. Aucun des types précédents n'acceptent la valeur null. Le compilateur sera en erreur par exemple avec le code suivant :

```
val name: String = null
```

Le langage essaye de minimiser l'utilisation de valeur null ou sinon de l'ex-

Interopérabilité entre Java et Kotlin

Kotlin est parfaitement interopérable avec Java. Vous pouvez appeler du code Java depuis Kotlin, et inversement. Toutefois, dans ce dernier cas, quelques adaptations sont réalisées par Kotlin. Ainsi, le type Kotlin Int se verra transformé, selon la situation, en int ou Integer en Java. De même, un object Kotlin sera représenté via le design pattern Singleton en Java. Vous n'aurez, évidement, pas accès à toutes les fonctionnalités propres à Kotlin, depuis votre code Java, comme les paramètres nommés. Pour une interopérabilité avancée, Kotlin met à disposition certaines annotations (@JvmOverloads,@JvmStatic, @Throws, …) pour indiquer explicitement au compilateur Kotlin comment convertir certaines parties de votre code pour Java.

pliciter. Ainsi, si null est une valeur possible, vous devez utiliser un type *nullable*, qui est représenté par le type suivi d'un point d'interrogation :

```
val name: String? = null
```

Ainsi, à partir du type, vous saurez si la valeur null est acceptée ou non et ainsi éviter l'erreur *à un milliard de dollars* : la NullPointerException. On pourrait rapprocher un type *nullable* au type Optional de Java 8. Pourtant, il n'en est rien : un type *nullable* n'offre pas les méthodes de transformation de Optional mais plutôt une syntaxe qui lui est propre. À l'utilisation, cette syntaxe peut prêter à débat : en effet, pour accéder par exemple à un champ d'un objet, le champ peut être accédé en utilisant le symbole '?', transformant ainsi le type de ce champ en *nullable*. Mais, à moins d'utiliser l'opérateur !!, Le type *nullable* se propage dans un arbre d'objet où la base de cet arbre est *nullable* : tout le monde n'appréciera pas…

```
val adam:Human? = createAdam()
val firstname:String? = adam?.identity?.name?.firstname
val lastname:String = adam!!.identity.name.lastname
```

Ces nouveaux types existent pour éviter les confusions entre objet et type primitifs, simplifier l'écriture de fonctions, fonctions retournant systématiquement un type, même implicite. Mais ces types servent aussi à exprimer vos intentions : votre API accepte-t-elle une valeur null ou non ?

Nouvelles collections

Kotlin propose également une nouvelle API de collections. Cette API expose deux types de collections : celles immuables, et celles mutables. Une liste immuable est une liste où il n'est pas possible d'ajouter de nouveaux éléments dans cette même liste. Vous ne trouverez pas de méthode add sur ces listes.

List immuable

```
fun main(args: Array<String>) {
    // Création d'une liste immuable
    val elts: List<String> = listOf("un", "deux", "trois")
    // Création d'une nouvelle liste avec un élément supplémentaire
    val newElts: List<String> = elts + "quatre"
    newElts.forEach(::println)
}
```

Pour ajouter un élément dans une liste, il faut créer une nouvelle instance avec les éléments de notre liste, et le nouvel élément. Cela peut paraître fastidieux, pourtant cette immuabilité est idéale dans un contexte fortement concurrentiel. L'utilisation de l'opérateur + rend lisible et simplifie une telle opération. L'utilisation de cet opérateur entre deux listes est possible grâce à la surcharge d'opérateur. Dans d'autres langages, cette surcharge a souvent mauvaise presse car rendant la programmation trop "magique". Pour éviter cette déconvenue, seul un ensemble restreint d'opérateurs peuvent être surchargés.

List mutable

```
fun main(args: Array<String>) {
    val elts: MutableList<String> = mutableListOf("un", "deux", "trois")
    elts.add("quatre") // ajout d'un élément dans une liste mutable
    elts.map(String::toUpperCase) // Exemple d'utilisation de l'API de collection
            .flatMap({ str -> str.asIterable() })
            .sorted()
            .distinct()
            .forEach(::println)
}
```

Immuabilité et Data class

La propriété d'un objet immuable est de ne pas changer d'état. Ce concept peut facilement être mis en pratique grâce aux Dataclass. Une Dataclass est une classe dont la principale caractéristique est de gérer uniquement de la donnée. Pour transformer une classe en Dataclass, elle doit être marquée avec le mot clé data. Suite à quoi Kotlin va automatiquement générer pour cette classe les méthodes equals,hashcode, la méthode toString mais surtout la méthode copy. Grâce à cette méthode, vous pouvez créer une copie altérée d'un objet, vous permettant d'utiliser la technique du "copy-on-write" qui tend à simplifier les problèmes d'accès concurrents.

```
data class Sprite(val name: String, val x: Int, val y: Int)
val spr1 = Sprite("hello", 320, 240)
val spr2 = spr1.copy(name = "copy of hello")
```

Les collections Kotlin exposent la majorité des méthodes venant des streams Java 8 et y ajoutent quelques méthodes (par exemple zip qui permet de faire de la composition entre deux listes). Si vous êtes développeurs Android, via Kotlin, vous aurez accès à toutes ces méthodes de manipulation de liste qui ne sont pas encore mises à votre disposition.

