NOUVELLE FORMULE : + 16 PAGES & NOUVELLES RUBRIQUES !

GNU LINUX MAGAZINE / FRANCE

DÉVELOPPEMENT SUR SYSTÈMES UNIX, OPEN SOURCE & EMBARQUÉ

N°200
JANVIER 2017

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

L 19275 - 200 - F: 7,90 € - RD

Smart TV / Tizen

CRÉEZ UNE APPLICATION POUR VOTRE TV CONNECTÉE ! p.80

- Testez le Tizen TV SDK
- Développez votre jeu Flappy Bird like !
- Déployez votre application sur la télé !

Audio / JUCE

TRAITEZ DES DONNÉES SONORES EN C++ p.48

Hack / evdev

DÉVELOPPEZ VOTRE CLAVIER PROGRAMMABLE p.66

Cartographie / QGis

DIFFUSEZ VOS DONNÉES GÉORÉFÉRENCÉES SUR LE WEB p.12

Système / init

DÉCOUVREZ LES FACES CACHÉES DE SYSTEMD p.24

Embarqué / IoT

EXPLOREZ LA PARTIE MIKROBUS DE LA WARP7 p.32

Interfaces graphiques avec WxPython – Comprenez et bloquez les failles CSRF...

ikoula
HÉBERGEUR CLOUD

VOTRE SERVEUR DÉDIÉ XEON® À LA CARTE

OFFRE DÉCOUVERTE -50 %*

DOMAINE ▼ | HÉBERGEMENT ▼ | SERVEUR DÉDIÉ | SERVEUR VIRTUEL VPS ▼ | CLOUD ▼

CODE PROMO
SERV50

14,99 €*
~~29,99 €~~

SETUP **10 €**
~~60 €~~

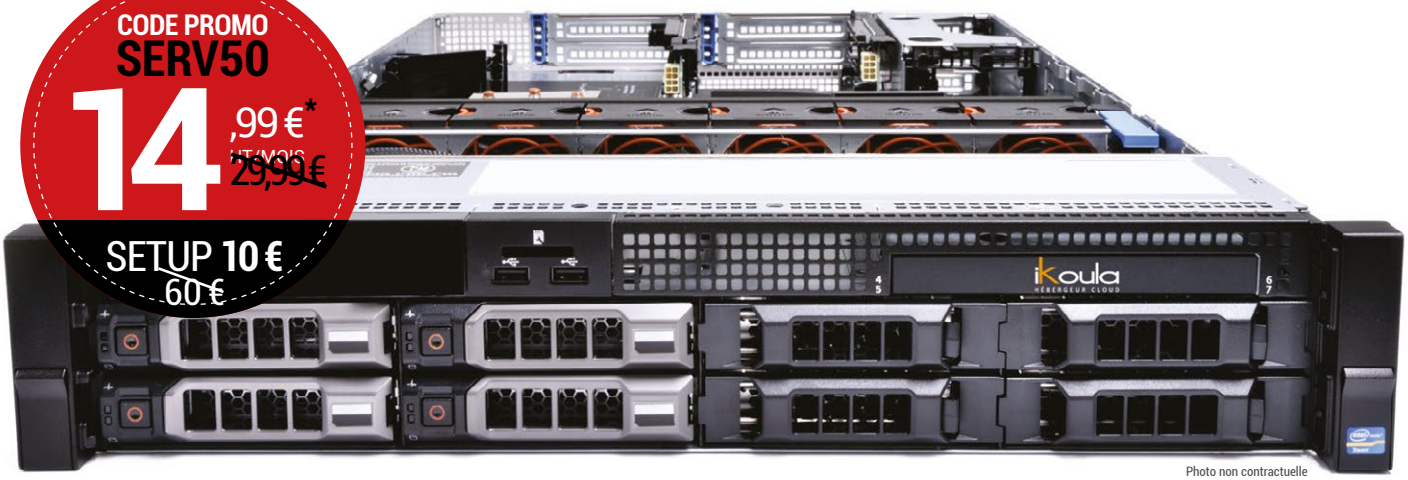


Photo non contractuelle

debian ubuntu CentOS Windows Server 2012

- Intel® Xeon® **E3 1220v5**
- 1 CPU (4C/4T) **@3 Ghz**
- **16 Go** RAM DDR4
- **1 To** SATA
- **GeForce** GT 710 1 Go
- **100 Mbs** Full duplex

COMMANDEZ SUR
https://express.ikoula.com

*Offre découverte -50 % sur la première période de souscription avec un engagement de 1 ou 3 mois et setup à 10 € HT (valable uniquement sur le plan Xeon® 1220v5 et Xeon® 1230v5, hors options et hors renouvellement). Voir toutes les conditions sur le site.

CONFIGUREZ VOTRE SERVEUR DÉDIÉ XEON®

PROCESSEUR ▼
- ● Intel® Xeon® **E3 1220v5 4C/4T @3 GHz**
- ○ Intel® Xeon® **E3 1230v5 4C/8T @3,4 GHz**

MÉMOIRE ▼
- ● **16 Go** DDR4
- ○ **32 Go** DDR4
- ○ **64 Go** DDR4

DISQUE DUR ▼
- ● **1 To** SATA
- ○ **2 To** SATA
- ○ **4 To** SATA
- ○ **240 Go** SSD
- ○ **480 Go** SSD
- ○ Disque secondaire

COULEUR ▼
- ○
- ○
- ●

ikoula
HÉBERGEUR CLOUD

sales@ikoula.com

01 84 01 02 50

GNU/Linux Magazine France
est édité par Les Éditions Diamond

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

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

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

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

Périodicité : Mensuel
Prix de vente : 7,90 €

SUIVEZ-NOUS SUR :

https://www.facebook.com/editionsdiamond

@gnulinuxmag

LES ABONNEMENTS ET LES ANCIENS NUMÉROS SONT DISPONIBLES !

EN VERSION PAPIER ET PDF :
www.ed-diamond.com

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

ÉDITO

Ça y est, 2017 est à nos portes, une nouvelle année démarre et, il y a 20 ans se produisait un événement important... enfin, pas exactement 20 ans... En septembre 1998 un OVNI apparaissait dans la presse française : *Linux Magazine* (toujours cette histoire de « GNU » oublié dans le titre). Cela remonte donc à 18 ans et 4 mois tout de même et le paysage de la presse informatique a bien changé : nous sommes passés d'une abondance de titres, de l' « âge d'or de la presse informatique » a une situation où seulement quelques titres subsistent. Parmi ces titres se trouve donc *GNU/Linux Magazine* qui est l'un des plus anciens et qui fête donc son 200ème numéro !

Comment se fait-il que nous soyons toujours là dans une période où beaucoup se satisfont des informations qu'ils glanent sur le Web ? J'aime à penser que c'est la qualité des contenus que nous proposons qui a réussi à vous fidéliser pour les uns depuis de nombreuses années et pour les autres plus récemment. Sur le Web la recherche est ciblée, une idée a germé et pris le contrôle de notre cerveau : nous voulons une réponse à une question précise. Dans un magazine comme *GNU/Linux Magazine*, on peut rencontrer une technologie, un algorithme qui nous étaient inconnus, auxquels nous ne nous serions pas intéressés forcément et qui vont miraculeusement résoudre un problème ou plus simplement nous donner de nouvelles idées. Ainsi, magazine et Web ne remplissent pas la même fonction.

L'objectif initial de *GNU/Linux Magazine* était l'acceptation de GNU/Linux et des logiciels libres. Aujourd'hui, ce n'est plus un combat de tous les jours : une grande majorité de personnes a compris que plusieurs voies étaient possibles. Ainsi, *GNU/Linux Magazine* est maintenant un magazine beaucoup plus technique et « professionnel » qui au cours de son existence a donné progressivement naissance à de multiples publications :

- *MISC* pour la sécurité informatique ;
- *Linux Pratique* pour l'utilisation des logiciels libres sous GNU/Linux ;
- *Open Silicium* pour l'électronique d'un point de vue professionnel ;
- et enfin *Hackable* pour tout ce qui touche à l'électronique de loisir pour les débutants.

L'intérêt de cette multiplication de titres est de pouvoir traiter les sujets de chaque domaine de manière plus pointue. Mais au final que reste-t-il à *GNU/Linux Magazine* ? Doit-on forcément s'interdire de parler de sécurité ou d'embarqué, car cela est fait dans une autre de nos publications ? C'était le choix qui avait été le nôtre jusqu'à présent... puis nous avons décidé de tout modifier, de revenir aux sources de ce qu'était *GNU/Linux Magazine*. En dehors d'une couverture et d'une maquette repensées que vous n'avez pas pu ne pas remarquer, nous vous proposons de nouvelles rubriques dans lesquelles vous pourrez retrouver le contenu auquel vous étiez habitué et celui qui était présent aux débuts du magazine :

- **Actualités & Humeur** : deux rubriques bien connues ont fusionné permettant la présentation des actualités du logiciel libre ou un billet d'humeur critique et tantôt sarcastique sur l'actualité technique du moment. Deux anciennes rubriques... mais une nouveauté : cette rubrique permettra également de présenter des ouvrages qui ont marqué nos auteurs et qu'ils ont trouvé utiles ;
- **IA, Robotique & Sciences** : avec cette nouvelle rubrique, nous tâcherons de comprendre les algorithmes et les outils permettant d'adapter le comportement des programmes (systèmes autonomes, reconnaissance de formes, analyse de données, etc.). Nous pourrons également nous intéresser à des applications informatiques dans des domaines scientifiques particuliers ;
- **Système & Réseau** : cette rubrique, connue, abordera tout ce qui est en rapport avec les tâches d'administration système ou réseau ;
- **IoT & embarqué** : nouvelle rubrique dédiée aux expérimentations autour de plateformes, cartes, FPGA, microcontrôleurs et objets connectés utilisant bien entendu des technologies *open source* ou libres ;
- **Kernel & bas niveau** : cette rubrique abordera tout ce qui est en rapport avec le développement kernel ou de pilotes, le support matériel, l'accès aux périphériques ou utilisant des langages de bas niveau tel que le C/C++ ;
- **Hacks & bidouilles** : la programmation est aussi un amusement ! Dans cette nouvelle rubrique, nous vous présenterons des « bricolages informatiques » : détournement de fonctionnalités ou de matériels, programmation à visée expérimentale, etc. ;
- **Libs & modules** : vous retrouvez dans cette rubrique la présentation de bibliothèques et de modules utilisés dans le cadre de développements divers ;
- **Mobile & Web** : là encore, une rubrique à laquelle vous êtes habitués et qui traitera des développements sur périphériques mobiles et pour le Web ;
- Sécurité & Vulnérabilité : comprendre la sécurité des systèmes informatiques pour pouvoir se protéger des attaques et éviter d'insérer des failles dans ses propres programmes, c'est aussi un sujet qui nous intéresse ! Voici donc la dernière nouvelle rubrique de ce *GNU/Linux Magazine « back to root »*.

Le monde change, vos attentes aussi. Cette nouvelle formule n'est pas seulement un changement esthétique, elle est la preuve que nos publications restent bien vivantes et tentent de s'adapter aux attentes de nos lecteurs, à VOS attentes.

Je vous souhaite une excellente lecture et beaucoup de découvertes dans ce 200ème numéro et j'espère vous retrouver dans un mois !

Donnez-nous votre avis sur la nouvelle formule du magazine en nous écrivant à lecteurs@gnulinuxmag.com !

Tristan Colombo

NE MANQUEZ PAS LA NOUVELLE FORMULE !

LINUX PRATIQUE N°99

NOUVELLES RUBRIQUES, ENCORE + PRATIQUE !

NOUVEAU : UN CAHIER RASPBERRY PI DANS CHAQUE NUMÉRO !

CREÉZ FACILEMENT VOTRE

MEDIA CENTER

À PARTIR DE VOTRE PC ET/OU VOTRE RASPBERRY PI

ACTUELLEMENT DISPONIBLE

CHEZ VOTRE MARCHAND DE JOURNAUX ET SUR :

http://www.ed-diamond.com

SOMMAIRE

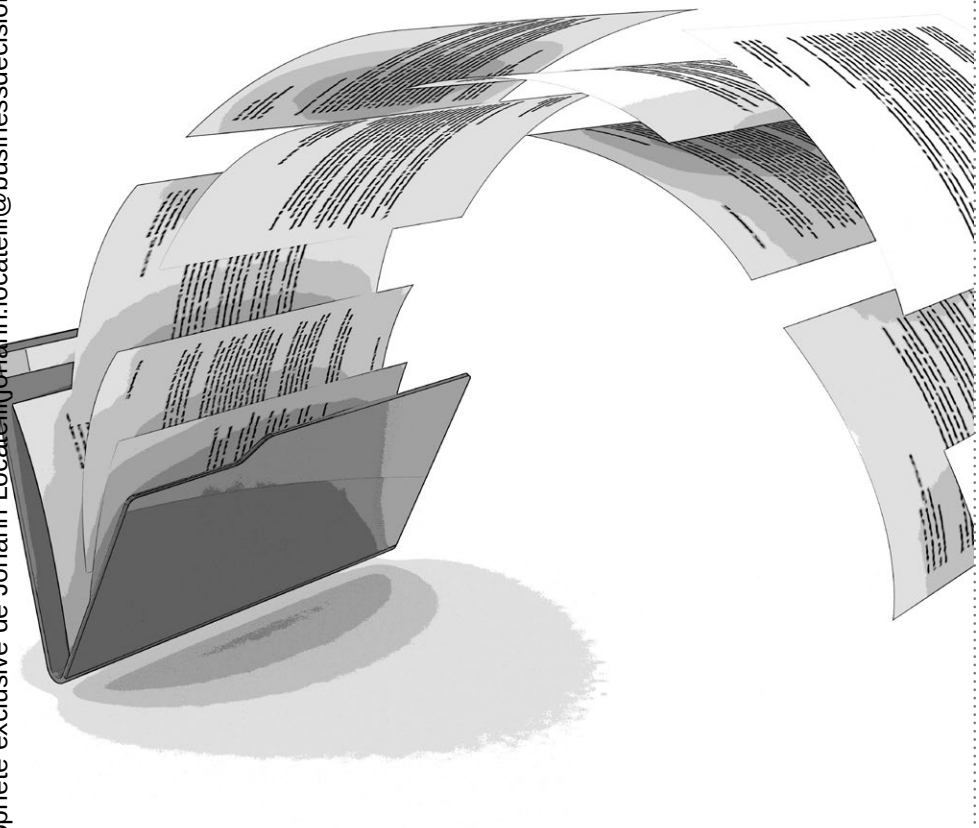
GNU/LINUX MAGAZINE FRANCE

N° 200

COMPILATION, INSTALLATION ET UTILISATION BASIQUE DE VIM 8

OLIVIER DELHOMME
[Développeur de logiciels libres pour mon loisir]

MOTS-CLÉS : VIM 8, ÉDITEUR DE FICHIERS, INSTALLATION, UTILISATION, CENTOS, DEBIAN.



Vim, l'éditeur de fichiers âgé de 25 ans cette année, peut être utilisé comme un véritable IDE [1] spécialisé. Je vous propose de découvrir cet éditeur dans sa dernière version.

J'utilise Vim de manière très basique depuis une bonne quinzaine d'années essentiellement pour éditer des fichiers de configuration ou de petits scripts shells. Depuis quelques semaines, j'ai découvert la véritable nature de Vim et son microcosme d'options de configuration et de *plugins*. Je l'utilise maintenant également pour écrire mes programmes en **Python** ou en **C** en profitant de fonctionnalités généralement intégrées dans des IDE spécialisés tels **Bluefish**, **Geany** ou **KDevelop** par exemple (pour ne pas parler d'**Emacs** ;-)). Dans cet article je vous propose de compiler sa dernière version, de l'installer et d'utiliser ses fonctionnalités de base.

1. COMPILATION ET INSTALLATION

Vim est certainement l'éditeur de fichiers textes qui se trouve en standard dans le plus grand nombre de distributions actuelles, cependant, à la date de rédaction de cet article, aucune d'entre elles ne fournit encore la version 8 qui est sortie le 12 septembre dernier.

Pour compiler Vim, il faut installer les dépendances et les outils qui nous serviront pour cela.

Sous **Debian Jessie** et dérivés :

```
# apt-get install build-essential git
libncurses5-dev python-dev libperl-dev
# apt-get install tcl-dev ruby ruby-dev lua5.2
liblua5.2-dev
```

Sous **Centos 7** et dérivés :

```
# yum groupinstall "Development Tools"
# yum install ncurses-devel lua-devel perl-
devel perl-ExtUtils-Embed.noarch
# yum install ruby ruby-devel tcl-devel tcl
python-devel
```

Pour récupérer le code source de Vim, on peut utiliser le dépôt sur **GitHub [2]** (**git** - mon choix pour cet article) ou le dépôt sur **Bitbucket [3]** (**mercurial**).

```
$ git clone https://github.com/vim/vim.git
$ cd vim
$ git checkout $(git tag | tail -n1)
$ ./configure --enable-pythoninterp=yes
--enable-perlinterp=yes --enable-luainterp=yes
--enable-rubyinterp=yes --enable-tclinterp=yes
```

À ce stade, il faut vérifier dans la sortie de la commande **configure** si le dossier de configuration de Python a été reconnu en regardant les tests liés à Python (ils sont localisés vers le début). Si les deux lignes suivantes sont présentes c'est que le script n'a pas réussi à le détecter tout seul :

```
checking Python's configuration directory...
can't find it!
```

Dans ce cas il faut lui indiquer explicitement le chemin en ajoutant une option au script configure comme suit.

Pour Debian : **--with-python-config-dir=/usr/lib/python2.7/config-x86_64-linux-gnu**.

Pour Centos : **--with-python-config-dir=/usr/lib64/python2.7/config**.

Pour déterminer le dossier du script de configuration de Python de votre système, vous pouvez utiliser la commande : **python2.7-config --configdir**.

Normalement les autres interpréteurs ont été pris en compte sans difficulté. Vim est compilé avec le maximum d'interpréteurs car il doit être capable de s'interfacer avec n'importe quel *plugin* que nous souhaiterions installer.

Par défaut cette installation se fera dans le dossier **/usr/local/**. Vous pouvez choisir un autre dossier en utilisant l'option **--prefix=mon_dossier** où **mon_dossier** est un chemin absolu vers le dossier d'installation (par exemple **/home/user/local**) où **user** serait votre *login*. N'oubliez pas alors d'ajouter **/home/user/local/bin** dans la variable **PATH** (par exemple pour bash : **export PATH="/home/user/local/bin:${PATH}"**).

Si vous deviez exécuter plusieurs fois le script **configure,** vous pouvez utiliser la commande **make distclean** pour en nettoyer le cache et ainsi permettre la réexécution de tous les scripts du **configure**.

Il reste maintenant à compiler et installer le logiciel avec les deux commandes suivantes :

```
$ make
$ sudo make install
```

Si vous avez un ordinateur comportant plus d'un seul cœur vous pouvez utiliser l'option **-j x** où **x** représente le nombre de cœurs qui seront utilisés pour compiler les sources.

Maintenant que Vim est compilé et installé, vous pouvez vérifier les options qui ont été incluses en lui demandant sa version :

```
$ vim --version

VIM - Vi IMproved 8.0 (2016 Sep 12, compiled Sep 15 2016 21:51:52)
Rustines incluses : 1-5
Compilé par dup@d630
Énorme version sans interface graphique.
  Fonctionnalités incluses (+) ou non (-) :
+acl             +file_in_path   +mouse_sgr       +tag_old_static
+arabic          +find_in_path   -mouse_sysmouse  -tag_any_white
+autocmd         +float          +mouse_urxvt     +tcl
-balloon_eval    +folding        +mouse_xterm     +termguicolors
-browse          -footer         +multi_byte      +terminfo
++builtin_terms  +fork()         +multi_lang      +termresponse
...
+eval            +mouse_dec      +statusline      -xterm_clipboard
+ex_extra        -mouse_gpm      -sun_workshop    -xterm_save
+extra_search    -mouse_jsbterm  +syntax
+farsi           +mouse_netterm  +tag_binary
         fichier vimrc système : "$VIM/vimrc"
     fichier vimrc utilisateur : "$HOME/.vimrc"
 2me fichier vimrc utilisateur : "~/.vim/vimrc"
      fichier exrc utilisateur : "$HOME/.exrc"
 fichier de valeurs par défaut : "$VIMRUNTIME/defaults.vim"
              $VIM par défaut : "/home/dup/local/share/vim"
Compilation : gcc -c -I. -Iproto -DHAVE_CONFIG_H     -g -O2 -U_
FORTIFY_SOURCE -D_FORTIFY_SOURCE=1
 Édition de liens : gcc   -L. -Wl,-z,relro -L/build/ruby2.1-bDDI00/
ruby2.1-2.1.5/debian/lib
...
```

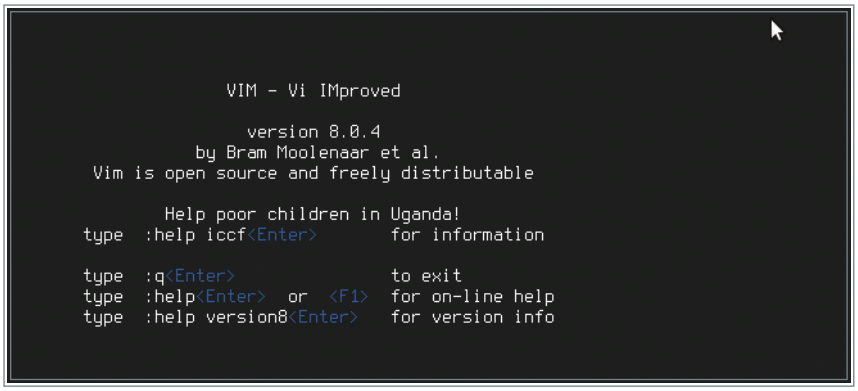


Fig. 1 : Exemple d'écran de démarrage de Vim. Sans mention d'un fichier sur la ligne de commande, Vim affiche sa version et quelques informations qui peuvent s'avérer utiles.

Maintenant que nous l'avons installé, lançons Vim, sans argument, depuis un terminal afin d'obtenir un écran similaire à celui de la figure 1.

NOTE

Pour quitter Vim tapez au clavier **:q!**. Il s'agit d'une commande (**q** - abréviation de « quit ») du mode « commande line » assortie d'un modificateur (**!**) qui indique à Vim de quitter même si le fichier modifié en cours d'édition n'est pas sauvegardé. Pour obtenir de l'aide tapez la commande **:help**. Dans le texte d'aide certains mots sont soit entourés de **|** soit d'une couleur spécifique (sur Debian et Centos il semble que par défaut cette couleur soit turquoise). Il s'agit de « liens » vers d'autres parties de la documentation. Pour suivre ces liens placez-vous sur l'un d'entre eux et tapez **<Ctrl>** + **<]>**. Pour revenir à votre position précédente vous pouvez utiliser **<Ctrl>** + **<T>** ou **<Ctrl>** + **<O>**. L'aide de Vim est très fournie et très détaillée, n'hésitez pas à la consulter.

Pour continuer la lecture de cet article, je vous conseille de copier un fichier texte et de l'ouvrir avec Vim afin d'essayer les commandes (**vim monfichier.txt**) au fur et à mesure.

Pour faire simple, dans Vim il existe 4 modes : le mode normal, le mode d'insertion, le mode visuel et le mode command-line. Les trois premiers modes sont utilisés pour éditer le texte tandis que le dernier est utilisé pour donner des ordres à Vim.

Les modes modifient le comportement des touches du clavier. Ainsi le mode d'édition permet l'écriture de tout caractère. Par exemple, on pourra écrire « :q ! » dans un fichier sans pour autant quitter vim !

NOTE

Pour sortir des modes visuels, du mode d'édition et du mode « command line » il faut appuyer sur la touche **<Esc>** ou **<Echap>** de votre clavier. En sortant de ces modes, on revient systématiquement dans le mode normal que l'on peut voir comme un mode pivot.

Le mode normal permet de naviguer dans le fichier, de couper, copier, coller du texte, de passer en mode d'édition ou en mode visuel. Le mode visuel autorise la sélection fine d'éléments du texte de manière visuelle et permet également de couper, copier et coller les sélections.

2.1 Déplacements dans le mode normal

Le mode normal permet en premier lieu la navigation dans le fichier édité : les différentes flèches du clavier sont utilisables pour cela mais également les touches **<h>**, **<j>**, **<k>** et **<l>** qui permettent respectivement de déplacer le curseur à gauche, en bas, en haut et à droite (voir **[4]** pour une explication sur le choix de ces touches). Les fichiers textes que vous éditez ne sont pas qu'une suite de caractères pour Vim. En effet, il est possible d'utiliser les touches **<w>** et **<e>** pour, respectivement, se déplacer au début du prochain mot ou à la fin de celui-ci (**<b>** permet de reculer d'un mot).

NOTE

<W> et **<B>** en majuscules déplacent le curseur jusqu'au prochain ou précédent espace. C'est utile pour les mots pouvant contenir des apostrophes par exemple.

Pour aller au début de la ligne on utilisera la touche **<^>** et pour aller à la fin de la ligne plutôt **<$>** ainsi les amateurs d'expressions rationnelles ne seront pas perdus ! Les touches/caractères **<{>** et **<}>** permettent de se déplacer au début et à la fin d'un paragraphe. Les touches **<H>**, **<M>** et **<L>** permettent respectivement de positionner le curseur en haut de la fenêtre, au milieu de la fenêtre et en bas de la fenêtre. Pour avancer de **n** lignes tapez le nombre **n** au clavier puis la touche « Entrée ». **<Ctrl>** + **<E>** et **<Ctrl>** + **<Y>** permettent respectivement de descendre et monter la fenêtre comme on le ferait dans un éditeur graphique avec la souris et la barre de défilement. Enfin, pour vous déplacer à la dernière ligne de votre fichier tapez **<G>** et pour revenir au début du fichier tapez **gg** (**<g>** deux fois – notons que l'on peut revenir à sa position d'origine en tapant **<Ctrl>** + **<O>**).

NOTE

On peut adjoindre un nombre avant d'indiquer un déplacement ou une commande. Par exemple si l'on tape **12w**, le curseur avance de 12 mots, **3}**, le curseur se place au début du troisième paragraphe suivant. Si l'on tape **7dd**, on coupe la ligne sur laquelle se trouve le curseur et les 6 lignes suivantes.

2.2 Commandes du mode normal

Tout d'abord voyons quelques commandes que l'on peut utiliser directement dans ce mode : **dd** permet de couper une ligne et la place dans le tampon par défaut (**D** en majuscule permet de supprimer depuis le curseur jusqu'à la fin de la ligne). **yy** permet de copier une ligne et la place également dans le tampon par défaut. **x** permet de supprimer un caractère de la ligne (**x** ne supprimera pas au-delà d'une ligne même en lui indiquant un plus grand nombre de caractères).

NOTE

Pour insérer le tampon par défaut sur la ligne se trouvant en dessous de celle du curseur utilisez **p** (pour « paste »). **P** en majuscule permet d'insérer la ligne au-dessus de celle où se trouve le curseur.

La force de Vim est d'avoir des commandes combinables qui s'exécuteront avec une sorte de condition. Sans entrer dans les détails on peut déjà faire beaucoup avec quatre commandes (que l'on a pratiquement déjà vues) : **d** pour détruire le texte (l'effacer, le couper), **c** pour changer le texte (cette commande change le mode et passe en mode d'édition) et **y** pour « yank » qui permet de copier du texte et enfin **v** pour sélectionner en mode visuel (cette dernière commande change également le mode et passe en mode visuel). À l'exception de la commande **v** qui passera simplement en mode visuel, on ne peut utiliser ces commandes sans leur adjoindre des caractères qui définiront une « étendue ».

Les « étendues » sont au minimum la combinaison de deux caractères. Le premier définit la condition ou la façon d'appliquer la commande et le deuxième caractère indique sur quoi s'applique la commande. Pour le premier caractère il existe trois conditions/façons : **a** pour « all » (tout), **i** pour « in » (à l'intérieur), **t** pour « 'til » ou « until » (jusqu'à). Le deuxième caractère est soit l'un des caractères de déplacement vu précédemment soit un caractère de la ligne de texte sur laquelle est le curseur. Toutes les combinaisons ne sont pas forcément admises mais il est possible d'écrire par exemple : **diw** qui supprime le mot entier (si on avait écrit **dw** seule la fin du mot aurait été supprimée), **ca)** coupe tout ce qui est entre les parenthèses, y compris les parenthèses et passe en mode d'édition, **yi]** copie tout ce qui se trouve entre les crochets, sans les crochets, **vt'** (sélectionne tout jusqu'au prochain caractère **'** sur la ligne et passe en mode visuel.

Pour passer en mode d'édition, un certain nombre de commandes peuvent être utilisées. Les commandes les plus usitées sont **i** et **I** (**i** en majuscule) qui permettent d'insérer du texte respectivement avant la position du curseur et au début de la ligne et les commandes **a** et **A** qui permettent respectivement d'ajouter du texte après la position du curseur ou à la fin de la ligne. Une fois en mode d'édition, il est possible de taper n'importe quel caractère (n'oubliez pas la touche **<Esc>** qui permet de retourner en mode normal).

NOTE

Pour répéter la dernière commande, il est parfois possible d'utiliser la commande **.** (point).

v, **V**, **<Ctrl>** + **<V>** sont les commandes qui permettent de passer en mode visuel. **v** en minuscule permet de sélectionner

le texte au caractère prêt. `V` en majuscule permet de sélectionner visuellement des lignes entières et enfin **<Ctrl>** + **<V>** permet de sélectionner un bloc de texte.

2.3 Le mode visuel

Dans ce mode, une fois la sélection effectuée, il est possible d'utiliser directement les commandes `d`, `D`, `c` et `y` vues précédemment. Le mode de sélection par bloc (**<Ctrl>** + **<V>**) permet, avec l'utilisation de la commande `I` (`i` en majuscule) de modifier n'importe laquelle des colonnes sélectionnées. Par exemple, sur un script bash, commenter les 3 paragraphes suivants le curseur en tapant **<Ctrl>** + **<V>** `3}I #` et **<Esc>**. Le travail par colonnes avec **<Ctrl>** + **<V>** et le mode de sélection par lignes avec `V` (pour copier des fonctions ou des portions de code) sont pour le moment les deux seules utilisations que j'ai du mode visuel.

2.4 Le mode « command line »

Pour utiliser des commandes dans ce mode il faut en premier lieu taper le caractère `:` qui permet l'entrée dans le mode. Dans ce mode il est possible de sauvegarder (`:w` – enregistre en écrasant le fichier), de quitter (`:q` – quitte sauf s'il persiste des modifications ; dans ce cas, pour quitter en perdant les modifications on peut ajouter le point d'exclamation ainsi `:q!`), d'ajouter un fichier à la ligne suivante (`:r nomdufichier`), d'éditer un autre fichier (`:e nomdufichier`). Comme dans le mode normal, il est possible parfois de combiner les commandes et l'on pourra écrire `:wq` pour enregistrer et quitter Vim. Toutefois les puristes utiliseront `:x` qui permet d'enregistrer les modifications, seulement s'il y en a, de chacun des fichiers éventuellement ouverts et qui a le bon goût d'être plus court à taper !

NOTE

Après avoir tapé le `:` vous pouvez utiliser les flèches de votre clavier pour naviguer dans l'historique des commandes déjà utilisées. Vous pouvez également taper `:hist` (un raccourci pour `:history`) pour obtenir l'historique des commandes tapées dans le mode *command line*.

2.5 Défaire et refaire

Vim possède quelques commandes intéressantes permettant de défaire ou refaire ce que vous venez de modifier. Tout d'abord tant que l'on reste en mode d'édition on effectue un et un seul changement (même si l'on édite totalement l'ensemble du texte). En effet l'ensemble d'un changement est clôt dès que l'on quitte le mode d'édition. Chaque commande dans le mode normal introduit un changement. Chacun des changements est réversible, en mode normal grâce à la commande `u` (pour *undo*), en mode *command-line* grâce à la commande `:earlier`. La commande **<Ctrl>** + **<r>** en mode normal et `:later` en mode *command-line* permettent de refaire ce qui vient d'être défait. L'intérêt des commandes `:earlier` et `:later` est qu'il est possible de leur indiquer soit un nombre de modifications à faire ou défaire, soit un laps de temps. Par exemple, pour retrouver l'état de votre fichier 1 heure auparavant il est possible d'écrire `:earlier 1h`.

CONCLUSION

Vous avez maintenant un Vim tout neuf sur votre système. Vim est activement développé aussi, entre l'écriture de ces quelques lignes et leur date de parution, un grand nombre de correctifs et ajouts auront été écrits.

Cet article, un peu rébarbatif, est un bon début pour commencer à pratiquer. À mon avis il est illusoire de penser que l'on peut apprendre (et retenir) un si grand nombre de modes et de commandes en quelques jours. Seuls la pratique régulière et l'apprentissage au fil de l'eau sur plusieurs semaines permettent d'ancrer dans vos doigts les commandes qui vous sont utiles. Pour vous aider, vous pouvez utiliser une fiche (*Cheat Sheet*) **[5]** qui vous servira de référence.

Parfois, avoir une personne à qui parler pour commencer à utiliser Vim peu aussi aider. Pour cela vous pouvez vous rapprocher de votre **GUL** (Groupe d'Utilisateurs de Linux) local **[6]**. Il organise probablement des sessions tuppervim **[7]** comme par exemple l'**ALDIL [8]** à Lyon.

Nous pourrons éventuellement voir dans de prochains articles le système de tampons, le système de macro particulièrement simple à utiliser, comment lui adjoindre des *plugins* et probablement bien d'autres choses encore ! ■

RÉFÉRENCES

[1] Définition d'un environnement de développement intégré : https://fr.wikipedia.org/wiki/Environnement_de_développement

[2] Le dépôt git de Vim : https://github.com/vim/vim.git

[3] Le dépôt mercurial de Vim : https://bitbucket.org/vim-mirror/vim

[4] Pourquoi **<h>**, **<j>**, **<k>**, **<l>** pour se déplacer : http://xahlee.info/kbd/keyboard_hardware_and_key_choices.html

[5] Une fiche de référence : https://www.fprintf.net/vimCheatSheet.html

[6] Liste des GUL : https://aful.org/gul/liste

[7] Événement tuppervim à Paris http://tuppervim.org/

[8] LE GUL Lyonnais : http://aldil.org/

ACTUELLEMENT DISPONIBLE

HACKABLE N°16 !

FAITES COMMUNIQUER VOS PROJETS SANS FIL !

NE LE MANQUEZ PAS

CHEZ VOTRE MARCHAND DE JOURNAUX ET SUR :

http://www.ed-diamond.com

DISSÉMINATION DE DONNÉES GÉORÉFÉRENCÉES – QGIS-SERVER ET OPENLAYERS

JEAN-MICHEL FRIEDT
[Institut FEMTO-ST, dpt. Temps-fréquence, Besançon]

ÉMILE CARRY
[Institut FEMTO-ST, dpt. Temps-fréquence, Besançon]

MOTS-CLÉS : GIS, QGIS-SERVER, QGIS, OPENLAYERS, DONNÉES GÉRÉFÉRENCÉES, WMTS, WFS

Les données géoréférencées, générées par l'auteur ou distribuées par les agences telles que l'ESA (sentinels.copernicus.eu) ou l'USGS (earthexplorer.usgs.gov), sont disséminées au travers d'un service accessible par le Web selon les formats WFS ou WMTS. Ces données sont alors exploitables au travers de logiciels spécialisés tels que QGis, ou de l'API OpenLayers pour être intégrées dans des pages Web.

Tout possesseur de téléphone portable devient en mesure aujourd'hui de produire de la donnée géoréférencée, qu'il s'agisse de photographies numériques ou de traces de parcours pour les données brutes, ou des données issues d'informations géoréférencées acquises. Nous avons par exemple présenté au **FOSS4G-fr [1]** et dans ces pages **[2]** la génération de modèles de bâtiments par les photographies géoréférencées prises par téléphone, et leur intégration dans **QGis** sur fond de carte **OpenStreetMaps** (OSM).

Obtenir de telles données est bien, mais les partager avec une communauté d'utilisateurs, c'est mieux. Le mode de transmission le plus pratique pour les volumes de données dont il est question est évidemment internet, ne serait-ce que pour centraliser les données partagées en vue de garantir leur cohérence si plusieurs utilisateurs les manipulent.

Trois protocoles de dissémination ont été mis en place à cet effet **[3][4]** : WM(T)S, WFS et WCS. Le premier dissémine des images représentant les données traitées, avec une résolution adaptée à l'échelle de la carte de l'utilisateur. Ne fournissant qu'une image des informations transmises, il n'est plus possible pour l'utilisateur de s'approprier ou retraiter lui-même des informations. Les deux autres protocoles disséminent les données brutes, vectorielles ou matricielles. Nous nous proposons d'aborder séquentiellement trois problèmes : un serveur de données pour disséminer les informations qui auront été traitées dans QGis (**`qgis-server`**) ; la récupération des informations sur un logiciel dédié tel que QGis pour le traitement des informations ; et finalement la récupération des données dans un client Web pour affichage comme couche OpenLayer**s**. Afin de ne pas faire durer le suspens plus longtemps, le lecteur est encouragé à consulter qgis.sequanux.org/jmfwms.html et jmfriedt.sequanux.org/reproj.html pour se faire une idée du résultat recherché.

Le contexte de notre étude porte sur le retrait de glaciers à front marin en milieu arctique. Les images des satellites **Landsat** sont acquises depuis les années 1970 et mises à disposition par l'**USGS**, par exemple sur http://landsatlook.usgs.gov/viewer.html. Elles sont plus ou moins bien géoréférencées (une petite correction est parfois nécessaire, surtout pour les plus anciennes), mais surtout souffrent d'une résolution médiocre (de l'ordre de 30 m/pixel) compte tenu des standards actuels. Plus de 40 ans d'histoire d'images satellites en font néanmoins une source irremplaçable pour appréhender l'évolution des régions qui vont nous intéresser. Notre travail initial sous QGis a été d'importer toutes ces images, les reprojeter dans un référentiel localement plan (WGS84/UTM33N, le référentiel projeté approprié pour le nord de la Norvège tel que nous en informe http://spatialreference.org/ref/epsg/wgs-84-utm-zone-33n/), s'assurer par quelques points de référence que les images sont convenablement positionnées, et dans le cas contraire les repositionner au moyen de 4 ou 5 points de référence sur le pourtour de la région considérée (presqu'île de Brøgger, Svalbard). Ces informations matricielles sont alors utilisées pour tracer manuellement (création d'un *shapefile* comprenant des lignes) les fronts marins des glaciers, créant ainsi un jeu de données vectorielles. Par ailleurs, un trajet en avion a été enregistré au moyen du récepteur GPS d'un téléphone portable (logiciel **OSMTracker** sous **Android**) et nous désirons insérer la séquence de points au format **GPX** dans nos cartes. Sous QGis, l'opération est triviale (***Vector > GPS Tools > Load GPX File***) mais pourrons-nous exporter cette information par le Web et l'inclure dans les pages affichées par un navigateur ? Nous verrons que la réponse n'est pas aussi triviale qu'il pourrait paraître.

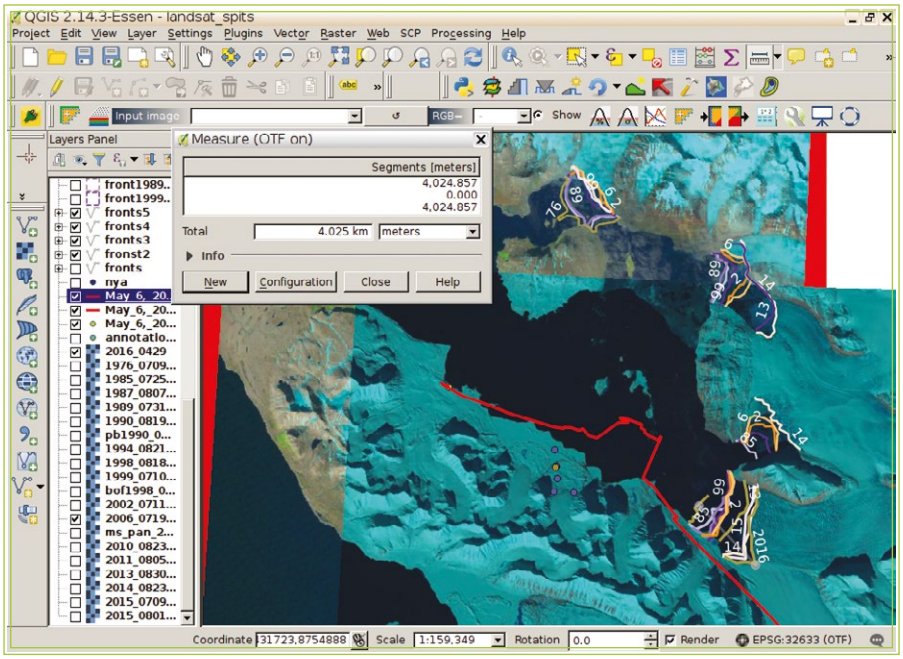


Fig. 1 : Projet QGis que nous désirons partager au travers d'internet, avec toutes les images sans nuages acquises par Landsat de la presqu'île de Brøgger au Svalbard, et la mesure de position des fronts, illustrant le recul de 4 km en 40 ans du Kongsbreen.