Nouvelles fonctionnalités

Des valeurs par défaut peuvent être utilisées en Kotlin. Utiliser une valeur par défaut dans un constructeur évite de devoir créer des variations de ces constructeurs avec des paramètres en plus ou en moins. Kotlin propose également la notion de paramètre nommé. Lors de l'appel d'une méthode, d'un constructeur, vous pouvez explicitement spécifier un paramètre par son nom. Cette fonctionnalité est idéale pour éviter des paramètres qui s'enchaînent sans savoir lequel correspond à quoi.

```
class Color(val name: String = "white",
            val red: Boolean=false,
            val green: Boolean=false,
            val blue: Boolean=false)

fun main(args: Array<String>) {
    val white = Color("still white")
    val black = Color("black", true, true, true)
    val blue = Color(name = "blue", blue = true)
    val yellow = Color(name = "yellow", red = true, green = true)
}
```

Quand on utilise une bibliothèque, quand on manipule un objet métier venant d'une API, il manque toujours une méthode sur ces objets : celle qui serait tellement pratique pour nous si elle existait. Pour pallier ce manque, généralement on crée alors sa classe utilitaire qui "simule" cette méthode. Kotlin résout le problème en apportant le système de méthode d'extensions qui existent en Swift et en C#.

```
fun Float.round(): Int {
    return Math.round(this)
}
    val round = 32.2f.round()
    println(round)
```

Une méthode peut être ajoutée à un type existant, n'importe lequel. Il devient facile d'ajouter cette fameuse méthode qui manquait sur cet objet

métier. Attention tout de même : depuis cette méthode d'extensions, vous n'aurez accès qu'aux éléments auxquels vous pouvez déjà accéder. Si vous pensiez pouvoir manipuler des propriétés ou des méthodes privées, cela ne sera toujours pas possible.
Dans la bibliothèque standard de Kotlin, on retrouve l'objet Pair qui permet d'associer deux objets. Pour utiliser ces deux objets, contenu dans notre Pair, on doit soit systématiquement l'utiliser, soit extraire ces objets, en déclarant deux variables. Opération d'autant plus fastidieuse quand cette paire est retournée par une fonction.

```
val coord: Pair<Double, Double> = randomGeo()
// extraction manuelle
val x1 = coord.first
val y1 = coord.second

// destructuring declarations
val (x2, y2) = randomGeo()
```

Les déclarations déstructurées (Destructuring Declarations) ont justement comme intention de simplifier cette mécanique. Kotlin va directement extraire des propriétés de vos objets et assigner ces propriétés aux variables voulues. C'est directement utilisable avec l'objet Pair, avec vos Dataclass et vous pouvez également implémenter cette mécanique sur vos propres classes. Enfin, la déclaration déstructurée fonctionne également avec les listes : idéal pour accéder directement aux premiers éléments de votre liste.

```
val (elt1, elt2) = listOf("first", "second", "third")
```

Nouvel écosystème

JetBrains édite IntelliJ. On pourrait croire, à juste raison, que seul IntelliJ supporte Kotlin. Pourtant, il n'en est rien : des plugins existent pour Eclipse et Netbeans. Ces plugins sont créés par JetBrains et sont entièrement fonctionnels. L'auto complétion est opérationnelle, tout comme l'exécution et le debug de code Kotlin. Le support dans IntelliJ et sa déclinaison Android Studio reste tout de même supérieur : régulièrement mis à jour, ce plugin est capable d'afficher le bytecode généré à partir de code Kotlin et propose même de générer ce bytecode en "pseudo" Java. C'est idéal pour comprendre les conversions réalisées par le compilateur pour rendre le code compatible avec Java. Aucun outil de build n'est mis en avant par rapport à un autre : que vous soyez un utilisateur de Maven ou de Gradle, les deux outils proposent des plugins pour qu'ils compilent votre code Kotlin. C'est au-delà de ces outils qu'on détecte un certain manque au niveau de l'écosystème. Par exemple, l'analyseur de code SonarQube ne supporte pas encore Kotlin. Mais ce manque est surtout dû à la jeunesse de l'écosystème et il devrait se combler avec le temps. Surtout que d'autres sociétés préparent le terrain pour Kotlin. Pivotal a annoncé que la prochaine version de son framework Spring sera compatible, de base, avec Kotlin.

Votre nouveau langage ?

Si vous êtes développeur Android, vous pouvez intégrer dès aujourd'hui le langage dans vos projets : il est officiellement supporté par Google et cohabite très bien avec Java. Nul besoin de passer tout votre projet en Kotlin pour déjà l'intégrer. Sur les autres plateformes, la question se pose surtout pour la JVM : Kotlin remplacera t'il Java ? Oracle a pris du retard avec Java 9, et Java 10 propose des fonctionnalités qui sont pour certaines déjà présentes dans Kotlin. Aurez-vous la patience d'attendre ? Si ce n'est pas le cas : pourquoi ne pas vous laisser séduire par Kotlin ? •

LA PARTIE OUTILLAGE

L'ensemble des outils disponible pour Kotlin est restreint : oubliez analyseur de code, et autres. Les outils actuellement disponibles sont des plug-ins pour IDE ou pour des outils de build pour compiler le code.
Les différents IDE (IntelliJ, Android Studio, Eclipse, Netbeans) supportent Kotlin via un plug-in. Par défaut, IntelliJ et Android Studio ont déjà le plug-in Kotlin d'installé. Pour Netbeans et Eclipse, il sera nécessaire d'installer le plug-in que ce soit via le gestionnaire de plug-in pour Netbeans ou la marketplace pour Eclipse. À noter que l'installation du plug-in peut se faire depuis le github même du projet : un glissé déposé d'un bouton dans Eclipse suffit à faire l'installation. Facile, simple et efficace !
Ces plug-ins apportent la coloration de la syntaxe, l'auto complétion, etc. Le niveau de fonctionnalité dépend de l'IDE que vous utilisez, mais les fondamentaux sont systématiquement présents.
L'utilisation de Kotlin avec votre outil de build est très simple : l'ajout d'un plug-in dans la chaîne de compilation ajoute le support de Kotlin. Nul besoin d'installer d'autres outils : c'est votre outil de build qui téléchargera les éléments nécessaires à la compilation de votre code en Kotlin.
Ainsi, avec Gradle, il suffit d'ajouter le plug-in Kotlin puis de l'activer (cf exemple ci-dessous). Ensuite, ajoutez votre code dans un fichier avec une extension kt, dans le répertoire src/main/kotlin ou même dans le répertoire src/main/java : les deux répertoires sont par défaut supportés. Puis compiler avec votre outil de build votre code.

```
buildscript {
    ext.kotlin_version = « <version to use> »

    repositories {
        mavenCentral()
    }

    dependencies {
        classpath "org.jetbrains.kotlin:kotlin-gradle-plugin:$kotlin_version »
    }
}
```

Activation du plug-in pour utiliser Kotlin sur la JVM :

```
apply plugin: "kotlin"
```

Activation du plugin pour utiliser Kotlin sur Android :

```
apply plugin: 'com.android.application'
apply plugin: 'kotlin-android'
```

Pour compiler du code Kotlin via votre chaîne d'intégration, cela se fait donc très simplement : l'ajout du plug-in Kotlin dans votre outil de build va s'occuper de tout. Nul besoin de configurer d'autres outils, ou d'installer des composants sur votre serveur.
La liste d'outil supportant Kotlin est certes petite, mais de qualité. Grâce à eux, l'intégration du langage dans votre flux de travail habituel se fait simplement et doucement : c'est le plus important.

Pour des **données JSON** toujours correctes : les schémas de validation.

Fabien Pigere
freelance html5

A l'heure actuelle, JSON s'impose comme format d'échange simple, lisible et léger entre applications. A l'origine prévu pour le Javascript comme le rappelle son nom (JavaScript Object Notation), il a, depuis, atteint une notoriété autre, existant dans tous les langages que cela soit du C++, du PERL, ou bien du PHP !

Ce format a fait sa place au détriment de XML, cependant le XML gardait une longueur d'avance dans un domaine : la validation par schéma (XSD). L'Internet Engineering Task Force a pensé à nous, et permet enfin la validation facile de données JSON grâce à un puissant système, qui reste cependant lisible sur le plan humain.Voyons comment ça marche !

Les schémas JSON, la base

Un schéma JSON est un objet JSON qui permet de valider les données; on pourrait presque parler de «meta json». Le plus petit schéma, est l'objet vide "{}". Les schémas JSON sont donc limités par les types de bases d'origine du JSON, qui sont les nombres (entier ou flottant cependant), les chaînes de caractères, les tableaux, les objets.
Commençons par un schéma d'utilisateur lambda :

```
{
"title" : "un utilisateur",
"type" : "object",
"properties": {
                "nom": {"type": "string"},
                "prenom": {"type": "string"},
                "password": {"type": "string"}
            },
}
```

Comme nous le voyons dans cet exemple de base, le schéma définit un objet qui a lui-même 3 champs. A cette étape cependant, la validation réussit avec des cas simples comme " {} ".En effet par défaut, une propriété n'est PAS obligatoire. Remédions à cela en rajoutant un champs de type "required":

```
{
...,
required:["nom","prenom","password"]
}
```

A cette étape, les 3 champs sont devenus obligatoires. Cependant, classiquement, le password doit avoir une taille minimum et maximum. On rajoute donc encore des contraintes :

```
"password": {"type": "string", "minLength":8,"maxLength":16}
```

De la même façon rajoutons une email valide :

```
"email":{"type": "string ", "format" : "email "}
```

Comme nous le voyons, cela reste lisible tout en étant puissant. Résumons les principales propriétés d'un schéma résumé dans un tableau :

Type	Option	Description
string		Une chaîne de caractères.
	minLength	La taille minimum acceptée.
	maxLength	Maximum possible.
	pattern	Une regex de validation, au format JS.
	format	Un format prédéfini (email, ipv4, ipv6, ...).
	enum	Tableau de choix prédéfinis.
integer		Un entier.
	minimum	Le plus petit nombre possible.
	maximum	Le plus grand.
	exclusiveMinimum	Le minimum est exclu.
	exclusiveMaximum	Idem.
	multipleOf	Le nombre doit être un multiple (par exemple un multiple de 10).
number		Un nombre.
	(voir integer)	
object		Un objet au sens JSON, contenant donc des strings, nombres, tableaux ou objets.
	properties	Objet contenant les propriétés attendues. ex : "properties": { nom": {"type": "string"}, "prenom" :{"type":"string"}}
	required	Tableau de strings indiquant les champs obligatoires pour que l'objet soit valide.
	additionalProperties	True ou false, permet de définir si l'objet peut avoir des propriétés supplémentaires que les properties définissent.
array		Tableau d'élément.
	items	Tableau d'éléments attendus.
	minItems	Nombre d'élément minimum.
	maxItems	Maximum autorisé.
	uniqueItems	Tout les éléments du tableau doivent être différent.
anyOf		L'un des sous-schémas fourni est vrai (ou tous).
allOf		Tous les sous-schémas sont vrais.
oneOf		Un et seulement un des schémas est vrai.
not		Ne correspond pas au sous-schéma fourni.

Un exemple ? Un exemple !