1. LE SERVEUR : QGIS-SERVER

QGis a tout prévu pour nous : un projet QGis s'exporte dans un format accessible par le Web au travers de qgis-server. Le principal exercice va donc porter sur l'installation de cet outil, puisqu'exporter le projet se résume à ***Project > Projet Properties*** et activer WMS, WFS et WCS en cochant toutes les couches (***Select All***) dans les deux derniers cas.

L'installation de qgis-server ne présente aucune difficulté et est décrite parfaitement sur http://docs.qgis.org/testing/en/docs/user_manual/working_with_ogc/ogc_server_support.html. On notera cependant que bien que le service WM(T)S

fonctionne convenablement avec des versions plus anciennes de qgis-server, l'export des couches vectorielles (formats WFS et WCS) ne semble pas opérationnel pour les versions précédent 2.12. Nous expérimenterons donc avec une version au moins supérieure à 2.14 de QGis et de son serveur qgis-server. Sous **Debian** GNU/Linux, cela signifie éviter **Wheezy** mais travailler dans la version stable à la date de cette rédaction.

À l'issue de l'installation du serveur comme service d'Apache2 (nécessitant le support FCGI fourni par **libapache2-mod-fcgid**), le bon fonctionnement du service est validé (dans le cas d'une installation locale) par :

```
$ wget -O - "http://127.0.0.1/cgi-bin/
qgis_mapserv.fcgi?SERVICE=WFS&VERSION=1.0
.0&REQUEST=GetCapabilities"
```

La réponse doit être du type :

```
 <Service>
  <Name>WMS</Name>
  <!-- Human-readable title for pick lists -->
  <Title>QGIS mapserver</Title>
  <!-- Narrative description providing
additional information -->
  <Abstract>A WMS service with QGIS mapserver
</Abstract>
...
```

Le service étant fonctionnel, divers projets QGis (fichier **.qgs**) sont placés, avec toutes les données associées aux diverses couches du projet, dans des sous-répertoires de **/usr/lib/cgi-bin**, en complétant avec une copie du serveur FastCGI (**qgis_mapserv.fcgi**) .

ATTENTION !

Les liens symboliques entre **/usr/lib/cgi-bin** et, par exemple, un répertoire utilisateur, ne sont autorisés qu'en modifiant la configuration par défaut de Apache2 sous Debian dans **/etc/apache2/conf-available/serve-cgi-bin.conf** en remplaçant le **+** par un **-** de l'option **SymLinksIfOwnerMatch**.

Si le projet QGis autorise l'export des couches au format WM(T)S ou WFS, nous validons la capacité à rapatrier ces couches (dans notre cas le projet et ses fichiers se trouvent dans **/usr/lib/cgi-bin/project/landsat** de qgis.sequanux.org) par :

```
$ wget -O - "http://qgis.sequanux.org/
cgi-bin/project/landsat/qgis_mapserv.fcgi
?SERVICE=WFS&VERSION=1.0.0&\
REQUEST=GetCapabilities"
```

La réponse est la liste des couches accessibles sous la forme :

```
<WFS_Capabilities xmlns:xsi="http://www.
w3.org/2001/XMLSchema-instance" ...
 <Service>
  <Name>WFS</Name>
...
  <FeatureType>
   <Name>fronts5</Name>
   <Title>fronts5</Title>
   <Abstract></Abstract>
   <SRS>EPSG:4326</SRS>
   <Operations>
    <Query/>
   </Operations>
   <LatLongBoundingBox maxx="12.2424"
minx="12.103" maxy="79.0268"
miny="78.9989"/>
  </FeatureType>
...
```

Nous voyons donc que la projection dans laquelle les coordonnées des points formant la couche (ici vectorielle) est renseignée (**EPSG:4326** signifie WGS84 en coordonnées sphériques tel que nous l'indique le CRS sous QGis), et l'extension de la couche (autour de 12°E et 79°N).

Ces points sont récupérés – au format GML – par une requête WFS de la forme :

```
$ wget -O - "http://qgis.sequanux.org/
cgi-bin/project/landsat/qgis_mapserv.fcgi
?SERVICE=WFS&VERSION=1.0.0&\
REQUEST=GetFeature&typename=fronts5&featu
reid=fronts5.toto"
```

La réponse est :

```
   <gml:LineString srsName="EPSG:4326">
    <gml:coordinates cs=","
ts=" ">12.11525618,79.02193675
12.1224397,79.02118041 12.11728195,
79.01669523 12.11808149,79.0123295
12.12970628,79.00850923
12.14676275,79.00549989
12.16023071,79.00480137
12.16426326,79.00331772
12.16469826,79.00161054
12.17939472,78.99997196
12.18340417,79.00381158 12.20189733,
79.00556583 12.21455655,79.00542793
12.2268052,79.00560265
12.2366082,79.00626179
12.24175315,79.00561112
</gml:coordinates>
```

Nous avons ici les diverses coordonnées qui forment la ligne d'une des couches vectorielles de notre projet QGis,

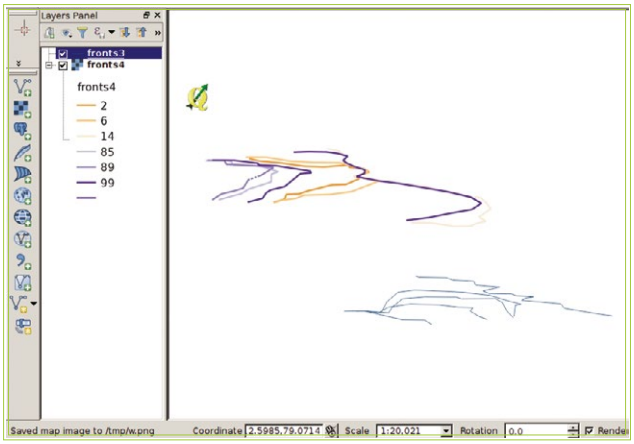


Fig. 2 : Obtention de couches WM(T)S – images au format bitmap des données ("fronts4") – et WFS – informations vectorielles ("fronts3"), depuis QGis. Noter le logo de QGis qui est un effet de bord du serveur de test installé par le site web décrivant l'installation de QGis. Effacer /opt/qgis-server/plugins/HelloServer ou tout au moins /opt/qgis-server/plugins/HelloServer/icons/icon.png pour s'affranchir de cet effet.

accessible au travers d'une requête HTML. Ces données sont donc accessibles par la communauté, il reste à en faire un bon usage avec les divers outils à notre disposition. Nous présentons quelques exemples ci-dessous.

2. LE CLIENT : QGIS

Le premier environnement le plus simple pour accéder à ces données est QGis lui même. Dans la barre des icônes de gauche de QGis, les trois icônes en forme de globe proposent l'accès respectivement aux données au format WM(T)S, WCS et WFS. Dans tous les cas, il nous faut renseigner l'url du serveur, par exemple http://qgis.sequanux.org/cgi-bin/project/landsat/qgis_mapserv.fcgi. Lors de la connexion, la liste des couches accessibles est proposée, et cliquer sur une couche suivi de ***Add*** donne accès au jeu de données. La seule subtilité dans cette application tient à ne pas utiliser un serveur trop ancien (avant 2.12) sous peine de voir la liste des couches s'afficher, mais une liste vide de points être retournée par les requêtes WFS. WM(T)S semble avoir toujours bien fonctionné (voir figure 2).

3. INTÉGRATION DANS UNE PAGE WEB : OPENLAYERS 2

La portée de notre serveur se retreint encore à un public relativement averti puisque utilisateur de QGis. Nombreuses sont les cartes sur le Web qui ne sont pas destinées à un auditoire de géographes mais simplement pour illustrer un concept à des utilisateurs probablement spécialistes dans d'autres domaines. Afin d'insérer des cartes et les couches générées sous QGis, l'environnement de développement en **JavaScript** OpenLayers semble idéal puisque fournissant la capacité à charger des couches dans tous les formats qui nous intéressent – WM(T)S, WFS ou GPX pour les traces GPS. Nous avons initialement été sensibilisés à OpenLayers comme environnement de développement pour accéder aux couches fournies par l'IGN au travers de Géoportail. Nous allons dans un premier temps nous intéresser à la version stable d'OpenLayers (version 2) qui semble mature, mais verrons que la nouvelle mouture (version 3) offre des fonctionnalités précieuses qui justifient la migration vers l'environnement en cours de développement.

Une carte OpenLayers est formée d'un appel aux bibliothèques JavaScript (**<script>...</script>**), la création d'une carte remplie d'un fond par défaut – nous choisissons le fond vectoriel OpenStreeMaps (OSM) – et recouvert de diverses couches additionnelles que nous fournirons depuis qgis-server.

Dans son expression la plus simple, une page OpenLayers ressemble à :

```
<html>
  <head>
    <title>Essai de lecture de WMS depuis openlayers</title>
    <script src="http://openlayers.org/api/OpenLayers.js"></script>
  </head>
  <body>
    <div style="width:100%;height:100%" id="map"></div>
    <script defer="defer" type="text/javascript">
      var map=new OpenLayers.Map('map');
      var wms=new OpenLayers.Layer.WMS("Basic map","http://vmap0.
tiles.osgeo.org/wms/vmap0",{layers: 'basic'} );
     map.addLayer(wms);
      map.setCenter(new OpenLayers.LonLat(12,79),9);
    </script>
  </body>
</html>
```

Et tant que nous n'insérons que des couches WMS – donc des images (puisque les données, même vectorielles, sont transmises sous forme d'images avec une résolution adaptée au facteur de grossissement), tout se passe bien. Nous pouvons ainsi ajouter une image Landsat accessible sous forme WMS, voir même un des fronts de glacier :

```
<html>
  <head>
    <title>Essai de lecture de WMS depuis
openlayers</title>
    <script src="http://openlayers.org/api/
OpenLayers.js"></script>
  </head>
  <body>
    <div style="width:100%;height:100%"
id="map"></div>
    <script defer="defer" type="text/javascript">
      var map=new OpenLayers.Map('map');
      var wms=new OpenLayers.Layer.WMS("Basic
map","http://vmap0.tiles.osgeo.org/wms/
vmap0",{layers: 'basic'} );
      var dm_wms2013=new OpenLayers.Layer.WMS(
        "Landsat image 2013",
        "http://qgis.sequanux.org/cgi-bin/
project/landsat/qgis_mapserv.fcgi",
        {layers: "2013_0830utm33n",
transparent:"true", format:"image/png" },
        {isBaseLayer: false}
      );
      var fronts1=new OpenLayers.Layer.WMS(
        "Glacier fronts1",
        "http://qgis.sequanux.org/cgi-bin/
project/landsat/qgis_mapserv.fcgi",
        {layers: "fronts4", transparent:"true",
format:"image/png"},
        {isBaseLayer: false}
      );
      map.addLayer(wms);
      map.setCenter(new OpenLayers.
LonLat(12,79),9);
      map.addLayer(dm_wms2013);
      map.addLayer(fronts1);
    </script>
  </body>
</html>
```

Le lecteur est encouragé à compléter cet exemple en ajoutant un front additionnel, par exemple la couche nommée **fronts5**, et observer la conséquence d'échanger l'ordre d'affichage des couches. Ces exemples sont fortement inspirés des excellents tutoriaux fournis par http://demo.3liz.com/wfst/wfs.html.

Pour WFS, le problème se corse : le principe de *Same Origin* impose que le serveur de données soit sur le même site que le serveur Web. Ceci est valable pour toute donnée vectorielle. Afin d'illustrer ce problème, nous proposons trois sources de données : un même fichier GPX stocké sur jmfriedt.sequanux.org, jmfriedt.free.fr, puis qgis.sequanux.org. Le serveur qgis-serveur fournissant les couches WFS est exécuté comme service Apache2 sur qgis.sequanux.org, tandis que la page Web qui fait appel à ce service peut être placée sur chacun de ces trois serveurs. La figure 3 illustre les interdictions d'accès rencontrées, tel qu'indiqué par **Firebug** : le fichier GPX stocké sur jmfriedt.sequanux.org peut être lu par la page http://jmfriedt.sequanux.org/jmfwfs.html mais est rejeté par la même page stockée sur http://jmfriedt.free.fr/jmfwfs.html. Cette dernière n'a accès qu'au même fichier GPX placé dans le répertoire du serveur jmfriedt.free.fr.

Les exemples de la figure 3 sont obtenus, après la définition de la carte, par l'appel aux couches vectorielles aux formats GPX ou WFS, de la forme :

```
var lgpxfree = new OpenLayers.Layer.Vector("GPX
track free", {
  strategies: [new OpenLayers.Strategy.Fixed()],
  protocol: new OpenLayers.Protocol.HTTP({
    url: "http://jmfriedt.free.fr/
May_6,_2016_11;20;10_2016-05-06_11-20-10.gpx",
    format: new OpenLayers.Format.GPX()
  }),
  style: {strokeColor: "red", strokeWidth: 5,
strokeOpacity: 0.8}
});
```

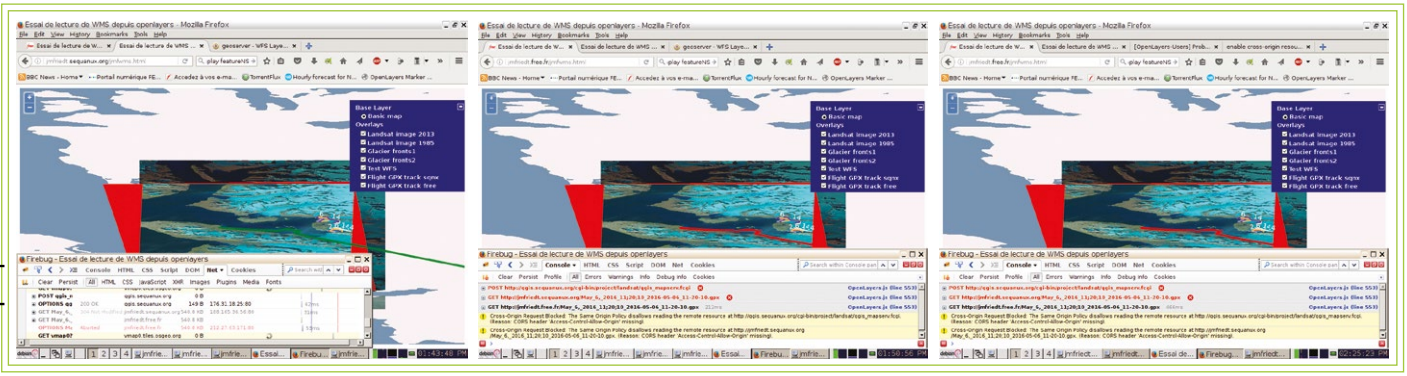


Fig. 3 : Trois illustrations du problème de Same Origin pour se convaincre de la source du problème.

```
var lgpxsqnx = new OpenLayers.Layer.Vector("GPX
track sqnx", {
  strategies: [new OpenLayers.Strategy.Fixed()],
  protocol: new OpenLayers.Protocol.HTTP({
    url: "http://jmfriedt.sequanux.org/
May_6,_2016_11;20;10_2016-05-06_11-20-10.gpx",
    format: new OpenLayers.Format.GPX()
  }),
  style: {strokeColor: "green", strokeWidth: 5,
strokeOpacity: 0.8}
});

var fronts=new OpenLayers.Layer.Vector("Test WFS",
{      // WFS
  strategies: [new OpenLayers.Strategy.BBOX()],
  protocol: new OpenLayers.Protocol.WFS({
    url : "http://qgis.sequanux.org/cgi-bin/
project/landsat/qgis_mapserv.fcgi",
    featureType : "fronts5",
  }),
});

  map.addLayer(wms); map.setCenter(new OpenLayers.
LonLat(12,79),9);
  map.addLayer(fronts);
  map.addLayer(lgpxsqnx);
  map.addLayer(lgpxfree);
```

Ce code fait appel au même fichier GPX stocké sur deux serveurs (afin de valider que seule la couche située sur le même serveur que la page Web appelante est acceptable), et la couche nommée **fronts5** fournie au format WFS par qgis.sequanux.org/cgi-bin/project/landsat/qgis_mapserv.fcgi.

Ayant dépassé tous les écueils de configuration, nous avons finalement la satisfaction de voir toutes les couches s'afficher sur la même carte, en ajoutant l'onglet de commandes qui permet d'activer ou désactiver chaque couche afin d'aisément comparer la position des fronts des glaciers et leur évolution dans le temps :

```
var map=new OpenLayers.Map('map');
ls=new OpenLayers.Control.LayerSwitcher({'div':OpenLayers.
Util.getElement('layerswitcher')});
map.addControl(ls);
ls.maximizeControl();
```

Il n'aura cependant pas échappé au lecteur que nos images satellites semblent fortement déformées (comparer les figures 1 et 4). Il s'agit ici de la conséquence de la projection de Mercator, qui tend à considérablement grossir l'axe des abscisses lorsque la zone considérée s'éloigne de l'équateur. Quoiqu'acceptable jusqu'à des latitudes de ±60°, l'effet devient vraiment significatif au-delà et carrément désagréable par 79°N. C'est pourquoi les projections locales, tentant de trouver un plan localement tangent à la sphère représentant le globe terrestre, ne prétendent pas à s'appliquer à l'ensemble de la Terre mais uniquement à un petit segment de longitude. Malheureusement, OpenLayers 2 ne supporte pas la reprojection à la volée des données bitmap, nous sommes coincés avec la projection imposée depuis **Google Maps** qui continue à déformer les pôles. Nous allons remédier à ce problème en passant sous OpenLayers 3 et retrouver ainsi les belles cartes de QGis.

4. À LA POINTE DE LA TECHNOLOGIE : OPENLAYERS 3

Dans la course aux fonctionnalités, nous ne cessons de casser ce qui marche pour mettre à jour les nouvelles fonctionnalités. Pourquoi donc casser ce bel exemple OpenLayers2 pour le remplacer par OpenLayers3 ? La déformation excessive des cartes aux latitudes élevées est un défaut bien connu

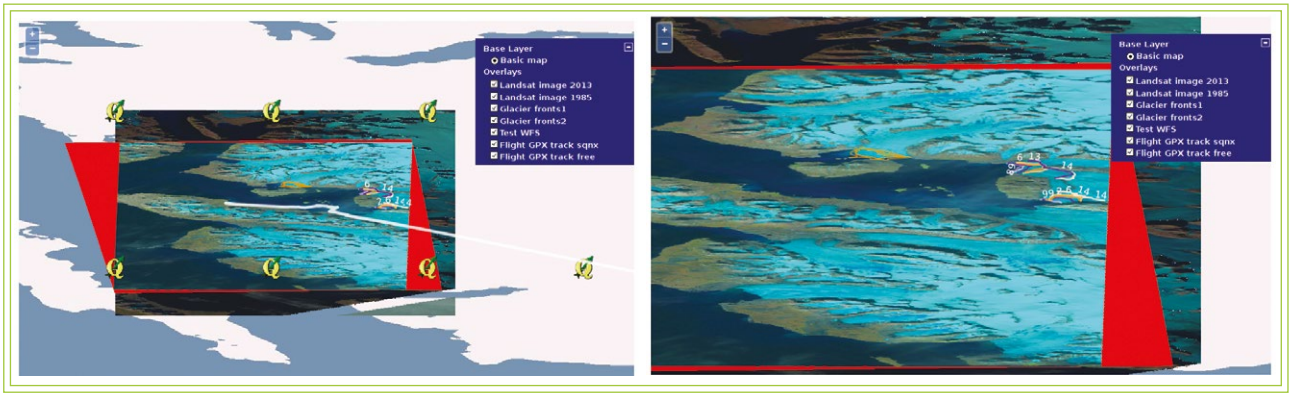


Fig. 4 : En local (127.0.0.1), tout se passe bien, les données et le serveur sont au même endroit. L'intérêt en terme de diffusion reste limité : placer toutes ses données sur le même site que le serveur résout les problèmes de Same Origin (l'exemple de droite est accessible sur qgis.sequanux.org/jmfwms.html).