Nous allons définir un schéma permettant de définir un produit, Ce produit aura un prix, des "tags" permettant de le définir pour un moteur de recherche, et un endroit de fourniture facultatif :
code complet sur wwwprogrammez.com

Cela amène quelques remarques :

Le champs tag est nécessairement de minimum 3 lettres, la propriété "minItems" mise à 1 permet d'avoir obligatoirement un exemplaire, et

"uniqueItems" permet de dire que tous les tags devront être différents. La propriété $schema permet de définir le type de schéma, dans notre cas, c'est la version 4. La dimension du produit est facultative MAIS si l'un des champs est rempli, ils devront tous l'être, de même pour la Localisation du produit. Nous avons donc la possibilité de définir des sous-schémas et de les réutiliser ensuite. Comme nous le voyons, ceci est très complet, voire complexe, tout en étant lisible par un humain.
Les datas suivantes sont donc valides :

```
[
{"id":1,"name":"test","price":1,tags:["abc"],Location:{"latitude":1,"longitudr":2}},
{"id":2,"name":"test","price":1,"tags":["abb","abc"]}
]
```

Maintenant que nous avons un schéma et des datas, validons tout ça ! La page de référence [1] contient la librairie qui correspond à votre langage, le choix est vaste ! Pour ma part, je vais commencer par une validation coté serveur en NODE, puis coté client en Javascript, mais je vous rappelle que ces schémas ne sont pas réservés au Web.

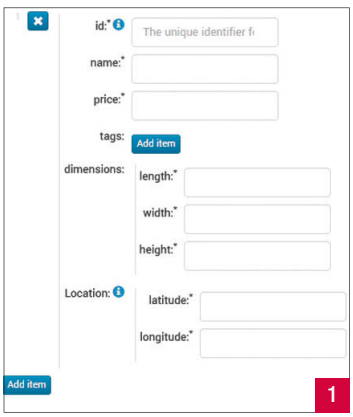
Coté Node:
Nous allons utiliser la librairie ajv[2] pour valider nos données :

```
npm install ajv

Puis le simple code suivant :
var Ajv = require('ajv');
var ajv = new Ajv();

var schema={...};
var data = [...];

var validate = ajv.compile(schema);
var valid = validate(data);
if (!valid)
   console.log(validate.errors);
else
   console.log("Données valides");
```

Coté client : Le code de validation est quasiment le même, il suffit d'inclure la librairie AJV, par exemple grâce à un CDN [3], mais je voudrais parler d'une librairie expérimentale qui permet de générer l'IHM à partir d'un schéma, la libraire JSON-FORMS [4]. Il faut pour cela inclure la librairie, et ayant personnellement un petit faible pour bootstrap, le plug-in bootstrap de la librairie. Une fois cela fait, la librairie s'initialise de cette façon :

```
   var BrutusinForms = brutusin["json-forms"];
   var bf = BrutusinForms.create(schema);
   var container = document.getElementById('container');
   bf.render(container, {});
// une fois le formulaire validé par l'utilisateur :
      var d = bf.getData();
      var b = bf.validate();
```

Le schéma rendu est sur la Figure 1, contenant des tooltips basés sur la description du champ, des input du type correct si le navigateur supporte les input html5. Cette librairie est encore au stade expérimental; je vous recommande d'aller sur le site régulièrement pour vous tenir informé, l'auteur étant très réactif !



CONCLUSION

La norme JSONSCHEMA n'est pas encore finie, la version 5 commence à s'imposer et la version 6 devrait voir le jour bientôt. Cependant on peut d'ores et déjà l'utiliser pour valider les données. De nombreux outils existent qui facilitent la mise en œuvre rapide, et la génération d'IHM dans divers langages (voir [1]), ce qui permet un gain de temps évident pour le prototypage, ou pour les applications demandant beaucoup de saisie. Je vous conseille de garder un oeil sur cette technologie très prometteuse. •

[1] http://json-schema.org/implementations.html
[2] https://github.com/geraintluff/jsv4-php
[3] https://cdnjs.cloudflare.com/ajax/libs/ajv/4.11.4/ajv.min.js
[4] https://github.com/brutusin/json-forms

Créez votre propre AdBlocker

• Sylvain SAUREL
Développeur Java / Android
sylvain.saurel@gmail.com
https://www.ssaurel.com

Selon de récentes études, près d'un quart des Français ont recours à des bloqueurs de publicités, les fameux AdBlockers, lorsqu'ils surfent sur Internet que ce soit depuis leur ordinateur ou depuis leur smartphone. Au-delà de la question éthique soulevée par ce choix, nous allons nous intéresser dans cet article aux aspects techniques de telles solutions en vous proposant d'implémenter votre propre AdBlocker sous Android.

Les chiffres sont formels, les internautes, qu'ils soient Français ou d'ailleurs, ont une aversion de plus en plus importante pour les publicités. Il est vrai que beaucoup d'éditeurs ont poussé le bouchon trop loin en ne cessant de proposer des publicités de plus en plus intrusives et de plus en plus fréquentes sur leurs sites. De fait, les bloqueurs de publicité, plus connus sous le nom Anglais d'AdBlockers, ont connu un essor très important ces dernières années et de plus en plus d'internautes y ont recours. Dans cet article, il n'est pas question de se pencher sur la question éthique soulevée par ce choix mais plutôt de s'intéresser au fonctionnement technique de tels bloqueurs sous Android. Ainsi, nous allons réaliser une application Android permettant aux utilisateurs de visualiser des sites Internet privés de leurs publicités envahissantes. Notre solution vous permettra de consulter vos sites favoris sans aucune distraction extérieure et sera basée sur le fonctionnement suivant :