Fig. 5 : Gauche : le passage à une projection locale WGS84/UTM33N résout la déformation observée auparavant dans OpenLayers 2 et son Google Mercator. Cependant, cette projection n'a pas prétention de s'étendre à tout le globe, et réduire le grossissement met en évidence la déformation sur les régions adjacentes, ici le nord Canada et la Sibérie (droite). Cet exemple est disponible sur jmfriedt.sequanux.org/reproj.html.

de la projection de Mercator utilisée par défaut par tous les environnements cartographiques de la toile (projection de code 900913 – l'écriture *leet* de Google – puisque la norme choisie n'a pas été validée par l'instance qu'est l'EPSG). Étant conscients des déficiences de la projection de Mercator, nous voudrions pouvoir sélectionner un mode de projection localement tangent à la région considérée – à savoir Universal Transverse Mercator (UTM). Et nous en arrivons à l'excuse pour passer de OpenLayers 2 à 3 : la version actuelle d'OpenLayers ne permet pas la projection au vol des images, et impose donc une sortie déformée, particulièrement gênante lorsque nous nous éloignons de l'équateur.

Nous reprenons donc toutes nos investigations, mais cette fois dans le contexte d'OpenLayers 3, avec la nécessité pour chaque nouvelle couche de préciser le mode de projection d'entrée (toutes nos données sont stockées en coordonnées sphériques, donc WGS84 de code EPSG:4326). OpenLayers ne connaît pas tous les modes de projection de sortie : nous devons les définir selon nos besoins (voir figure 5).

Heureusement, **proj4** se charge de cette opération pour nous, et http://spatialreference.org/ref/epsg/wgs-84-utm-zone-33n/ nous informe de la façon de définir WGS84/UTM33N (la bande qui couvre la Norvège – nous aurions choisi UTM31N pour la France). Ainsi, le mode de projection qui va nous intéresser se définit par :

```
proj4.defs('EPSG:32633','+proj=utm +zone=33 +ellps=WGS84
+datum=WGS84 +units=m +towgs84=0,0,0 +no_defs');
```

Et nous pouvons faire appel aux outils de conversion de la forme :

```
var osmLayer = new ol.layer.Tile({ source: new ol.source.OSM() });

var map = new ol.Map({
  controls: ol.control.defaults().extend([new ol.control.
ScaleLine()]),
  target: 'map',
  view: new ol.View({projection: 'EPSG:32633',center: ol.proj.
fromLonLat([10, 79], 'EPSG:32633'),zoom: 9})
});

var dm_wms2013 = new ol.layer.Tile({
  preload: Infinity,
  visible: true,
  source: new ol.source.TileWMS(({
    url: 'http://qgis.sequanux.org/cgi-bin/project/landsat/qgis_
mapserv.fcgi',
    params: {'LAYERS': '2013_0830utm33n', 'TILED': true,
'VERSION': '1.3.0',
    'FORMAT': 'image/png8', 'WIDTH': 256, 'HEIGHT': 256, 'CRS':
'EPSG:4326'},
    serverType: 'geoserver'
  }))
});

...
map.addLayer(osmLayer);
map.addLayer(dm_wms2013);
```

Ici la projection de la carte respecte la norme EPSG (WGS84/UTM33N s'appelle **EPSG:32633**) que nous avons défini par proj4 auparavant, tandis que les données en entrée (ici issues de la fonction WM(T)S du serveur) sont au format WGS84 en coordonnées sphériques (aussi nommée **EPSG:4326**).

5. AJOUT AUTOMATIQUE DE TOUTES LES COUCHES WMS D'UN PROJET QGIS-SERVER DANS OPENLAYERS2

Une dernière question que nous pouvons nous poser, dans le contexte d'un travail collaboratif où chaque utilisateur est susceptible d'ajouter ses propres couches, tient en la capacité à automatiser la génération d'une page Web contenant toutes les couches fournies par un serveur QGis. Cela signifie donc récupérer la description XML des capacités du serveur, découper cette description pour extraire le nom des couches disponibles, et finalement générer automatique le script JavaScript affichant toutes ces couches. De telles fonctionnalités sont fournies (paquet **php-xml** dans la distribution Debian) par **simplexml**. Ainsi, un premier script qui se contente de lister toutes les couches disponibles en recherchant d'abord les fonctionnalités WMS du serveur, puis en étudiant **[5]** la liste des propriétés (**FeatureTypeList**) et finalement les propriétés de chacun de ces objets (**FeatureType**) pour en extraire le nom (**FeatureTypeList->FeatureType->Name**) ressemble à :

```
<html><head><title>XML depuis PHP
</title></head><body>
<?php
  if (extension_loaded('simplexml')) {
    echo "extension installed:<hr>";
    $mypix = simplexml_load_file(urlencode(
      'http://127.0.0.1/cgi-bin/project/
landsat/qgis_mapserv.fcgi? SERVICE=WFS&VERSION
=1.0.0&REQUEST=GetCapabilities'));
    foreach ($mypix->FeatureTypeList as
$pixinfo):
      foreach ($pixinfo->FeatureType as $nam):
        if ($nam->Name!='') echo
$nam->Name,"<br>";
      endforeach;
    endforeach;
  }
  else echo "missing extension<br>";
?>
</body></html>
```

Ayant compris ce mécanisme, il devient simple de générer les champs décrivant chaque couche WMS affichée, en lui attribuant le même nom de couche que celui utilisé dans QGis. Le résultat est :

```
<html><head><title>Toutes les couches WMS depuis
openlayers</title>
  <script src="http://openlayers.org/api/OpenLayers.js">
</script>
</head>
<body>
  <div style="width:100%;height:100%" id="map"></div>
  <script defer="defer" type="text/javascript">
    var map=new OpenLayers.Map('map');
    ls=new OpenLayers.Control.
LayerSwitcher({'div':OpenLayers.Util.
getElement('layerswitcher')});
    map.addControl(ls);
    ls.maximizeControl();
    var wms=new OpenLayers.Layer.WMS("Basic map",
      "http://vmap0.tiles.osgeo.org/wms/vmap0",{layers:
'basic'} );
<?php
  $k=1;
  if (extension_loaded('simplexml')) {
    $mypix = simplexml_load_file(urlencode(
      'http://127.0.0.1/cgi-bin/project/landsat/qgis_
mapserv.fcgi?SERVICE=WFS&VERSION=1.0.0&REQUEST=GetCapabiliti
es'));
    foreach ($mypix->FeatureTypeList as $pixinfo):
      foreach ($pixinfo->FeatureType as $nam):
        if ($nam->Name!='')
        {echo "var dm_",$k,"=new OpenLayers.Layer.
WMS(\"",$nam->Name,"\",";
         echo "\"http://127.0.0.1/cgi-bin/project/landsat/
qgis_mapserv.fcgi\",";
         echo "{layers: \"",$nam->Name,"\",";
         echo "transparent:\"true\", format:\"image/
png\"},{isBaseLayer: false});\n";
         $k++;
        }
      endforeach;
    endforeach;
    echo "map.addLayer(wms);\n";
    echo "map.setCenter(new OpenLayers.LonLat(12,79),9);\n";
    for ($x=1;$x<$k;$x++) {echo "map.addLayer(dm_$x);\n";}
  } else echo "missing extension<br>";
?>
</script></body></html>
```

Ce qui donne la figure 6. Il serait d'une part très fastidieux d'ajouter à la main toutes les couches de ce projet, mais surtout dans cet exemple l'affichage s'adapte automatiquement aux nouvelles couches ajoutées par les divers contributeurs au projet.

6. AJOUT D'IMAGES ORTHORECTIFIÉES PRODUITES PAR MICRODRONE

Nous avions présenté le flux de traitement permettant de générer une série d'images orthorectifiées et de modèles numériques d'élévation associés avec le souci de comparer ces jeux de données entre eux, mais sans prétention de les positionner dans un référentiel absolu partagé par convention entre plusieurs utilisateurs. Il s'avère **[6]** que la solution de simplement translater un jeu de données par rapport à l'autre ne suffit plus lorsque la zone considérée s'étend sur plusieurs hectares. Nous avons observé que dans ce cas, le positionnement par GPS (mono-fréquence, telle que fourni sur microdrone **DJI Phantom3 Professional**) n'est exact qu'à une dizaine de mètres, alors que nous visons un positionnement au décimètre près pour des comparaisons de vols successifs. Il s'avère que vouloir corriger les défauts d'échelle et de position du modèle numérique d'élévation en lui appliquant les corrections permettant de superposer les images orthorectifiées est une approche peu judicieuse, et qu'il vaut mieux insérer dans le flot de traitement l'étape d'exploitation des points de contrôle au sol pour corriger les défauts du modèle de caméra et de leur position (http://forum-micmac.forumprod.com/campari-residuals-differ-from-the-point-cloud-t881.html). Pour notre part, les points de contrôle au sol (GCP) sont acquis a posteriori dans **Google Earth** – les coordonnées sphériques fournies avec 6 décimales sont précises à 11 cm à l'équateur : 7 points aisément reconnaissables dans les jeux de données sont d'une part identifiée dans Google Earth (bandes de parking, barrière, coin à la base de bâtiments) avec des coordonnées converties de coordonnées sphériques (WGS84) vers un référentiel projeté localement tangent à la Terre (WGS84/UTM31N), et d'autre part leur position (en pixel) identifiée sur les images acquises par drone. Ce couple de fichiers (coordonnées dans l'espace v.s coordonnées sur les photographies) alimente l'outil **Campari** de **Micmac**. Après traitement dans ces conditions, nous observons une exactitude de position par rapport aux couches vectorielles de Openstreetmaps mais surtout par rapport à une image aérienne de l'IGN (BD ORTHO, convertie de Lambert93 à WGS84/UTM31N puisque

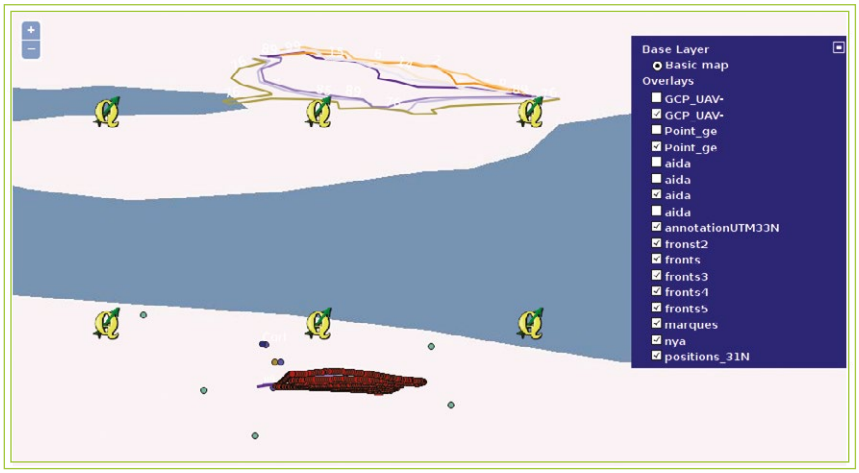


Fig. 6 : Ajout automatique de toutes les couches fournies par un serveur WMS sur dans un contexte d'OpenLayers2. Le script d'exemple /opt/qgis-server/plugins/HelloServer qui affiche le logo de QGis a été volontairement laissé en place pour distinguer le serveur en production (qgis.sequanux.org) du serveur de tests (127.0.0.1).

nous avons vu que OpenLayers 2 ne sait pas le faire à la volée) meilleure qu'une vingtaine de centimètres – erreur attribuable à notre manque d'assiduité à pointer les points de contrôle. Ce jeu de données est disponible sur http://qgis.sequanux.org/cgi-bin/project/femto/qgis_mapserv.fcgi.

CONCLUSION

Nous avons proposé un environnement de travail permettant de disséminer des informations géoréférencées soit vers des utilisateurs de logiciels dédiés de gestion de systèmes d'informations géographiques (SIG) ou vers les API Web associées (OpenLayers). Est-ce que ce

Fig. 7 : Résultat de l'insertion de trois images orthorectifiées, en comparaison de données de référence que sont les images aériennes de la BD ORTHO de l'IGN et les données vectorielles de Opensteetmaps. En haut à gauche : deux images orthorectifiées acquises à 30 minutes d'intervalle positionnées au moyen de 7 GCPs. En haut à droite : le même jeu de données, positionné uniquement par les positions GPS du drone au moment de la prise de vue. Le trait rouge, indiquant la différence de position, est long de 9,8 m. En bas à gauche : superposition d'une image produite par drone (moitié gauche) avec une image IGN (moitié droite). En bas à droite : comparaison de deux orthophotos produites à un mois d'intervalle.

mode de dissémination est réellement utilisé en pratique ? L'institut polaire norvégien (NPI) propose ses informations dans ce format tel que décrit sur http://geodata.npolar.no/#basemap-services. Ainsi, sous QGis, configurer l'onglet ***WM(T)S*** vers http://geodata.npolar.no/arcgis/rest/services/Basisdata/NP_Ortofoto_Svalbard_WMTS_25833/MapServer/WMTS? donne accès aux informations bitmap que sont les photographies aériennes de très haute résolution (jusqu'à 16 cm/pixel !) tandis que l'url http://geodata.npolar.no/arcgis/services/CryoClim/glaciers/MapServer/WmsServer? dans l'onglet ***WFS*** donne accès aux informations vectorielles que sont les limites des glaciers extraites des images satellitaires SPOT.

Le lecteur est encouragé à reproduire ces expériences sur ses propres zones d'intérêt. Compte tenu de la résolution médiocre de Landsat, le résultat est peu convaincant pour la vallée de Chamonix, mais les glaciers groenlandais (Illulisat près de la baie de Disco) ou la banquise antarctique sont parfaitement appropriés à cette démonstration, voir la mer d'Aral sur http://www.nasa.gov/mission_pages/landsat/news/40th-top10-aralsea.html (un peu pénible à assembler car de superficie supérieure à ce que le site de l'USGS permet d'analyser) ou la mer morte sur http://earthobservatory.nasa.gov/IOTD/view.php?id=77592 et pour laquelle l'extension des bassins d'exploitation du sel est particulièrement visible. Le lecteur saura-t-il calculer la surface ainsi affectée ? Indice : tracer un polygone avec l'outil ***New Shapefile*** puis dans la table des attributs, créer un nouveau champ pour chaque polygone avec la variable **`$area`**.

Une autre voie d'étude qui semble intéressante dans le cadre de la diffusion sur le Web de données géoréférencées est **qgis2web**, greffon de QGis actuellement inexploitable sous Debian testing/sid compte tenu d'une dépendance cassée. Finalement, étant donné que toutes les informations pertinentes à une utilisation en surface sont disponibles sur OpenStreetMaps, ne serait-il pas temps de commencer à considérer la cartographie des services sous terrains (eau, gaz, électricité), qui même s'ils sont propriété de leurs exploitants respectifs, sont devenus des services nécessaires au quotidien d'un habitant d'Europe occidentale actuel : chaque kilomètre de route contient plusieurs dizaines de kilomètres de services sous terrains **[7]** qui ne demandent qu'à être cartographiés à l'occasion des ouvertures de routes. ■

RÉFÉRENCES ET NOTES

[1] PIERROT-DESEILLIGNY M. et FRIEDT J.-M, *« La photogrammétrie pour tous : MicMac pour la reconstruction 3D de scènes géoréférencées à partir de photographies numériques »*, tutorial à FOSS4G-fr 2016, décrit sur jmfriedt.free.fr/foss4g_2016

[2] FRIEDT J.-M. Friedt, TOLLE F., et BERNARD É., *« Utilisation de Micmac pour la génération de modèle numérique d'élévation par traitement d'images acquises par microdrone »*, GNU/Linux Magazine France 191, pp.48–57 (Mars 2016)

[3] ERLE S., GIBSON R., et WALSCH J., *« Building the geospatial web »*, dans *« Mapping Hacks – Tips & Tools for Electronic Cartography »*, O'Reilly (2005)

[4] GIBSON R. et ERLE S., *« Google Maps Hacks – Tips & Tools for Geographic Searching and Remixing »*, O'Reilly (2006)

[5] Les divers champs se déduisent de l'analyse de la sortie de **`wget -O - "http://127.0.0.1/cgi-bin/mon_projet/qgis_mapserv.fcgi?SERVICE=WFS&VERSION=1.0.0&REQUEST=GetCapabilities"`**

[6] LISEIN J., PINEUX N., PIERROT-DESEILLIGNY, DEGRÉ A., et LEJEUNE P., *« Détection de l'érosion dans un bassin versant agricole par comparaison d'images multidates acquises par drone »*, Colloque Drones et moyens légers aéroportés d'observation, 26/06/2014 (Montpellier), disponible sur http://orbi.ulg.ac.be/handle/2268/171616

[7] Chaque kilomètre de route à Hong-Kong recouvre 47 km de services enterrés, http://www.uti.hk/media/attachments/2nd_ICUMAS_full_paper.pdf ou http://www.scmp.com/news/hong-kong/article/1647088/small-army-utility-specialists-keeps-hongkongers-safe-pipes-cables, ou en d'autres termes il existe 47 câbles, tuyaux, fibres optiques et autres modes de transmission de fluides et d'informations sous chaque route.

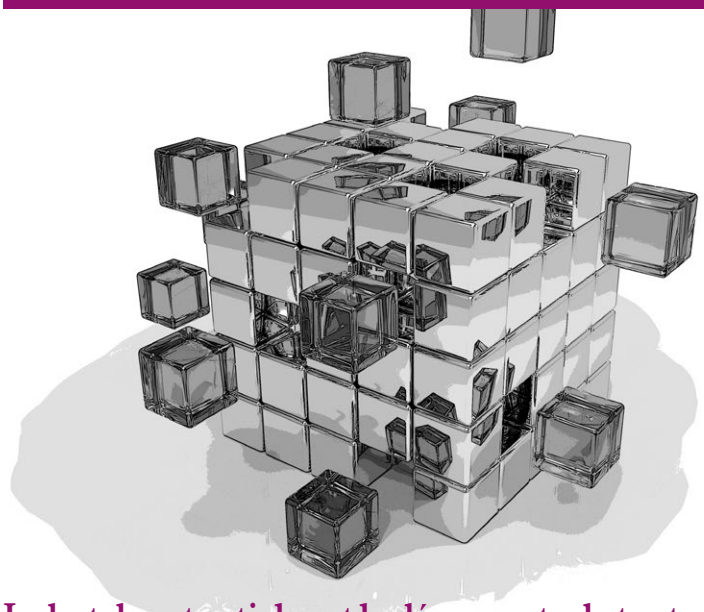
REMERCIEMENTS

J.-P Culas (CM-Drones, Besançon) a effectué les vols au-dessus du parking de FEMTO-ST. Les membres du forum Micmac (http://forum-micmac.forumprod.com/) ont une fois de plus partagé leurs connaissances pour corriger les défaillances dans les traitements que nous proposions initialement.

À LA DÉCOUVERTE DE SYSTEMD

ISSAM MEJRI
[Linuxien depuis 2005]

MOTS-CLÉS : INIT, SYSTEMD, SERVICES, SYSTEMCTL, SYSTEMD-ANALYZE



Le but de cet article est la découverte de toutes les faces cachées de systemd et particulièrement l'initialisation du système et l'administration des services, l'optimisation des performances et la gestion élémentaire des journaux.

Après une longue réflexion, j'ai décidé de rédiger un article sur **systemd**. Pourquoi ? Tout d'abord, pour une raison évidente, puisque ce dernier a envahi aujourd'hui tout ou presque la plupart des distributions GNU/Linux, notamment RHEL 7, CentOS sans oublier Debian et Ubuntu. D'autre part, il est nécessaire pour un utilisateur régulier de Linux ou certainement pour un administrateur de migrer vers systemd et de comprendre sa nouvelle philosophie pour la gestion du système. Eh oui, je peux vous affirmer que systemd est une révolution pour les systèmes GNU/Linux, non pas parce qu'il remplace le vieux init, mais parce qu'il incarne la majorité des fonctionnalités du système : la gestion des ressources, l'arrêt et le démarrage des services, la détection des périphériques, la journalisation, la virtualisation par conteneur... Bref, systemd malgré les polémiques qu'il a déclenché depuis sa naissance au sein de la firme Red Hat, est présent malgré tout au cœur de l'OS Linux.