1. Définition d'une liste de noms de domaines servant des publicités en ligne ;
2. Chargement de la liste des noms de domaines servant des publicités en ligne au sein de l'application ;
3. Chargement d'un site Internet au sein du composant WebView proposé par le SDK Android en standard ;
4. Interception des requêtes émises lors du chargement du site Internet ;
5. Filtrage des requêtes émises pour ne conserver que les contenus ne provenant pas d'hôtes servant des publicités en ligne ;
6. Affichage de la page épurée à l'utilisateur final ;

Définition d'une liste de noms de domaines

Afin de constituer une liste de noms de domaines suffisamment large, nous allons nous appuyer sur le site Internet https://pgl.yoyo.org qui fournit une liste complète des domaines servant des publicités en ligne. Elle sera définie au sein d'un simple fichier texte où chaque ligne correspond à un domaine servant des publicités :

```
101com.com
101order.com
123found.com
180hits.de
180searchassistant.com
1x1rank.com
207.net
247media.com
```

Chargement de la liste des noms de domaines

Au sein de notre application Android, nous allons placer le fichier texte ainsi constitué dans le répertoire assets. Au lancement de l'application, il sera nécessaire de charger le contenu du fichier en mémoire au sein d'une collection de type Set afin d'éliminer les éventuels doublons contenus dans la liste. Pour ce faire, nous créons une classe AdBlocker au sein de laquelle nous définissons une méthode loadFromAssets. A l'intérieur de cette dernière, nous allons ouvrir le fichier contenu dans le répertoire assets puis le parcourir ligne par ligne en nous appuyant sur la bibliothèque open source Okio proposée par Square. Chaque élément du fichier sera ainsi chargé en mémoire au sein du Set AD_HOSTS lorsque cette méthode sera appelée. Il reste ensuite à nous assurer que ce chargement aura bien lieu au lancement de l'application afin que la liste soit tout de suite à disposition au moment de l'affichage d'une page Web. Dans ce but, une méthode init statique est définie au sein de la classe AdBlocker. Une bonne pratique va consister à réaliser l'exécution de la méthode loadFromAssets au sein d'un Thread dédié afin de ne pas bloquer le Thread principal de l'application en charge de l'affichage à l'écran. L'appel à loadFromAssets est donc réalisé au sein d'une AsyncTask dans la méthode init :

```
public class AdBlocker {
    private static final String AD_HOSTS_FILE = "ads_website.txt";
    private static final Set<String> AD_HOSTS = new HashSet<>();

    public static void init(final Context context) {
        new AsyncTask<Void, Void, Void>() {
            @Override
            protected Void doInBackground(Void... params) {
                try {
                    loadFromAssets(context);
                } catch (IOException e) {
                    // ...
                }
                return null;
            }
        }.execute();
    }

    private static void loadFromAssets(Context context) throws IOException {
        InputStream stream = context.getAssets().open(AD_HOSTS_FILE);
        BufferedSource buffer = Okio.buffer(Okio.source(stream));
        String line;

        while ((line = buffer.readUtf8Line()) != null) {
            AD_HOSTS.add(line);
        }

        buffer.close();
        stream.close();
    }
}
```

Enfin, il faut définir une classe MyApplication héritant de l'objet Application du SDK standard afin d'appeler la méthode init de la classe AdBlocker au chargement de l'application. Il reste à préciser au sein de l'Android Manifest que c'est bien notre classe MyApplication qui devra être appelée et non l'implémentation par défaut de la classe Application :

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
```

```
  package="com.ssaurel.adblocker">

  <uses-permission android:name="android.permission.INTERNET" />

  <application
    android:allowBackup="true"
    android:icon="@mipmap/ic_launcher"
    android:label="@string/app_name"
    android:roundIcon="@mipmap/ic_launcher_round"
    android:supportsRtl="true"
    android:theme="@style/AppTheme"
    android:name=".MyApplication">
    <activity android:name=".MainActivity">
      <intent-filter>
        <action android:name="android.intent.action.MAIN" />
        <category android:name="android.intent.category.LAUNCHER" />
      </intent-filter>
    </activity>
  </application>
</manifest>
```

Quant au contenu de la classe MyApplication, il aura la forme suivante :

```
package com.ssaurel.adblocker;

import android.app.Application;

public class MyApplication extends Application {
  @Override
  public void onCreate() {
    super.onCreate();
    AdBlocker.init(this);
  }
}
```

Chargement d'un site Internet au sein d'une WebView

Le SDK Android met à disposition la classe WebView afin de permettre aux développeurs de charger des pages Internet au sein de leurs applications. Nous allons donc nous appuyer sur cette classe pour créer notre interface utilisateur. Celle-ci sera simple puisque composée simplement d'un composant WebView prenant la totalité de l'espace disponible comme on peut le voir au sein du fichier layout activity_main.xml :

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
  android:layout_width="match_parent"
  android:layout_height="match_parent"
  tools:context="com.ssaurel.adblocker.MainActivity">

  <WebView
    android:id="@+id/webView"
    android:layout_width="match_parent"
    android:layout_height="match_parent"/>
</RelativeLayout>
```

Au niveau du code Java de l'activité principale, il faut dans un premier temps récupérer la référence de la WebView. Ensuite, nous associons un objet WebViewClient à cette dernière avant d'activer l'exécution de scripts Javascript sur les pages chargées. Enfin, la dernière étape consiste à charger le site Internet en passant son URL en paramètre de la méthode loadUrl de l'objet WebView :

```
package com.ssaurel.adblocker;

import android.support.v7.app.AppCompatActivity;
import android.webkit.WebViewClient;

public class MainActivity extends AppCompatActivity {

  private WebView webView;

  @Override
  protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);
    webView = (WebView) findViewById(R.id.webView);
    webView.setWebViewClient(new WebViewClient());
    webView.getSettings().setJavaScriptEnabled(true);
    webView.loadUrl("https://www.ssaurel.com/blog");
  }
}
```

L'exécution de notre application en l'état donne le résultat présenté à la figure 1 sur laquelle on constate qu'un bandeau publicitaire est bien affiché en haut de la page :

Interception et filtrage des requêtes émises

L'étape suivante est d'intercepter et de filtrer les requêtes émises par l'objet WebView afin de ne pas afficher le contenu provenant de domaines servant des publicités. Pour ce faire, nous allons nous appuyer sur la liste chargée précédemment en mémoire au sein de la classe AdBlocker.