1. COUP D'OEIL SUR SYSTEMD

« Systemd est un système d'initialisation et un gestionnaire de session pour Linux », son nom est un raccourci de *system daemon*, je n'ai rien inventé, c'est la description traduite donnée par l'auteur de systemd lui-même : Lennart Poettering, un ingénieur allemand recruté par **Red Hat**.

Concrètement, il s'agit d'un ensemble de binaires assurant le fonctionnement du système. Il est réputé en tant que remplaçant du système d'initialisation **SysVInit** et par la suite **Upstart**. C'est le premier processus démarré (PID=1) et il active ensuite les autres services. L'ancien init démarre séquentiellement les services en se basant sur les scripts init configurés pour être lancés dans le niveau d'exécution par défaut. Un tel démarrage est relativement long, car il dépend des éventuels processus à lancer au démarrage. Si un service ne peut plus démarrer pour une raison ou une autre (réseau non activé par exemple), le démarrage est suspendu jusqu'au *timeout* défini, ce qui ralentit énormément la phase de démarrage. C'est un inconvénient parmi d'autres laissant une image dépréciée de SysVInit. La lecture et l'exécution des scripts shell sont aussi des causes non négligeables.

C'est une des raisons pour lesquelles systemd est né dans la firme de Red Hat. Il est toujours compatible avec SysV et LSB init scripts et il est caractérisé par :

- les scripts init sont remplacés par des unités à configuration déclarative et ceci pour chaque service système, mais également pour d'autres composants, tels que les sockets et le montage des systèmes de fichiers, l'activation des zones de swap, les *timers* (crontab) et les niveaux d'exécution qui seront désormais appelés unités cibles ;
- des fonctionnalités de parallélisation, accélérant la vitesse de démarrage d'un système ;
- le démarrage à la demande des démons sans recours à un service distinct ;
- la gestion automatique de la dépendance des services, qui peut éviter les longues périodes d'attente, en ne démarrant pas un service en réseau lorsque celui-ci n'est pas disponible, par exemple ;
- assure la configuration des noms d'hôtes et le montage des systèmes de fichiers ;
- gère la journalisation et peut transmettre les journaux à syslog ;
- gère les sessions utilisateurs et leurs ressources via les groupes de contrôle Linux ;
- une méthode de suivi groupée des processus liés à l'aide des groupes de contrôle Linux ;
- gère les machines virtuelles et conteneurs via les cgroups et libvirt.

2. ARCHITECTURE DE SYSTEMD

Dans cet article, c'est la distribution GNU/Linux CentOS 7 qui a été utilisée dans mon modeste atelier. En effet, cette distribution incarne en natif systemd et par conséquent, l'ensemble des services systèmes seront gérés par systemd, et non plus par les scripts d'initialisation (démarrage, arrêt et rechargement) situés classiquement dans **/etc/init.d/**. Commençons tout d'abord par inspecter les paquets livrés avec notre fameux systemd :

- Les différents fichiers installés en même temps que systemd peuvent être découverts par la commande :

```
# rpm -ql  systemd
    ###### Les binaires systemd #####
/usr/bin/busctl
/usr/bin/coredumpctl
/usr/bin/hostnamectl
/usr/bin/journalctl
/usr/bin/kernel-install
...
```

- Les fichiers de configuration relatifs à systemd sont :

```
# rpm -ql  systemd | grep etc
   ###### Les fichiers de configuration systemd ######
/etc/systemd
/etc/systemd/bootchart.conf
/etc/systemd/coredump.conf
/etc/systemd/journald.conf
...
```

Le fichier **/etc/systemd/system.conf** est le fichier maître de la configuration de systemd, il définit ainsi les paramètres principaux d'exécution et d'environnement de ce dernier. On peut noter aussi le fichier **/etc/systemd/journald.conf** qui définit également les options liées à la journalisation du service, **systemd-journald** (voir plus loin dans l'article), sans oublier la commande magique **/usr/bin/systemctl** (se prononce *system control* pour les anglophones), on découvrira par la suite que cette commande est l'outil principal pour la gestion d'un système disposant de systemd.

La configuration des différents services (le remplaçant des scripts init) est située dans le répertoire **/usr/lib/systemd/system**. Dans ce répertoire, tous les fichiers avec l'extension **.service** représentent les fichiers d'initialisation des services, c'est l'équivalent des scripts init System V, ils renseignent les informations sur la manière de démarrer, d'arrêter et de recharger un service, mais aussi d'autres paramètres dont nous parlerons plus tard.

On remarque aussi qu'on a des fichiers à l'extension **.socket**, **.mount**, **.target**, **.path**, **.timer**. En effet, systemd, outre la gestion des services, est aussi le responsable d'autres tâches systèmes tels que le montage des systèmes de fichiers, la planification des tâches, l'arrêt et le démarrage du système, la gestion des fichiers temporaires, l'écriture dans les journaux... bref systemd est partout !

L'ensemble des fichiers précédents s'appelle des unités et chaque type d'unité est dédiée à la réalisation d'une tâche précise :

1. Le type service : les unités de service servent à démarrer les démons auxquels l'accès est fréquent, comme un serveur web par exemple.

2. Le type socket : ces unités portent une extension **.socket** et représentent les sockets de communication interprocessus (IPC). Le contrôle des sockets est transféré à un démon ou à un service démarré exprès lorsqu'un client se connecte. Les unités de socket servent à lancer un service au démarrage et à lancer les services moins fréquemment utilisés à la demande. Celles-ci sont similaires dans le principe aux services qui utilisent le super-serveur xinetd pour démarrer un service à la demande et lorsque son socket est sollicité.

3. Le type **.path** : on les appelle les unités de chemin, elles servent à retarder l'activation d'un service jusqu'à ce que se produise une modification spécifique du système de fichiers. C'est un usage courant pour les services qui utilisent les dossiers spool (file d'attente), comme les systèmes d'impression.

4. Le type **.target** : regroupe plusieurs unités et permet de définir des notions de niveau d'exécution.

5. Le type **.mount** : ce type d'unité permet la gestion des systèmes de fichiers en relation avec le fichier fstab.

6. Le type timer : permet de réaliser une partie de fonctionnalité de cron depuis la version 212 de systemd.

Voici un exemple d'un fichier d'unité de service relatif au serveur web httpd :

```
# vim /usr/lib/systemd/systemd/httpd.service
```

```
[Unit]
Description=The Apache HTTP Server
After=network.target remote-fs.target nss-lookup.target
Documentation=man:httpd(8)
Documentation=man:apachectl(8)

[Service]
Type=notify
EnvironmentFile=/etc/sysconfig/httpd
ExecStart=/usr/sbin/httpd $OPTIONS -DFOREGROUND
ExecReload=/usr/sbin/httpd $OPTIONS -k graceful
ExecStop=/bin/kill -WINCH ${MAINPID}
# We want systemd to give httpd some time to finish
gracefully, but still want
# it to kill httpd after TimeoutStopSec if something went
wrong during the
# graceful stop. Normally, Systemd sends SIGTERM signal
right after the
# ExecStop, which would kill httpd. We are sending
useless SIGCONT here to give
# httpd time to finish.
KillSignal=SIGCONT
PrivateTmp=true

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

On remarque déjà une grande différence par rapport aux scripts init qui comportent souvent des centaines de lignes pour la simple gestion d'un service et de son environnement d'exécution. Ici, la configuration est simple à comprendre et elle est totalement déclarative, ce qui évite énormément le risque d'erreur d'une part et la rapidité d'exécution au démarrage système d'autre part.

Un fichier d'unité est divisé en trois sections :

- La section **[Unit]** : comprend une description de l'unité elle-même et les règles de dépendances associées, en effet systemd effectue une gestion intelligente de dépendance entre les services systèmes. Ceci permet la mise en place des règles pour le démarrage éventuel d'un service. Par exemple, il est judicieux de démarrer le service réseau avant que le serveur SSH ou le serveur web soient démarrés, ainsi on évite qu'un service échoue après un timeout pendant lequel il est en attente d'un autre service qui lui est nécessaire pour son exécution.

  La directive **After** plus haut indique bien que le serveur web démarrera après le démarrage des unités **network.target**, **remote-fs.target** et **nss-lookup.target** : c'est le principe de dépendance introduit par systemd.

- La section **[Service]** : comprend les commandes de démarrage, d'arrêt et de rechargement du service ainsi que le signal à envoyer lors de l'utilisation de la commande **kill** au PID du processus (service).

  La directive **ExecStart** indique la commande relative au lancement du service.

  La directive **ExecReload** indique ce qu'il faut faire pour une prise en charge d'une modification de la configuration du service.

  La directive **ExecStop** permet d'indiquer une commande à exécuter pour arrêter le service.

- La section **[Install]** : indique le comportement d'un service suite à son activation ou à sa désactivation. Dans notre cas, la directive **WantedBy** indique que le service est nécessaire à la cible **multi-user.target**. On peut considérer **multi-user.target** comme étant le niveau d'exécution (hommage à l'ancien init !) 3, c'est-à-dire sans interface graphique. Pas de panique, nous aborderons cette notion plus tard.

Les paramètres des sections décrits précédemment seront différents d'un type d'unité à l'autre et un fichier d'unité peut avoir un nombre important de paramètres par défaut qui n'ont pas été décrits dans le fichier pour des raisons de clarté, mais il est possible bien sûr de les changer si nécessaire et selon le besoin.

3. LA GESTION DES SERVICES

Le démarrage du système et les processus du serveur sont gérés par le gestionnaire de système et de services systemd. Ce programme fournit une méthode permettant d'activer les ressources du système, les démons du serveur et autres processus à la fois lors du démarrage et sur un système en cours d'exécution. Les démons sont des processus qui attendent ou s'exécutent en arrière-plan pour effectuer plusieurs tâches. Habituellement, les services s'exécutent automatiquement au démarrage et poursuivent leur exécution jusqu'à l'extinction ou jusqu'à ce qu'ils soient arrêtés manuellement.

Avec systemd, les scripts de services basés sur le shell ne sont plus utilisés que pour quelques services anciens. Ces derniers sont désormais remplacés par les fichiers d'unités vus dans le paragraphe précédent.

3.1 État d'un service

La commande **systemctl** est une commande universelle permettant d'interagir avec le système et entre autres l'affichage et la modification d'état d'un service.

Pour visualiser l'état d'un service :

```
# systemctl status httpd.service
● httpd.service - The Apache HTTP Server
   Loaded: loaded (/usr/lib/systemd/system/httpd.service;
disabled; vendor preset: disabled)
   Active: active (running) since Sat 2016-06-11 00:33:45 GMT;
8s ago
     Docs: man:httpd(8)
           man:apachectl(8)
  Process: 22082 ExecStop=/bin/kill -WINCH ${MAINPID}
(code=exited, status=1/FAILURE)
 Main PID: 22123 (httpd)
   Status: "Processing requests..."
   CGroup: /system.slice/httpd.service
           ├─22123 /usr/sbin/httpd -DFOREGROUND
           ├─22126 /usr/sbin/httpd -DFOREGROUND
           ├─22128 /usr/sbin/httpd -DFOREGROUND
           ├─22130 /usr/sbin/httpd -DFOREGROUND
           ├─22131 /usr/sbin/httpd -DFOREGROUND
           └─22132 /usr/sbin/httpd -DFOREGROUND
  #####   Les messages journaux ######
Jun 11 00:33:45 ldap.pme.com httpd[22123]: [Sat Jun 11
00:33:45.593866 2016] [so:warn] [pid 22123] AH01574:...pping
Jun 11 00:33:45 ldap.pme.com httpd[22123]: AH00558: httpd: Could
not reliably determine the server's fully ...ssage
Hint: Some lines were ellipsized, use -l to show in full.
```

C'est magique, ici on peut apprécier le changement par rapport à l'affichage des anciennes commandes telles que « service httpd status ». Le résultat renvoyé ne comprend pas uniquement l'état du service, mais également d'autres informations sur le ou les processus lancés et leur PID respectif, le groupe de contrôle auquel appartient le service (**system.slice** dans notre cas), depuis quand ce service est actif, mais aussi les messages journaux générés et qui sont utiles pour un éventuel débogage en cas de problème. Il n'est alors pas la peine de fouiller dans **/var/log/messages** ou encore dans les fichiers journaux propres au service ! Finalement, on a une image plus parfaite sur un service en cours d'exécution.

La sortie d'état comporte plusieurs mots-clés indiquant l'état du service :

- Loaded : le fichier de configuration de l'unité a été traité ;
- Active (running) : en cours d'exécution avec un ou plusieurs processus qui se poursuivent ;
- Active (exited) : a terminé une configuration ponctuelle avec succès ;
- Active (Waiting) : en cours d'exécution, mais en attente d'un évènement ;
- Inactive : pas exécuté ;
- Enabled : sera lancé au démarrage ;
- Disabled : ne sera pas lancé au démarrage ;
- Static : ne peut pas être activé, mais peut être démarré par une unité activée automatiquement.

3.2 Démarrage, arrêt et redémarrage d'un service

Lorsqu'on modifie un fichier de configuration ou que l'on procède à d'autres mises à jour d'un service, il peut être nécessaire de redémarrer le service. Un service qui n'est plus utilisé peut être interrompu avant de désinstaller le logiciel. Un service rarement utilisé peut être démarré manuellement par un administrateur uniquement quand on a besoin.

Pour redémarrer le service httpd :

```
# systemctl restart httpd
```

Pour recharger le fichier de configuration de httpd :

```
# systemctl reload httpd
```

Lorsque l'extension est omise, c'est le **.service** qui sera la valeur par défaut.

Pour stopper le serveur web :

```
# systemctl stop httpd
```

Vérifions le nouvel état :

```
# systemctl status httpd
httpd.service - The Apache HTTP Server
   Loaded: loaded (/usr/lib/systemd/system/httpd.
service; disabled; vendor preset: disabled)
   Active: inactive (dead)
```

Pour désactiver complètement le lancement d'un service, que ce soit manuellement ou au démarrage :

```
systemctl mask httpd.service
```

Pour rétablir un service masqué :

```
systemctl umask httpd.service
```

3.3 Activation, désactivation d'un service au démarrage

Le lancement d'un service sur un système en cours d'exécution ne garantit pas son lancement au redémarrage de la machine. De même, l'interruption d'un service sur un système en cours d'exécution ne l'empêchera pas de redémarrer en même temps que le système. Les services s'exécutent au démarrage lorsque des liens sont créés dans les répertoires de configuration systemd appropriés. On crée ces liens et on les supprime encore une fois à l'aide de la commande **systemctl**.

Pour rendre un service disponible à chaque redémarrage, toujours avec le service httpd :

```
# systemctl enable httpd.service
```

Pour désactiver ce même service :

```
# systemctl disable httpd.service
```

4. UNITÉ CIBLE ET NIVEAU D'EXÉCUTION

Maintenant avec systemd, on ne parlera plus ou presque de niveau d'exécution que pour des raisons de compatibilité avec l'ancien processus init. En effet, on n'aura plus recours au fichier **/etc/inittab** et sa présence sur le système est même inutile. L'unité cible ou target pourra désigner l'état d'un système. Par analogie à System V, les niveaux d'exécution seront nommés comme suit :

- le niveau d'exécution 0 qui correspond à l'arrêt système sera **poweroff|shutdown.target** ;
- le niveau d'exécution 1 qui correspond au mode de secours sera **rescue|emergency.target** ;
- le niveau d'exécution 3 qui correspond au mode multi-utilisateur sans serveur graphique sera **multi-user.target** ;
- le niveau d'exécution 5 qui correspond au mode de connexion graphique sera **graphical.target** ;
- le niveau d'exécution 6 qui correspond au redémarrage du système sera **reboot.target**.

Pour afficher le niveau d'exécution par défaut, tapez cette commande :

```
# systemctl get-default
graphical.target
```

On est en mode graphique.

Pour basculer en mode multi-utilisateur :

```
# systemctl set-default multi-user.target
```

Donc au prochain redémarrage, c'est le niveau d'exécution 3 qui sera pris en compte.

Si on veut changer le niveau d'exécution à chaud, pour des travaux de maintenance par exemple :

```
# systemctl isolate rescue.target
```

Remarque : Il est aussi possible de basculer d'un mode à l'autre depuis GRUB en passant un paramètre au noyau Linux via : **systemd.unit = rescue.target** puis [CTRL+X] pour démarrer.

Un fichier d'unité cible ou target tel que **graphical.target** se présente comme suit :

```
[Unit]
Description=Graphical Interface
Documentation=man:systemd.special(7)
Requires=multi-user.target
Wants=display-manager.service
Conflicts=rescue.service rescue.target
After=multi-user.target rescue.service rescue.target
display-manager.service
AllowIsolate=yes
```

C'est simple à comprendre, la cible **graphical.target** suppose que la cible **multi-user.target** soit atteinte pour quelle soit activée, et ceci grâce à la directive **Requires**. La directive **AllowIsolate** est très importante, car c'est celle-ci qui permet de basculer vers le mode **graphical.target** si elle est initialisée à **yes** bien évidemment.

5. ADMINISTRATION DES UNITÉS

Systemd nous facilite énormément la vie en proposant des commandes simples à retenir, mais d'une robustesse sans égale, puisqu'on peut administrer l'ensemble des objets systemd en passant par l'unique commande **systemctl** (encore !). Dans cette partie, j'expose une liste non exhaustive des commandes qui vous seront utiles au quotidien.

Pour afficher la liste des unités de service :

```
# systemctl -t service
UNIT                    LOAD   ACTIVE SUB       DESCRIPTION
avahi-daemon.socket     loaded active running   Avahi mDNS/DNS-SD Stack
Activation Socket
cockpit.socket          loaded active listening Cockpit Web Service Socket
smb.service             loaded active running Samba SMB Daemon
sshd.service            loaded active running OpenSSH server daemon
...
```

même chose pour les sockets :

```
# systemctl -t socket
UNIT                  LOAD   ACTIVE SUB       DESCRIPTION
avahi-daemon.socket   loaded active running   Avahi mDNS/DNS-SD Stack
Activation Socket
cockpit.socket        loaded active listening Cockpit Web Service Socket
cups.socket           loaded active running   CUPS Printing Service Sockets
dbus.socket           loaded active running   D-Bus System Message Bus Socket
...
```

Pour inspecter suivant l'état de l'unité, si un service a échoué par exemple :

```
# systemctl -state failed
 UNIT          LOAD   ACTIVE SUB    DESCRIPTION
● kdump.service loaded failed failed Crash recovery kernel arming

LOAD   = Reflects whether the unit definition was properly loaded.
ACTIVE = The high-level unit activation state, i.e. generalization of SUB.
SUB    = The low-level unit activation state, values depend on unit type.
```

Plus haut dans l'article, nous avons abordé la directive **Requires** permettant de savoir quelle sont les unités nécessaires pour d'autres unités, le même résultat peut être atteint en passant cette commande :

```
# systemctl show -p
"Requires" graphical.
target
Requires=multi-user.
target
```

Par contre, si on veut afficher tous les attributs d'une unité :

```
# systemctl show httpd.
service
Type=notify
Restart=no
NotifyAccess=main
RestartUSec=100ms
TimeoutStartUSec=1min 30s
TimeoutStopUSec=1min 30s
...
```

Comme prévu, on aperçoit dans les dernières lignes les directives déjà vues dans le fichier d'unité **httpd.service** et l'ensemble des attributs par défaut.

Systemd gère les dépendances entre les services, pour afficher l'arbre de dépendances pour le service httpd :

```
# systemctl list-
dependencies httpd.service
httpd.service
● ├─-.mount
● ├─system.slice
● └─basic.target
●   ├─alsa-restore.service
●   ├─alsa-state.service
●   ├─firewalld.service
...
```

Pour affiner la requête précédente, on pourra lister les unités à activer avant le démarrage du serveur web httpd :

```
# systemctl list-
dependencies --after
httpd.service
```

Le serveur web doit également démarrer avant l'activation des unités suivantes :

```
# systemctl list-dependencies --before httpd.service
httpd.service
● └─shutdown.target
●   ├─systemd-reboot.service
●   └─final.target
●     └─systemd-reboot.service
```

Pour afficher une liste des sockets avec leur type correspondant :

```
# systemctl --show-type  list-sockets
LISTEN                   TYPE      UNIT                    ACTIVATES
/dev/log                 Datagram  systemd-journald.socket  systemd-journald.service
/run/dmeventd-client     FIFO      dm-event.socket          dm-event.service
/run/dmeventd-server     FIFO      dm-event.socket          dm-event.service
/run/lvm/lvmetad.socket  Stream    lvm2-lvmetad.socket      lvm2-lvmetad.service
...
```

6. SUPERVISER AVEC SYSTEMD

L'un des avantages de systemd est de permettre un démarrage rapide du système. Ceci dit, c'est grâce à la parallélisation du lancement des services et au démarrage à la demande qu'on peut atteindre un temps de démarrage plus faible. Pour pouvoir superviser le temps de démarrage du système, systemd dispose d'une commande analysant le processus de démarrage, en spécifiant également le temps consommé pour chaque service. On peut alors détecter facilement lequel des processus a mis plus de temps pour s'activer. La commande est :

```
# systemd-analyze blame
1min 8.023s kdump.service
         17.544s openvpnas.service
          8.734s plymouth-quit-wait.service
          7.840s network.service
          7.390s firewalld.service
...
```

On constate que le système a mis une minute pour démarrer et que le service Openvpnas a consommé 17 secondes pour démarrer. Si on désactive ce service, le processus de démarrage s'accélère certainement.

Vérifions ceci avec la même commande :

```
# systemd-analyze time
Startup finished in 1.027s (kernel) + 2.643s (initrd) +
41.396s (userspace) = 45.067s
```

Systemd nous fournit un outil qui nécessite un article entier, c'est la commande **journalctl**, cette commande permet de fouiller dans les logs du système pour détecter une éventuelle erreur pour un service particulier. Les messages du Log sont traités par les deux services : **systemd-journald** et **rsyslog**.

Le démon **systemd-journald** propose un service de gestion des journaux amélioré qui collecte les messages du noyau des toutes premières étapes de processus de démarrage, de la sortie standard et de l'erreur standard des démons à leur démarrage et à leur exécution, et de syslog. Il écrit ces messages dans un journal d'évènements structuré qui, par défaut, n'est pas persistant d'un démarrage à l'autre. Cela permet aux messages syslog et aux évènements ratés par syslog d'être recueillis dans une base de données centrale. Les messages syslog sont également transmis par **systemd-journald** pour un traitement supplémentaire.

La commande **journalctl** permet d'afficher les messages du démarrage courant :

```
# journalctl -b
```

Pour visualiser les messages de démarrage ayant la priorité **error** et supérieur :

```
# journalctl -b -p err
```

Pour inspecter les journaux d'un service particulier :

```
# journalctl -u httpd
```

L'équivalent da la commande **tail -f /var/log/messages** est :

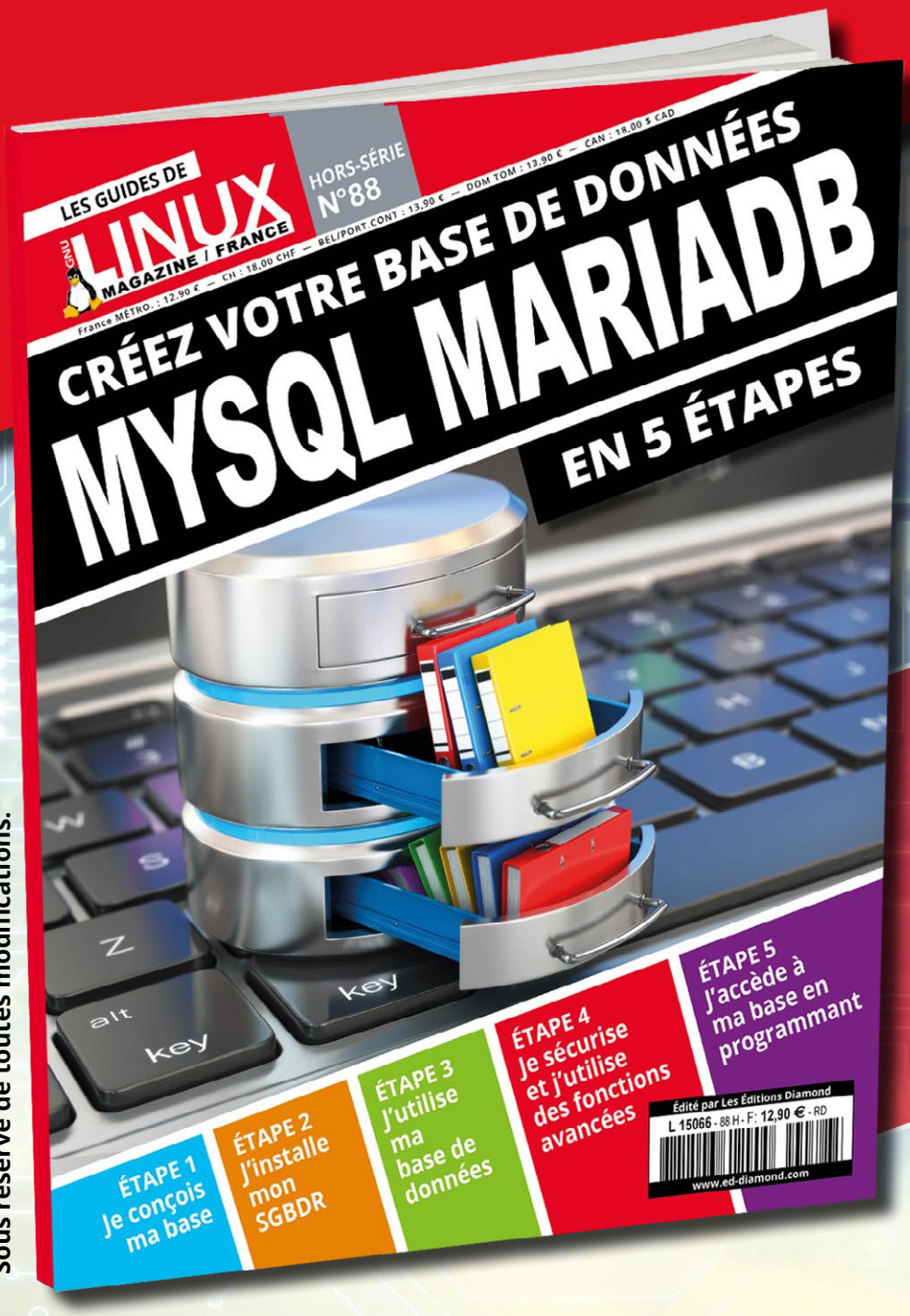
```
# journalctl -f
```

CONCLUSION

Systemd, le remplaçant d'init, mérite d'être présent dans nos distributions GNU/Linux (la distribution Debian 8 Jessie est livrée avec systemd et prochainement toutes les distributions GNU/Linux). Malheureusement, les UNIX ne sont pas capables d'adopter ce nouveau-né, car systemd est fondé sur les groupes de contrôles qui sont disponibles uniquement dans le noyau Linux, c'est la raison pour laquelle systemd a provoqué une polémique dans la communauté open source dès sa sortie. Mais aujourd'hui, je pense que la plupart ont été convaincus par ce système à moins qu'il ne soit remplacé par une autre technologie dans le futur... Qui sait ? ■

DISPONIBLE DÈS LE 13 JANVIER

GNU/LINUX MAGAZINE HORS-SÉRIE N°88 !



CRÉEZ VOTRE BASE DE DONNÉES MYSQL / MARIADB EN 5 ÉTAPES

NE LE MANQUEZ PAS

CHEZ VOTRE MARCHAND DE JOURNAUX ET SUR :

http://www.ed-diamond.com

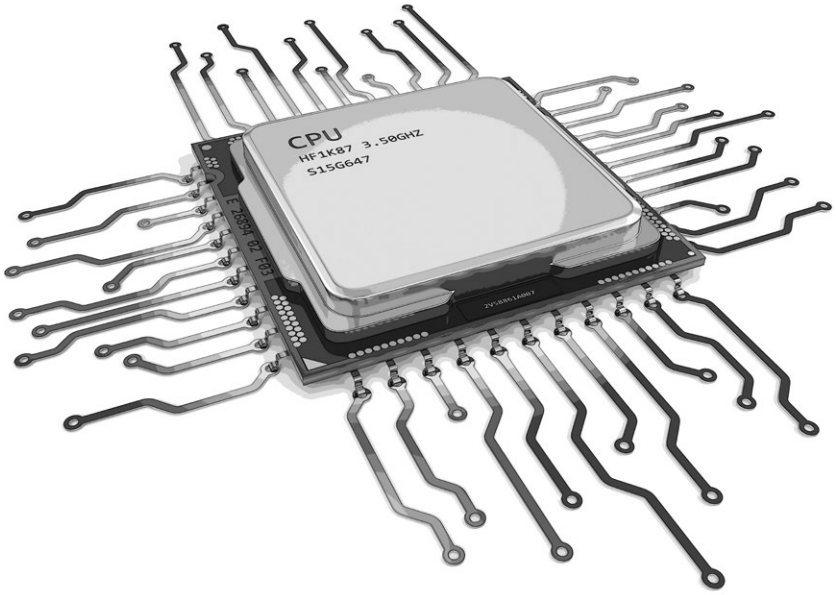
LE RELAIS 2 X 5 V ...

DANS L'IOT OU L'ART DE PILOTER EN BLE LES PÉRIPHÉRIQUES DE LA WARP7

PIERRE-JEAN TEXIER
[Ingénieur Linux Embarqué, Intervenant Linux Embarqué à ESTEI Bordeaux , Co-auteur de l'ouvrage « Yocto For Raspberry-pi »]

JEAN CHABRERIE
[Ingénieur Systèmes Embarqués, Qt Enthousiaste]

MOTS-CLÉS : MIKROBUS, WARP7, IOT, BLUETOOTH LOW ENERGY



L'important choix de cartes intégrant des capteurs/actionneurs, en fait aujourd'hui, un choix difficile pour l'utilisateur final. C'est pour ces raisons que MikroElektronika a créé un standard : MikroBUS. Il facilite l'interaction entre microcontrôleur ou microprocesseur et les cartes d'extensions, appelées « add-ons », utilisant cette connectique.

Cet article se propose d'explorer la partie MikroBUS d'un des tous derniers SBC (Single Board Computer) du marché : la WaRP7 (« WearAble Reference Platform »). On commencera dans un premier temps par une succincte présentation de la WaRP7 et d'une partie « board bring-up » (via Yocto/OE). Puis, viendra la partie MikroBUS qui sera mise en avant à travers un mini projet architecturé autour du Bluetooth Low Energy et du framework Qt5 pour Android tout en y intégrant une carte add-ons, carte qui se base sur le standard MikroBUS.

Le présent article s'inscrit dans la continuité des manipulations ayant permis la rédaction d'un article précédent paru dans le n°20 d'**Open Silicium [1]**, les auteurs renverront vers celui-ci les lecteurs qui désirent avoir de plus amples connaissances sur la diversité que propose la cible utilisé dans ce numéro de Linux Magazine. Dans le premier article, il était question de découvrir la **WaRP7** et ses capteurs au travers une

application « IoT » minimaliste (récupération d'une température, la pression atmosphérique ainsi qu'une valeur représentant le rythme cardiaque), ceci en utilisant la connectivité Bluetooth et le *framework* **Qt5**, le tout basé sur une distribution **Yocto/OE** mis à disposition par les auteurs.

Que le lecteur ne pouvant pas se procurer le premier article soit rassuré, on se permettra tout de même ici de faire des rappels (ouf !) quant aux éléments essentiels (présentation de la cible, génération et installation de l'image sur notre cible, une partie Bluetooth, puis quelques mots sur le SDK Yocto/Qt5).

1. INTRODUCTION

1.1 La cible : « WaRP7 », petit rappel

La plateforme cible **[2]** est composée de 2 cartes :

- Une carte « **fille** », qui est construite autour d'un **System on Chip NXP i.MX7 Solo [3]** (avec un coeur Cortex A7 + un coeur Cortex M4). De plus, cette carte embarquera la gestion de la connectivité (Wifi/BLE) ;
- Une carte « **mère** », qui contiendra l'ensemble des capteurs (Gyroscope, Altimètre, etc.), ainsi que l'extension **MikroBUS** (qui nous intéresse particulièrement dans cet article).

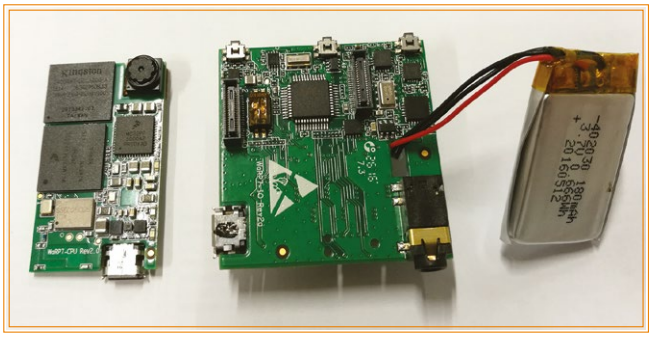


Fig. 1: Non, la WaRP7 n'est pas timide

1.2 Construction de notre distribution « IoT »

La génération de la distribution pour notre cible, repose sur l'utilisation du BSP (*Board Support Package*) NXP articulé autour du projet Yocto **[4]** et de deux couches supplémentaires (nous ne ferons pas ici, de détails quant à l'utilisation de Yocto/OE, le lecteur pourra en outre, se référer aux différents articles parus dans les précédents numéros d'open silicium).

La première étape consiste donc à récupérer l'ensemble des sources contenues dans le BSP. Pour ce faire, NXP utilise l'utilitaire **repo** (outil **Python** développé par **Google [5]**), ceci afin de permettre une meilleure gestion des référentiels **Git** compris dans celui-ci :

```
$ mkdir ~/bin
$ curl http://commondatastorage.googleapis.com/
git-repo-downloads/repo >  ~/bin/repo
$ chmod a+x ~/bin/repo
$ PATH=${PATH}:~/bin
$ mkdir warp7_glmf200
$ cd warp7_glmf200
$ repo init -u https://github.com/Freescale/
fsl-community-bsp-platform -b krogoth
$ repo sync
```

Comme prévu, après récupération de notre base logicielle, nous pouvons maintenant télécharger les deux couches supplémentaires essentielles à la bonne construction de notre image.

Téléchargeons dans un premier temps la couche distribution mis à disposition par les auteurs **[6]**, pour rappel, cette couche spécifique est intimement liée à l'article (ceci afin de ne pas se perdre dans le *framework* qu'est Yocto/OE) :

```
$ cd sources/
$ git clone https://github.com/bdx-iot/meta-iot.git
```

Dans un second temps, récupérons la couche permettant d'intégrer l'environnement Qt5 à notre distribution :

```
$ cd sources/
$ git clone https://github.com/meta-qt5/meta-qt5.git
```

Nous voilà maintenant en possession de l'ensemble des sources. L'étape d'après consiste en la création de l'environnement, on retrouve ci-après les différentes étapes :

- création du dossier de construction **warp7-build/**,
- appel du script **oe-init-build-env**,
- mise à jour de la variable **MACHINE**,
- puis un prompt concernant la licence FSL EULA (à accepter).

La commande suivante s'occupera donc de nous placer dans un environnement de travail spécifique à notre plateforme de développement :

```
$ MACHINE=imx7s-warp source setup-environment
warp7-build/
```

Il reste maintenant à définir les chemins vers les couches précédemment téléchargées, pour ce faire, modifions le fichier **conf/bblayers.conf** pour y rajouter les deux lignes suivantes :

```
${BSPDIR}/sources/meta-iot \
${BSPDIR}/sources/meta-qt5 \
```

Dernière étape avant de pouvoir lancer la construction de l'image, il nous faut spécifier la distribution utilisée (dans notre cas une distribution spécifique à l'article dérivé de la distribution **poky**) dans **conf/local.conf**, en mettant à jour la variable **DISTRO** :

```
DISTRO ?= 'iot'
```

Lançons maintenant la construction de notre image par le biais de la commande **bitbake** :

```
$ bitbake iot-qt-image
```

Après cette longue étape qu'est la construction, l'utilisateur pourra retrouver très facilement l'ensemble des fichiers utiles au bon démarrage de la plateforme (Bootloader, Kernel, RootFileSystem) dans **tmp/deploy/images/imx7s-warp**.

1.3 Premier Flash, premier Boot, ...

Rappelons qu'afin de mettre à jour le système avec l'ensemble des binaires compilés, les étapes se déroulent comme suit :

- Connexion en liaison série (via un émulateur de terminal comme **microcom**), puis arrêt de l'auto-boot au niveau du bootloader (u-boot) :

```
$ microcom --speed 115200 --port /dev/ttyUSB0
...
U-Boot 2016.07+fslc+ge6b4241 (Sep 14 2016 -
11:21:52 +0200)

CPU:   Freescale i.MX7S rev1.2 at 792MHz
...
Hit any key to stop autoboot:  0
```

- Montage de la partition eMMC côté bootloader :

```
=> ums 0 mmc 0
```

- Décompression et opération de copie coté système hôte sur le device spécifique :

```
$ cd tmp/deploy/images/imx7s-warp
$ gunzip qt-image-imx7s-
warp-20161118195314.rootfs.sdcard.gz
$ sudo dd if=qt-image-imx7s-
warp-20161118195314.rootfs.sdcard
of=/dev/sdX (où X représente le device)
```

Une fois la copie terminée, l'utilisateur pourra redémarrer la carte pour voir apparaître la séquence de démarrage :

```
Starting kernel ...