La classe WebViewClient propose une méthode shouldInterceptRequest que nous allons surcharger afin de pouvoir intercepter et filtrer les requêtes émises par la WebView au chargement d'un site Internet. Au sein de cette méthode, nous allons définir une HashMap permettant d'associer une URL donnée à un booléen indiquant si celle-ci est autorisée à l'affichage ou non. Cela nous permettra d'éviter d'effectuer plusieurs fois une vérification pour une même URL.

Si une URL n'a pas encore été testée, nous allons appeler la méthode isAd de l'objet AdBlocker qui nous renverra true si l'URL passée en paramètre fait partie des domaines servant des publicités. Nous ajoutons le résultat de ce test au sein de l'objet HashMap pour indiquer son statut. Dans le cas où une URL a déjà été testée et est donc présente dans l'objet HashMap, nous récupérons son statut au sein de cette dernière. En fin de méthode, nous retournons l'objet renvoyé par l'appel à la méthode parente shouldInterceptRequest dans le cas où l'URL est autorisée afin de permettre son affichage de manière normale.

Dans le cas où il s'agit d'une URL destinée à afficher de la publicité, nous renvoyons une réponse Web vide via un appel à la méthode createEmptyResource de l'objet AdBlocker. Ceci nous donne le code suivant lors de la définition de l'objet WebViewClient de notre WebView :

```
webView.setWebViewClient(new WebViewClient() {
  private Map<String, Boolean> loadedUrls = new HashMap<>();

  @TargetApi(Build.VERSION_CODES.HONEYCOMB)
```

```
@Override
public WebResourceResponse shouldInterceptRequest(WebView view, String url) {
  boolean ad = false;

  if (!loadedUrls.containsKey(url)) {
    ad = AdBlocker.isAd(url);
    loadedUrls.put(url, ad);
  } else {
    ad = loadedUrls.get(url);
  }

  if (ad) {
    Log.i("MyAdBlocker", "Blocked Url Ad = " + url);
  }

  return ad ? AdBlocker.createEmptyResource() :
        super.shouldInterceptRequest(view, url);
 }
});
```

Dans ce code, nous avons appelé les méthodes statiques isAd et createEmptyResource de la classe AdBlocker. La dernière partie de notre travail va donc consister à les implémenter. La méthode isAd s'appuie sur le contenu du Set AD_HOSTS chargé précédemment. Afin de définir comment tester si une URL est liée à un domaine servant des publicités, considérons des publicités provenant du réseau Google Doubleclick par exemple. Celles-ci peuvent avoir plusieurs formes :

```
pubads.g.doubleclick.net
adclick.g.doubleclick.net
googleads.g.doubleckick.net
```

Or, dans notre Set, nous n'avons qu'une entrée liée au domaine doubleclick.net. Notre stratégie va donc consister à extraire le domaine d'une URL et à comparer le contenu de chaque chaîne de sous-domaines avec la liste des URLs non autorisées. Ce travail étant implémenté de manière récursive au sein d'une méthode statique isAdHost dans la classe AdBlocker que l'on complète de la sorte :

```
public static boolean isAd(String url) {
  HttpUrl httpUrl = HttpUrl.parse(url);
  return isAdHost(httpUrl != null ? httpUrl.host() : "");
}

private static boolean isAdHost(String host) {
  if (TextUtils.isEmpty(host)) {
    return false;
  }

  int index = host.indexOf(".");
  return index >= 0 && (AD_HOSTS.contains(host) | |
  index + 1 < host.length() && isAdHost(host.substring(index + 1)));
}
```

Il est bon de noter que nous tirons profit ici de la classe HttpUrl de la bibliothèque open source OkHttp mise à disposition par Square. Enfin, nous terminons en ajoutant la méthode statique createEmptyResource à la classe AdBlocker pour retourner un objet de type WebResource Response vide lorsqu'une requête émise visée à afficher de la publicité :

```
@TargetApi(Build.VERSION_CODES.HONEYCOMB)
public static WebResourceResponse createEmptyResource() {
  return new WebResourceResponse("text/plain", "utf-8", new ByteArrayInput
Stream("".getBytes()));
}
```

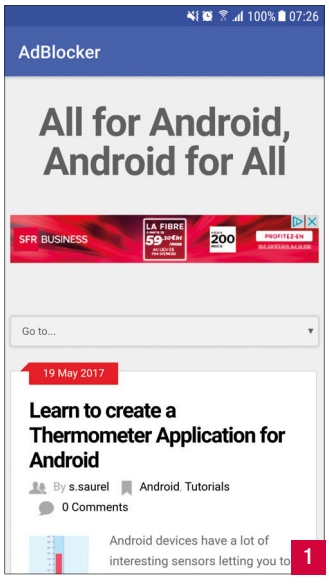
Affichage de la page épurée sans publicités

Le code de notre application implémenté, il est temps de tester l'affichage d'un site Web épuré sans aucune publicité. Le résultat final pouvant être visualisé à la figure 2.

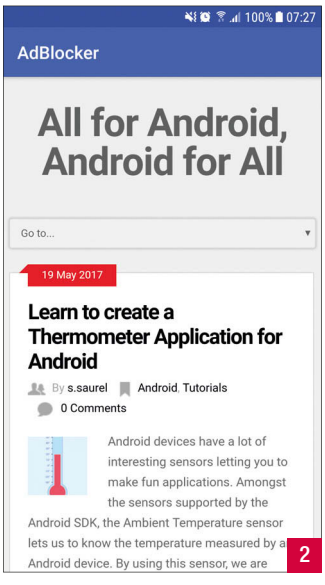
Comme nous avions pris soin d'afficher un message de log détaillant les URLs bloquées par notre implémentation de l'objet WebViewClient, nous pouvons visualiser ce qui a été intercepté et filtré par notre AdBlocker dans les messages de logs du périphérique cible (figure 3).

CONCLUSION

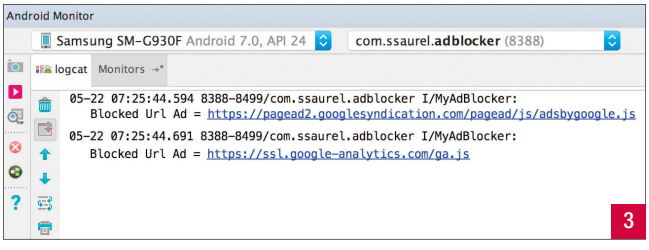
Cet article nous aura permis de voir avec quelle simplicité il est possible de créer un AdBlocker personnalisé sous Android en utilisant le composant WebView pour charger et filtrer certaines URLs servant des publicités. A partir du code de base présenté ici, il suffit d'ajouter une barre d'adresses, une barre de progression et quelques boutons standards proposés par les navigateurs Web pour obtenir une application Android de type navigateur garantie sans publicités. Succès garanti auprès des utilisateurs ! Enfin, une autre piste d'amélioration serait d'implémenter un chargement dynamique des URLs servant des publicités depuis un serveur Web par exemple afin de prendre en compte facilement les nouveaux serveurs pouvant apparaître dans le futur et d'en faire bénéficier les utilisateurs finaux sans mise à jour de l'application. •