Booting Linux on physical CPU 0x0
Linux version 4.1.32-fslc+g9d3f7f9 (pjtexier@
bdxiot) (gcc version 5.3.0 (GCC) ) #1 SMP
PREEMPT
...
IOT powered by Yocto/OE (Welcome to Bordeaux)
2.1.1 iot /dev/ttymxc0

iot login: root
Password: iot
 ___ ___ _____
|_ _/ _ \_   _|
 | | | | || |
 | | |_| || |
|___\___/ |_|

root@iot:~#
```

1.4 Activation de l'interface Bluetooth

Rappelons ici, comment attacher, initialiser et activer notre interface Bluetooth, afin que celle-ci soit accessible par les autres périphériques compatibles :

```
root@iot:~# hciattach ttymxc2 bcm43xx 3000000
flow -t 20
root@iot:~# hciconfig hci0 up
root@iot:~# hciconfig hci0 name 'Hello GLMF200'
// Nom de l'interface visible depuis n'importe
quel périphérique bluetooth
root@iot:~# hciconfig hci0 piscan
```

1.5 SDK Qt5

1.5.1 Génération

Nous avons évoqué lors de l'article précédent la notion de SDK et chaîne de compilation croisée (mais pas que). Ceci afin de pouvoir générer du code Qt5 sur notre cible. La commande ci-dessous permettra la génération d'un SDK générique et compatible Qt5 :

```
$ bitbake meta-toolchain-qt5
```

Une fois la génération terminée, le script d'installation du SDK se trouvera dans **tmp/deploy/sdk**.

1.5.2 Installation

Elle se fera tout simplement en exécutant le script suivant où il conviendra de spécifier le chemin d'installation (**/opt/iot/sdk** dans cet exemple) :

```
$ cd tmp/deploy/sdk
$./iot-glibc-x86_64-meta-toolchain-qt5-
cortexa7hf-neon-toolchain-2.1.1.sh
IOT powered by Yocto/OE (Welcome to
Bordeaux) SDK installer version 2.1.1
==========================================
Enter target directory for SDK (default:
/opt/iot/2.1.1): /opt/iot/sdk
```

1.5.3 Intégration à l'IDE Qt creator

Afin de se voir l'environnement de compilation croisée, intégré à notre IDE préféré, il nous faudra :

- Dans ***Outils > Options > Compiler et Exécuter***
- Puis dans la section ***Compilateurs > Ajouter,*** choisir ***« GCC »***
- On donnera un nom à notre GCC spécifique (par exemple WaRP7_GLMF200),
- Puis nous spécifierons le chemin du compilateur : **`/opt/iot/sdk/sysroots/x86_64-pokysdk-linux/usr/bin/arm-poky-linux-gnueabiarm-poky-linux-gnueabi-gcc`**
- Dans ***Versions de Qt***, Qt est normalement automatiquement détecté. Sinon le lecteur devra ajouter le chemin vers qmake : **`/opt/iot/sdk/sysroots/x86_64-pokysdk-linux/usr/bin/qt5/qmake`**
- Dans ***Kits***, il ne reste plus qu'à choisir les éléments précédemment mis en place (Compilateur et version), il sera aussi recommandé de spécifier le mkspecs « **`linux-oe-g++`** » :

2. MIKROBUS

Dans cette partie nous aborderons, dans un premier temps, le standard MikroBUS, nous présenterons autant les aspects matériels que les aspects logiciels que propose celui-ci. Nous ferons ensuite un bref tour d'horizon sur les modules que propose MikroElektronika.

2.1 Le standard : introduction

Le standard MikroBUS définit les connectiques des cartes principales (on parle souvent de carte mère) ainsi que celles des cartes additionnelles (add-on) utilisées pour l'interfaçage du microcontrôleur ou du microprocesseur avec les circuits et modules complémentaires. De plus, il spécifie la disposition physique du brochage, les broches de communication et les broches d'alimentation utilisées.

On remarque, bien évidemment, que l'objectif de mikroBUS est de jouer sur la flexibilité matérielle, permettant ainsi un interfaçage plus facile avec un grand nombre de cartes complètes, qui plus est, standardisées (plus de 200 modèles commercialisés par MikroElektronika tout de même !), chacune avec un simple capteur (humidistance), un module radio (RFID), un écran (OLED), une connectique (RS232), ou tout autre module électronique **[7]**.

NOTE

À noter que mikroBUS est un standard ouvert et donc n'importe qui peut prétendre implémenter mikroBUS dans sa conception matérielle, à condition bien sûr de respecter les conditions de MikroElektronika.

2.2 Le standard : Description du connecteur MikroBUS

Le connecteur MikroBUS est composé de 16 broches (2x8). L'ensemble de celles-ci est toujours positionné de la même manière comme le montre la figure 2.

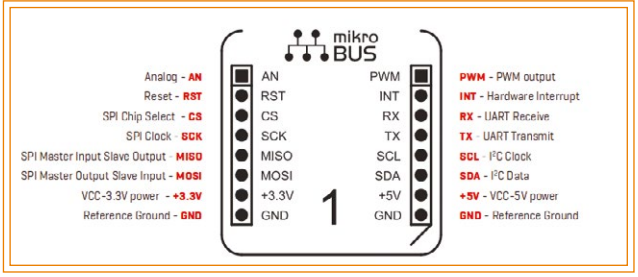


Fig. 2: pinout du standard MikroBUS

On retrouvera sur chaque module :

- Des broches par type de bus de communication : SPI, UART, I2C.
- 4 broches complémentaires : PWM (ou MLI en français), interruption matérielle (INT), entrée analogique (AN) et reset (RST).

Puis un couple pour les alimentations :

- 3,3V / GND
- 5V / GND

2.3 Quelques Examples

Il est bien évidemment compliqué de présenter ici l'ensemble des modules proposés par la firme MikroElektronika. Afin de donner au lecteur un bref aperçu des cartes disponibles, les auteurs auront choisi d'en exposer 2 au travers de cet article.

2.3.1 La « Relay Click Board »

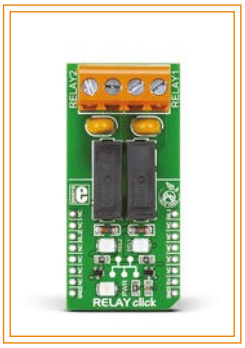


Fig. 3: La Relay Click sous les projecteurs

Cette carte embarque deux relais de puissance (G6D1AASI-5DC de référence Omron). Le pilotage vers la carte principale (dans notre cas la WaRP7) s'effectue via les broches suivantes du connecteur MikroBUS :

- Broche **PWM** pour le Relais 1 (RL1),
- Broche **CS** pour le Relais 2 (RL2).

De notre point de vue, ce ne sera ni plus ni moins que l'activation (état haut ou état bas) d'une broche GPIO (ouverture ou fermeture du relais).

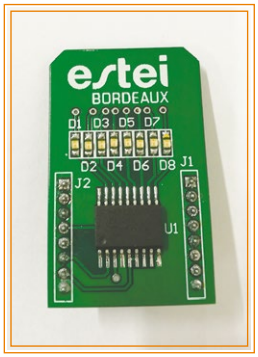


Fig. 4: L'inconnu au bataillon

2.3.2 La « custom »

Comme remarqué précédemment, il est possible de réaliser sa propre carte autour du standard MikroBUS. Dans cet exemple, le module est composé d'un gpio expander (microchip **mcp23s08**) pilotable au travers d'une interface SPI (mais aussi via le bus i2c). Cette option permet d'étendre les capacités de la carte principale quand celle-ci arrive à ses limites (ou si aucune GPIO physique n'est présente sur la carte mère).

3. JOUONS UN PEU AVEC LA « RELAY CLICK BOARD »

Viens maintenant le temps de jouer un peu avec notre module MikroBUS, celui-ci permettant d'étendre les capacités de notre WaRP7, nous mettant ainsi des relais à disposition (ceci afin de commander par exemple une lampe, ou tout autre objet ...). Dans ce chapitre nous verrons comment les piloter de façon simpliste en accédant aux GPIO depuis l'espace utilisateur. Nous aborderons ensuite le sujet du *device tree* et l'accès à la configuration des GPIO depuis celui-ci.

3.1 Plug !

Insérons notre module relais et constatons qu'il est très facile d'intégration entre notre plateforme cible et les modules MikroBUS (WaRP7 <-> Relay Click Board), ceci grâce à la standardisation :

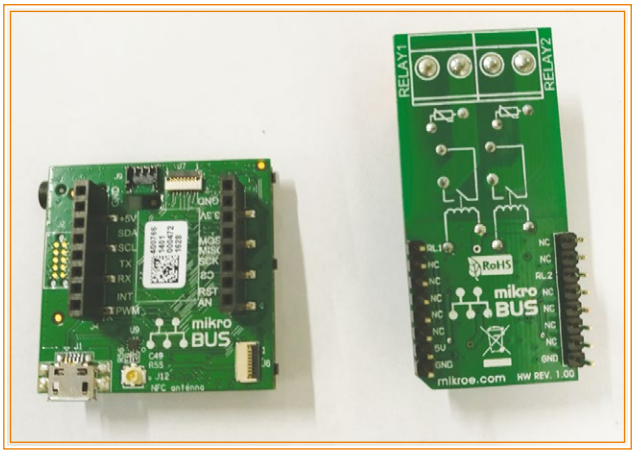


Fig. 5: La WaRP7 en position !

3.2 And Play !

Sous GNU/Linux, l'accès aux GPIO **[8]** s'effectue dans **/sys/class/gpio/gpioN** où N représente le numéro de la GPIO (à condition que le driver associé soit présent → **GPIO_SYSFS**).

Nous savons par le biais de la schématique (disponible en **[2]**), que le relais 1 et sur la broche CPU « GPIO7_IO8 », qui, en espace utilisateur devient **gpio200**.

NOTE

Comment calculer le numéro de gpio utilisé en mode userspace ?

Le calcul est sous la forme : **((port_GPIO - 1)*32) + n°gpio**.

Exemple pour le relais 1 : **((7-1)*32) + 8 = 200**

La première étape consistera à exporter cette GPIO pour la rendre accessible depuis le système de fichier virtuel **/sys**, la commande si après créera donc un point d'accès **gpio200** :

```
root@iot:~# echo 200 > /sys/class/gpio/export
```

L'interface étant disponible, il nous faut la paramétrer en tant que broche de sortie :

```
root@iot:~# echo out > /sys/class/gpio/
gpio200/direction
```

Une fois configurée, il nous est possible de la piloter, ceci en agissant simplement sur la tension de sortie de la broche, via la commande suivante :

```
root@iot:~# echo 1 > /sys/class/gpio/gpio200/
value
```

La broche de sortie passée à l'état haut, il nous est facile de constater le résultat sur la carte relais (led REL1), comme nous le montre la figure 6 :

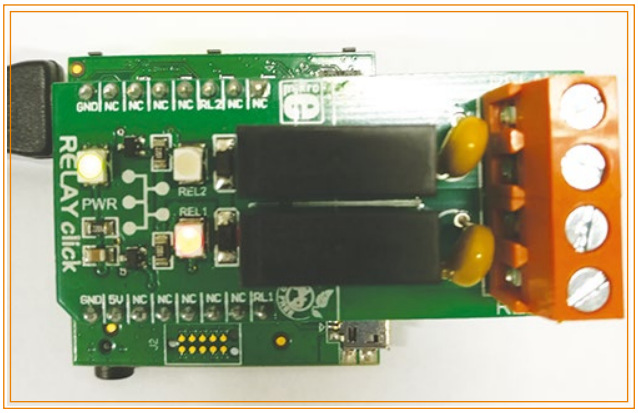


Fig. 6: Relais 1 en fonctionnement

Nous ferons de même pour le relais numéro 2 (RL2 sur la « relay click ») qui lui est sur « GPIO4_IO23 » :

```
root@iot:~# echo 119 > /sys/class/gpio/export
root@iot:~# echo out > /sys/class/gpio/gpio119/
direction
root@iot:~# echo 1 > /sys/class/gpio/gpio119/
value
```

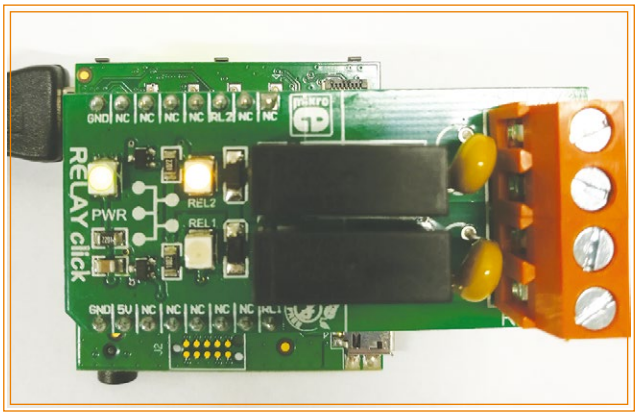


Fig. 7: Relais 2 en fonctionnement

3.2 Intégration au device-tree : « gpio-exporter & pin-muxing »

L'inconvénient des étapes (**export/direction**) précédentes réside dans le fait qu'il nous est obligatoire de les réexécuter à chaque démarrage de la WaRP7, car non statique au sein de la configuration matérielle. Il serait donc intéressant d'avoir une entrée statique. Nous irons un peu plus loin que la simple configuration des GPIO.

En effet, il serait aussi plaisant de pouvoir associer un nom à nos GPIO lors de la configuration de celles-ci, pour ce faire, nous intégrerons le travail de Martin FUZZEY **[9]** (qu'on ne manquera pas de remercier pour le travail effectué), qui a en effet développé un driver Kernel permettant d'avoir au sein de l'espace utilisateur, un lien nom/numéro de GPIO. L'avantage de cette solution est que finalement, l'utilisateur ne se soucie plus du numéro de la GPIO, de ce fait, la configuration est beaucoup plus portable et maintenable.

3.2.1 Device tree : une petite introduction

L'arbre de périphériques (*device tree* ou encore DT) est une structure de données permettant de décrire le matériel d'un système, dérivée du format utilisé par Open Firmware pour encapsuler les informations de plateforme et les transmettre au noyau, qui utilise alors les données du DT pour trouver et enregistrer les périphériques du système.

La structure de données elle-même est un arbre de nœuds nommés et de propriétés. Chaque nœud contient des propriétés de simples paires « nom-valeur » et des nœuds fils. Afin d'être interprétée correctement par le noyau, l'arborescence doit suivre une structure prédéfinie. Une « liaison » (en anglais, *binding*) est une description de la façon dont un périphérique est décrit dans le DT. Un grand nombre de périphériques disposent de liaisons bien établies et documentées.

3.2.2 Anatomie de notre fichier dts « imx7s-warp-relay.dts »

Afin de ne pas surcharger le fichier dts (*device tree source*) principal (**imx7s-warp.dts**), les auteurs auront choisi de créer un fichier dts spécifique à l'intégration du module « relay click », en voici sa constitution :

```
#include "imx7s-warp.dts"

/ {
 gpio_exporter: gpio-exporter {
  compatible = "linux,gpio-exporter";
  pinctrl-names = "default";
  pinctrl-0 = <&pinctrl_gpioexporter_relay>;

  out_RL1 {
   gpios = <&gpio7 8 GPIO_ACTIVE_HIGH>;
   output;
   initial-state = <0>;
  };

  out_RL2 {
   gpios = <&gpio4 23 GPIO_ACTIVE_HIGH>;
   output;
   initial-state = <0>;
  };
```

```
 };
};

&iomuxc {
 pinctrl-names = "default";
 imx7s-warp {

  pinctrl_gpioexporter_relay: gpioexportergrp {
   fsl,pins = <
    MX7D_PAD_ENET1_RGMII_TD2__GPIO7_IO8
0x14
    MX7D_PAD_ECSPI2_SS0__GPIO4_IO23     0x14
   >;
  };
 };
};
```

Essayons de détailler celui-ci :

- **#include "imx7s-warp.dts"** fera référence au fichier principal la WaRP7 (définition de la carte), qui lui-même inclus la définition du SoC (System on Chip) i.MX7 (**imx7d.dtsi**).
- Au niveau du contenu du fichier :
- La création d'un nœud (**node**) avec son label associé, que l'on appellera gpio-exporter :

```
gpio_exporter: gpio-exporter {
   compatible = "linux,gpio-exporter";
```

- Le mot clé **compatible** est une propriété qui permettra de faire le lien avec la partie driver (respectant la règle du **plateform driver** comme présenté par Pierre Ficheux en **[10]**). Ceci via la structure **of_device_id** qui sera utilisée avec le tableau **gpio_exporter_dt_ids[]**. Le lecteur curieux pourra jeter un coup d'oeil au code source dans **tmp/work/imx7s_warp-poky-linux-gnueabi/linux-fslc-imx/4.1-1.0.x+gitAUTOINC+9d3f7f9343-r0/git/drivers/gpio/gpio-exporter.c** :

```
static const struct of_device_id gpio_
exporter_dt_ids[] = {
 { .compatible = "linux,gpio-exporter", },
} ;
```

- La propriété **pinctrl-0** permet de donner la liste des broches qui seront soumises à une définition spécifique de leur état, ceci grâce au sous-système **pinctrl** qui permet de gérer le multiplexage des broches. Dans notre cas, la configuration s'effectuera dans le nœud fils **pinctrl_gpioexporter_relay** :

```
   pinctrl-0 = <&pinctrl_gpioexporter_relay>;
```

- Il convient ensuite de créer un nœud fils (**child node**), le nom de celui-ci représentera le nom exposé au sein de l'espace utilisateur, nous garderons la même syntaxe que la carte « **relay click** », à savoir **RL1** pour le pilotage du relais 1 :

```
   out_RL1 {
    gpios = <&gpio7 8 GPIO_ACTIVE_HIGH>;
    output;
    initial-state = <0>;
   };
```

Il nous faudra aussi renseigner les propriétés à notre nœud fils :

- **gpios** : Référence vers la GPIO à exporter (binding standard défini dans les sources du Noyau Linux : **Documentation/devicetree/bindings/gpio/gpio.txt**). En plus du *phandle* (nœud vers le contrôleur associé), nous retrouverons le numéro de GPIO, 8 pour notre application. Puis le second argument signifie que celle-ci est active sur un niveau haut (**GPIO_ACTIVE_HIGH**).
- **output** : pour spécifier que l'on désire la positionner en sortie,
- **initial-state** : où on fixera sont état initial (état bas dans notre cas)

Il en sera de même pour le relais numéro deux, où seul le nom du sous-nœud et de la GPIO associé seront donc à modifier pour le piloter :

```
   out_RL2 {
    gpios = <&gpio4 23 GPIO_ACTIVE_HIGH>;
    output;
    initial-state = <0>;
   };
};
```

Nous en resterons là quant à la présentation du fichier *device tree*, le lecteur pourra se référer au très bon article écrit par Thomas Petazzoni paru dans le n°17 d'Open Silicium **[11]**.

3.3 Activation du driver Kernel

La partie *device tree* étant faite, il nous reste à activer le support du driver **gpio-exporter** au sein de notre image Kernel (zImage). Pour ce faire, nous passerons une fois de plus par l'environnement Yocto/OE, en invoquant la commande suivante :

```
$ bitbake linux-fslc-imx -c menuconfig
```

Puis dans ***Devices Drivers > GPIO Support*** il nous faudra activer l'option comme le montre la figure 8.

Nous pourrons dès à présent relancer une phase de compilation pour générer le fichier zImage et notre nouveau fichier

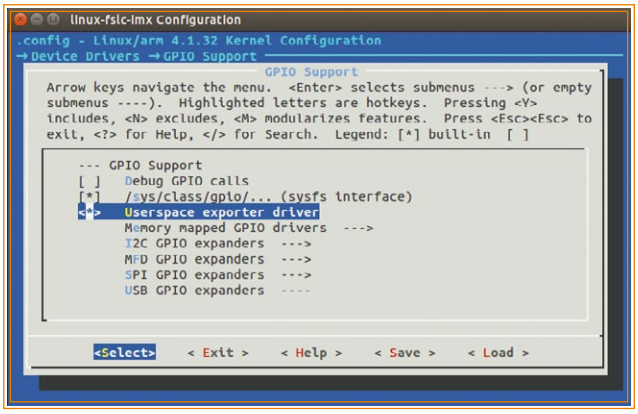


Fig. 8: CONFIG_GPIO_EXPORTER au sein de menuconfig

NOTE

L'ensemble des patchs relatifs au kernel (gpio-exporter, dts et fichier defconfig), sont directement intégrés au sein de la couche distribution mis à disposition par les auteurs : https://github.com/bdx-iot/meta-iot/tree/master/recipes-kernel (au travers une recette dérivée)

dtb (transformation en un fichier binaire du fichier dts), qui sera donc **imx7s-warp-relay.dtb** :

```
$ bitbake linux-fslc-imx
```

3.4 Intégration

Nous allons, dans cette sous-partie, mettre à jour le fichier **zImage** ainsi que le fichier **imx7s-warp-relay.dtb** sur notre cible. Pour ce faire, il sera question, ici aussi, de monter la partition eMMC en mode mass storage via u-boot. Nous finirons naturellement cette sous-partie par une phase de test sur cible.

3.4.1 zImage et .dtb

Pour la copie de la zImage :

```
$ sudo cp tmp/work/imx7s_warp-poky-
linux-gnueabi/linux-fslc-imx/4.1-
1.0.x+gitAUTOINC+9d3f7f9343-r0/build/arch/arm/
boot/zImage /media/<username>/Boot imx7s-/
```

Pour la copie du fichier dtb spécifique à la gestion du relais :

```
$ sudo cp tmp/work/imx7s_warp-poky-
linux-gnueabi/linux-fslc-imx/4.1-
1.0.x+gitAUTOINC+9d3f7f9343-r0/build/arch/arm/
boot/dts/imx7s-warp-dtb /media/<username>/Boot
imx7s-/
```

Plaçons-nous maintenant côté cible et spécifions à notre chargeur d'amorçage (u-boot dans notre cas) que nous souhaitons utiliser le nouveau fichier dtb fraichement généré (imx7s-warp-relay.dtb) intégrant la partie **gpio-exporter** pour la gestion de nos relais, mais nous souhaitons, dans une moindre mesure, que celui-ci soit pris en compte par défaut, ceci afin d'éviter l'obligation d'une connexion série à chaque démarrage de la cible, il nous suffit donc de faire :

```
=> setenv fdt_file imx7s-warp-relay.dtb
```

Ceci surcharge la variable **fdt_file**, variable qui contient la référence vers le fichier dtb à utiliser (de base, imx7s-warp.dtb dans les sources de u-boot). Sauvegardons maintenant notre environnement :

```
=> save
Saving Environment to MMC...
Writing to MMC(0)... done
```

Il nous est maintenant possible de démarrer notre cible via l'exécution de la commande **bootcmd** (qui contient une suite de commandes à exécuter au démarrage) :