Site internet affiché avec publicités

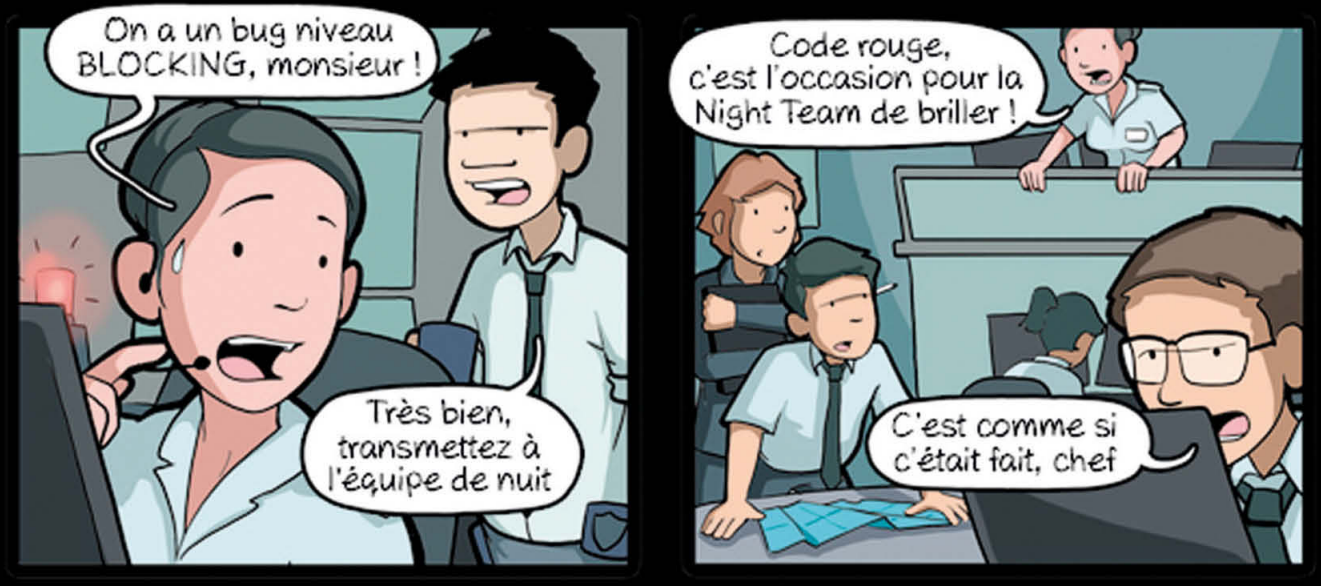


Notre AdBlocker en action

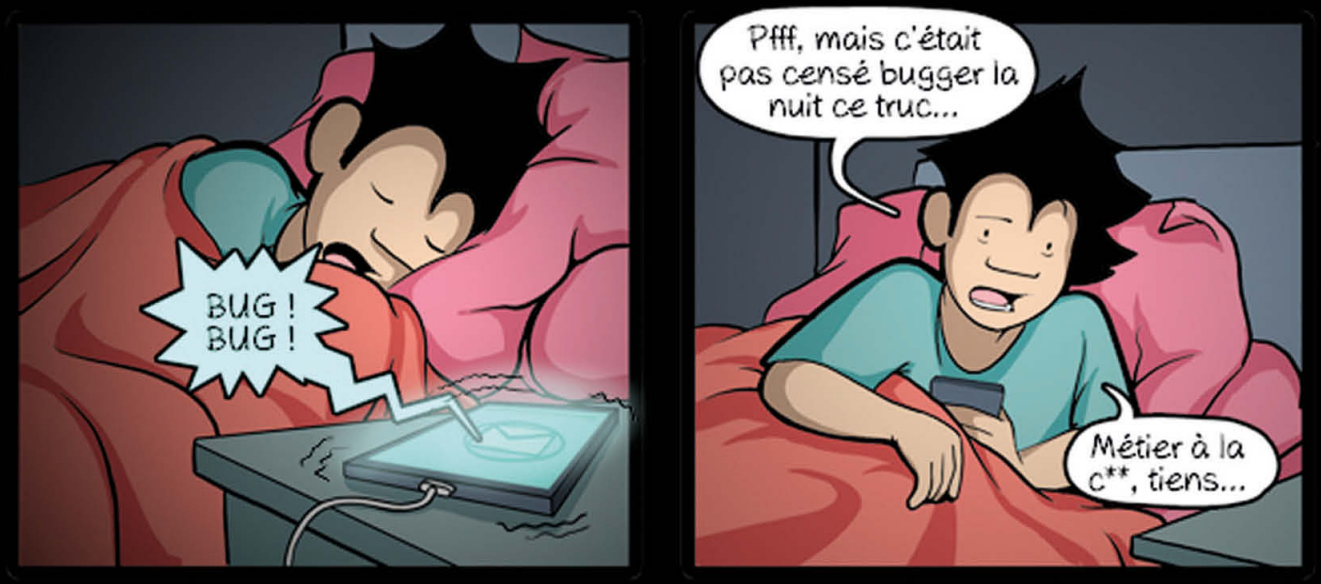


Visualisation des URLs filtrées

Comment les clients pensent que les urgences de nuit sont gérées



Comment les urgences de nuit sont gérées



Une publication Nefer-IT, 7 avenue Roger Chambonnet, 91220 Brétigny sur Orge - redaction@programmez.com
Tél. : 01 60 85 39 96 - Directeur de la publication & Rédacteur en chef : François Tonic
Secrétaire de rédaction : Olivier Pavie
Ont collaboré à ce numéro : S. Saurel
Nos experts techniques : M. Pointier, D. Dublin, A. Zanchetta, J-F Richard, F. Sages, G. Cottrez, C. Polonio, V. Bequart, P-A Chopin, H. Mroczkowski, M. Fertin, D. Briceno-Aguillera, E. Auberix, J-P Hallot, A. Chever, D. Wursteissen, F. Pigere, V. Piard, W. Moulin, M. Eglem J. Thiriet, A. Canut, M. Roussel, M. Grief, S. Warin.
Couverture : © querbeet - © Poppy Project - © sjharmon - **Maquette :** Pierre Sandré.
Publicité : PC Presse, Tél.: 01 74 70 16 30, Fax : 01 40 90 70 81 - pub@programmez.com.
Imprimeur : S.A. Corelio Nevada Printing, 30 allée de la recherche, 1070 Bruxelles, Belgique.
Marketing et promotion des ventes : Agence BOCONSEIL - Analyse Media Etude - Directeur : Otto BORSCHA oborscha@boconseilame.fr
Responsable titre : Terry MATTARD Téléphone : 09 67 32 09 34
Contacts : Rédacteur en chef : ftonic@programmez.com - Rédaction : redaction@programmez.com - Webmaster : webmaster@programmez.com -
Publicité : benoit.gagnaire@programmez.com - Evenements / agenda : redaction@programmez.com
Dépôt légal : à parution - Commission paritaire : 1220K78366 - ISSN : 1627-0908 - © NEFER-IT / Programmez, juillet 2017

Abonnement : Service Abonnements PROGRAMMEZ, 4 Rue de Mouchy, 60438 Noailles Cedex. - Tél. : 01 55 56 70 55 - *abonnements.programmez @groupe-gli.com* - Fax : 01 55 56 70 91 - du lundi au jeudi de 9h30 à 12h30 et de 13h30 à 17h00, le vendredi de 9h00 à 12h00 et de 14h00 à 16h30. **Tarifs** abonnement (magazine seul) : 1 an - 11 numéros France métropolitaine : 49 € - Etudiant : 39 € CEE et Suisse : 55,82 € - Algérie, Maroc, Tunisie : 59,89 € Canada : 68,36 € - Tom : 83,65 € - Dom : 66,82 € - Autres pays : nous consulter.
PDF : 35 € (monde entier) souscription sur www.programmez.com

Sur abonnement ou en kiosque

Le magazine des pros de l'IT

Mais aussi sur le web

Ou encore sur votre tablette

ikoula
HÉBERGEUR CLOUD

PRÉSENTE

CLOUD**IKOULA**ONE

Le succès est votre prochaine destination

MIAMI SINGAPOUR PARIS
AMSTERDAM FRANCFORT

*CLOUD**IKOULA**ONE* est une solution de Cloud public, privé et hybride qui vous permet de déployer **en 1 clic et en moins de 30 secondes** des machines virtuelles à travers le monde sur des infrastructures SSD haute performance.

www.ikoula.com sales@ikoula.com 01 84 01 02 50

ikoula HÉBERGEUR CLOUD

NOM DE DOMAINE | HÉBERGEMENT WEB | SERVEUR VPS | SERVEUR DÉDIÉ | CLOUD PUBLIC | MESSAGERIE | STOCKAGE | CERTIFICATS SSL