```
=> run bootcmd
switch to partitions #0, OK
...
6390056 bytes read in 94 ms (64.8 MiB/s)
Booting from mmc ...
reading imx7s-warp-relay.dtb
37834 bytes read in 12 ms (3 MiB/s)
Kernel image @ 0x80800000 [ 0x000000 - 0x618128 ]
## Flattened Device Tree blob at 83000000
...
Starting kernel ...
```

3.4.2 Test en espace utilisateur

Afin de simplifier (encore plus) l'accès aux GPIO (via **gpio-exporter**), les auteurs auront préféré mettre en place la notion de lien symbolique afin de se créer un point d'accès dans **/dev/mikrobus/***. En temps normal l'accès aux GPIO s'effectue dans **/sys/devices/platform/gpio-exporter//out_RL***. Cette technique nous donnera donc le résultat suivant sur notre plateforme :

```
root@iot:~# ls -l /dev/mikrobus/out_RL*
lrwxrwxrwx    1 root     root
50 Nov 18 10:26 /dev/mikrobus/out_RL1
-> /sys/devices/platform/gpio-exporter//
out_RL1/value
lrwxrwxrwx    1 root     root
50 Nov 18 10:26 /dev/mikrobus/out_RL2
-> /sys/devices/platform/gpio-exporter//
out_RL2/value
```

NOTE

Liens Symboliques

Le script concernant la génération des liens symboliques est entièrement consultable sur le github des auteurs : https://github.com/bdx-iot/meta-iot/blob/master/recipes-iot/clicks-board-init/clicks-board-init/mikrobus.sh.

Tout comme en section précédente, nous pouvons de façon très simple, et ceci sans connaître le numéro de la GPIO, piloter nos relais (plutôt pas mal non ?) :

```
root@iot:~# echo 1 > /dev/mikrobus/out_RL1 // Pour activer le relais
root@iot:~# echo 0 > /dev/mikrobus/out_RL2  // Pour désactiver le relais
```

4. MISE EN SITUATION ...

Nous clôturerons cet article sur un mini-projet (par simple habitude des auteurs) permettant ainsi au lecteur de mettre en pratique les notions évoquées tout au long de l'article. Celui-ci nous permettra d'aborder quelques nouvelles techniques, comme par exemple l'introduction au QML.

4.1 Présentation du projet

Nous nous servirons du travail effectué durant cette première partie de l' article, qui de ce fait, nous servira de base pour notre application « connectée ». Nous proposons ici de faire une évolution du serveur BLE du précédent article.

L'idée sera donc de créer :

- une application « serveur BLE », permettant le pilotage de notre carte « relay click », application qui sera sur la WaRP7.
- un client associé, qui, lui, sera sous forme d'application Android et qui enverra les données au serveur BLE (commande des relais).

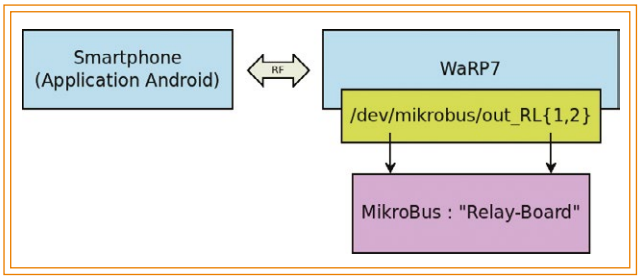


Fig. 9: structure du projet

La figure 9 illustre l'architecture globale que l'on se propose de mettre en œuvre durant cette dernière partie de l'article.

4.2 Le Serveur

Interaction avec nos GPIO

La première étape consistera à créer une fonction nous permettant d'écrire sur les GPIO. Il suffit d'utiliser la classe **QFile** **[12]** qui permet la gestion de fichiers (et oui, sous Linux tout est fichier) :

```
m_relay1File = new QFile("/dev/mikrobus/out_RL1");
m_relay2File = new QFile("/dev/mikrobus/out_RL2");
```

Ceci étant fait, il ne reste plus qu'à ouvrir (via l'appel **open()**) le fichier (avec la permission d'écriture !), écrire la valeur souhaitée (**0** ou **1**) puis le refermer via l'appel **close()** :

```
m_relay1File->open(QIODevice::ReadWrite);
m_relay1File->write(data);
m_relay1File->close();
```

Dans le cas présent, « **data** » est de type **QByteArray** (simple tableau d'octets) contenant la valeur à envoyer sur notre GPIO. Nous verrons plus tard comment cette dernière est envoyée.

Et la partie BLE ?

Dans cet exemple, nous avons décidé de ne pas utiliser les services **[13]**/caractéristiques **[14]** standards proposés par le protocole Bluetooth, et de ce fait, nous avons créé les nôtres pour les besoins de cet article, c'est bien plus ludique quand même !

La figure 10 indique comment nous allons architecturer nos données.

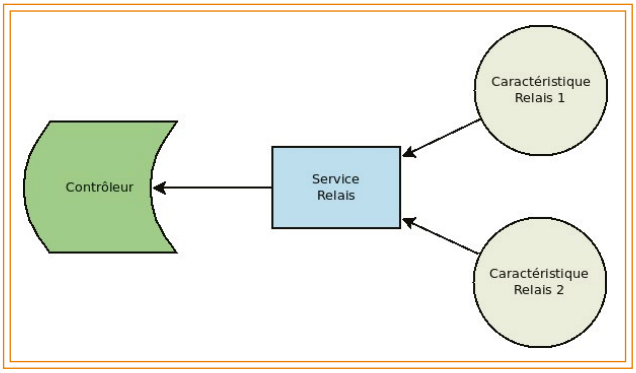


Fig. 10: Architecture du serveur BLE

Il faut donc d'abord avoir nos propres **UUID** (Service + Caractéristiques) :

```
#define SERVICE_UUID            0x1820
#define CHAR_UUID_RELAY1        0x2aa4
#define CHAR_UUID_RELAY2        0x2aa5
```

Que nous allons ensuite « caster » en **QBluetoothUuid** :

```
QBluetoothUuid serviceRelayUuid((quint32)
SERVICE_UUID);
QBluetoothUuid charRelay1Uuid((quint32)
CHAR_UUID_RELAY1);
QBluetoothUuid charRelay2Uuid((quint32)
CHAR_UUID_RELAY2);
```

Les présentations étant faites, nous pouvons rentrer dans le vif du sujet, la création du serveur. Et bien, pour ce faire, rien de plus simple avec l'API proposé par Qt :

Nous déclarons dans un premier temps un « **advertiser** » qui définit la configuration de base de notre serveur, ceci grâce à la classe suivante :

```
QLowEnergyAdvertisingData   m_advertisingData;
```

Configurons son mode, son nom (GLMF_200_BLE) et nous y ajoutons un nouveau service (**serviceRelay** dans notre exemple) :

```
//Mode d'accessibilité
m_advertisingData.setDiscoverability(
                QLowEnergyAdvertisingData::D
iscoverabilityGeneral);
//Nom du serveur
m_advertisingData.setLocalName("GLMF_200_
BLE");
//Ajout du service Relais
m_advertisingData.setServices(QList<QBluetooth
Uuid>() <<
                              serviceRelayUuid
                                );
```

Nous créons ensuite les caractéristiques voulues (**charRelay1**). Le descripteur va définir la gamme de fonctionnalités de la caractéristique associée.

```
QLowEnergyCharacteristicData charRelay1;
charRelay1.setUuid(charRelay1Uuid); //uuid
définie de la caractéristique
charRelay1.setValue(QByteArray(2, 0));
charRelay1.setProperties(type); //précise le
type de la propriété
const QLowEnergyDescriptorData clientConfig(
//Descripteur standard
                        QBluetoothUuid::Client
CharacteristicConfiguration,
                        QByteArray(2, 0));
charRelay1.addDescriptor(clientConfig);
```

Puis nous les associons au service préalablement configuré. Nous définissons son type comme « Primaire » car il donne accès aux fonctionnalités standard des services :

```
//Couplage du service avec la caractéristique créée.
QLowEnergyServiceData serviceRelayData;
serviceRelayData.setType(QLowEnergyServiceData::
ServiceTypePrimary);
serviceRelayData.setUuid(serviceRelayUuid);
serviceRelayData.addCharacteristic(charRelay1);
//Ajout relai 1
serviceRelayData.addCharacteristic(charRelay2);
//Ajout relai 2
```

La dernière étape arrive. Il ne reste plus qu'à créer le contrôleur, qui est, entre autres, le point d'entrée des périphériques Bluetooth. On ajoute alors le service relais à celui-ci et on démarre le serveur. Ainsi on rend la connexion possible :

```
//création du controlleur BLE
m_bleController = QLowEnergyController::
createPeripheral();
//Permettra de savoir si un client se déconnecte afin
de relancer l'"advertising"
connect(m_bleController,
  SIGNAL(stateChanged(QLowEnergyController::
ControllerState)),
  this, SLOT(controllerStateChanged
(QLowEnergyController::ControllerState)));

//ajout du service
m_serviceRelay = m_bleController-
>addService(serviceRelayData);
//permettra de récupérer les données reçues
connect(m_serviceRelay,
  SIGNAL(characteristicChanged
(QLowEnergyCharacteristic,QByteArray)),
  this, SLOT(characteristicChanged(
QLowEnergyCharacteristic,QByteArray)));

//démarrage "advertising"
m_bleController->startAdvertising(
QLowEnergyAdvertisingParameters(),m _advertisingData,
m_advertisingData);
```

Il ne reste donc plus qu'à implémenter la fonction qui traite la réception d'une donnée afin de savoir à quel relais elle s'applique et dans quel état nous devons le mettre. Il suffit assez simplement de lire l'**UUID** de la caractéristique reçue et de commuter celui concerné en fonction :

```
void CServerBLE::characteristicChanged(
QLowEnergyCharacteristic c, QByteArray data)
{
    switch(c.uuid().toUInt32())
    {
    case CHAR_UUID_RELAY1:
        m_relay1File->
open(QIODevice::ReadWrite);
```

```
        m_relay1File->write(data);
        m_relay1File->close();
        break;
    case CHAR_UUID_RELAY2:
        m_relay2File->
open(QIODevice::ReadWrite);
        m_relay2File->write(data);
        m_relay2File->close();
        break;
    }
}
```

Le travail est fini et le serveur est ainsi prêt à fonctionner. Si le lecteur veut s'y tenter, il pourra d'ores et déjà compiler l'application pour la déployer sur la cible (**`bitbake ServiceRelay`**). Pour de plus amples informations sur la gestion des paquets/déploiement, les auteurs renverront le lecteur sur un excellent article de Pierre Ficheux **[15]**).

NOTE

Tempétaure

Dans l'exemple complet proposé sur le Github des auteurs, nous avons rajouté la lecture de la température présente sur la Warp7 afin de donner un peu de vie à l'interface.

4.3 « Dans la peau du client »

Création de notre environnement

Le serveur est fonctionnel, donc prêt à être piloté. Il ne manque plus qu'à lui associer un client. L'idée ici est de proposer une application Android avec le *framework* Qt en QML.

NOTE

QML

QML (pour *Qt Meta Language*) est un langage de programmation graphique qui ressemble beaucoup au JSON. Il a été créé principalement pour les applications mobiles, son utilisation étant bien plus simple que le Designer standard pour plateforme mobile.

Avant de pouvoir démarrer le projet, il faut mettre en place notre environnement. Pour cela, il est suggéré ici, de suivre le tutoriel Qt pour Android qui est très bien documenté (http://doc.qt.io/qt-5/android-support.html), qui consiste en l'installation de la chaîne de compilation Qt pour Android ainsi que le NDK et le SDK fournis par Google. Il faudra bien entendu installer Java !

Afin de ne pas sacrifier de nombreuses pages, les auteurs renverront le lecteur sur le GitHub en ce qui concerne l'ensemble des sources à télécharger. A la suite de cela, nous pouvons alors ouvrir Qt Creator et :

- Dans ***Tool > Options > Android***, il faudra ici remplir correctement les arborescences (JDK, NDK et SDK) comme le montre la figure 11.

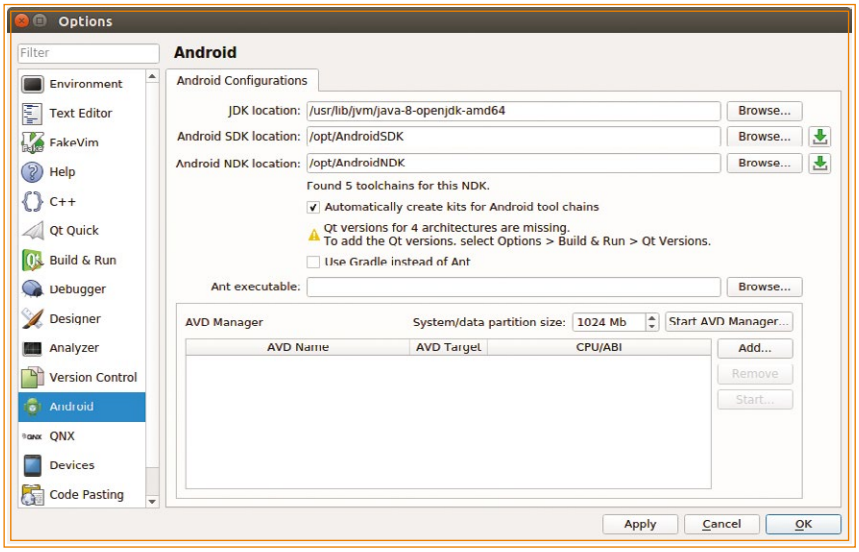


Fig. 11: Configuration de notre environnement

- Puis dans ***Tool > Options > Build & Run > Kits***, il faudra créer le Kit associé (voir figure 12).

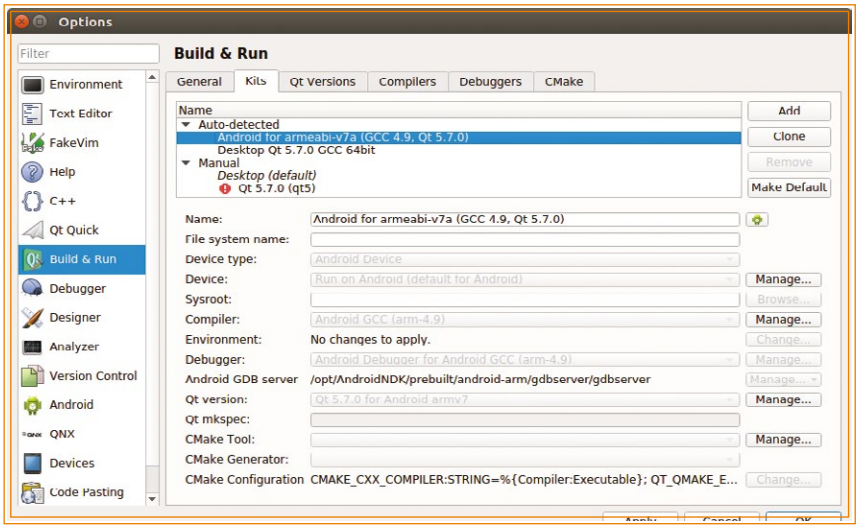


Fig. 12: Configuration de notre Kit

Un premier projet

En premier lieu, créons notre premier projet QML (Qt Quick Application) tout en sélectionnant le Kit Android pour le futur déploiement (smartphone).

Une application QML se scinde toujours en deux parties :

- une partie C++
- une partie script QML.

Il existe donc plusieurs outils pour communiquer entre les deux couches que nous expliciterons au cours de cette partie.

La partie C++

Dans cette partie, nous décrirons le client Bluetooth Low Energy. Et surtout, comment envoyer une donnée au serveur.

Dans un souci de simplicité, nous avons retiré les méthodes de découverte des périphériques présents. Partons du postulat que nous connaissons déjà l'adresse (MAC) de notre serveur. Nous chercherons simplement à nous y connecter de façon directe.

NOTE

Identifier l'adresse mac de l'interface BLE

Pour connaître l'adresse physique de l'interface bluetooth de notre WaRP7, il suffira d'exécuter la commande suivante :

```
root@iot:~# hciconfig
hci0:  Type: BR/EDR  Bus: UART
 BD Address: 43:43:A1:12:1F:AC  ACL MTU:
1021:8  SCO MTU: 64:1
 UP RUNNING PSCAN ISCAN
 RX bytes:766 acl:0 sco:0 events:52
errors:0
 TX bytes:1766 acl:0 sco:0 commands:52
errors:0
```

Voici donc l'introduction des premières lignes de code. Le type d'adresse peut être soit « Public » soit « Random ». Le premier définit un mode de fonctionnement standard avec une adresse « statique » et le second définit un mode dans lequel l'adressage peut évoluer, c'est une sécurité dont nous n'avons pas besoin pour la suite :

```
//création du contrôleur avec l'addresse du
serveur en paramètre
m_controller =  new QLowEnergyController(QBlue
toothAddress(address), this);
//on crée les connections
//Pour récupérer les services présents
connect(m_controller,
            SIGNAL(serviceDiscovered(QBluetoothUuid)),
            this,
            SLOT(addLowEnergyService(QBluetoothUuid)));
//Pour savoir quand la connection est faite
connect(m_controller,
            SIGNAL(connected()),
            this,
            SLOT(deviceConnected()));
//on se connecte
m_controller->setRemoteAddressType(QLowEnergyController
::PublicAddress);
m_controller->connectToDevice();
```

La connexion n'est pas instantanée, mais une fois faite, on lance alors la découverte des services associés au périphérique :

```
m_controller->discoverServices();
```

Puis, à la réception des services, il est possible de lancer la découverte des caractéristiques associées au service via la fonction « **discoverDetails()** » :

```
//on stock l'addresse du pointeur
m_connectedService = service;
if (m_connectedService->state() == QLowEnergyService::
DiscoveryRequired)
{
//permet de récupérer une mise à jour des
caractéristiques venant du serveur
connect(m_connectedService,
  SIGNAL(characteristicChanged(QLowEnergyCharacteristic,
QByteArray)),
  this,
  SLOT(serviceCharacteristicChanged(QLowEnergy
Characteristic,QByteArray)));

//Permet de découvrir les caractéristiques associées au
service
connect(m_connectedService,
  SIGNAL(stateChanged(QLowEnergyService::ServiceState)),
  this,
  SLOT(serviceDetailsDiscovered(QLowEnergyService::
ServiceState)));
//démarrer la découverte des caractéristiques
m_connectedService->discoverDetails();
}
```

Et enfin, à la réception des caractéristiques, on conserve la valeur contenue dans celles-ci et on met à jour l'interface en fonction. Les caractéristiques se reconnaîtront grâce à leurs **UUID** :

```
//on récupère le service
    QLowEnergyService *service = qobject_
cast<QLowEnergyService *>(sender());
    //on récupère ses caractéristiques
    const QList<QLowEnergyCharacteristic> chars =
service->characteristics();
    //on les parcours pour les traiter
    foreach (const QLowEnergyCharacteristic &ch, chars)
    {
         //relay1, on met à jour relay1
        if (ch.uuid().toUInt32() == CHAR_UUID_RELAY1)
        {
            m_relay1Characteristic = ch;
            m_relay1Changed = ch.value().toInt();
            Q_EMIT relay1Changed();
        }
        //Si uuid relay2, on met à jour relay2
        if (ch.uuid().toUInt32() == CHAR_UUID_RELAY2)
        {
            m_relay2Characteristic = ch;
            m_relay2Changed = ch.value().toInt();
            Q_EMIT relay2Changed();
        }
    }
```

Le plus dur est fait et, afin d'envoyer une nouvelle valeur au serveur en vue de changer l'état d'un des deux relais, il suffira d'utiliser la fonction suivante :

```
void CBLEDiscover::switchRelay1(bool value)
{
    m_connectedService->writeCharacteristic(
                m_relay1Characteristic,
                QByteArray::number(value));
}
```

A partir de là, il ne reste plus qu'à créer notre interface !

La partie QML

Lorsque l'on crée un fichier QML, on peut soit le modifier sous la forme d'un script, soit via le designer QML de Qt. Il existe donc deux types de fichiers qml :

- Fichier.ui.qml qui pourra être édité via l'éditeur et/ou par script,
- Fichier.qml qui pourra être modifié uniquement par script

Généralement, les **fichiers.ui.qml** sont associés à un **fichier.qml** afin de créer les connexions vers le C++. Dans l'exemple, nous avons créé deux fichiers **.qml** (associés chacun à un fichier.ui.qml) :

- main.qml : qui consiste en un simple bouton permettant d'enclencher la connexion directement avec le serveur dont l'adresse est celle spécifiée dans le code principal.
- Relay.qml : permet de piloter les deux relais.

Pour que le QML puisse communiquer avec notre classe, il faut pour ce faire, passer celle-ci en paramètre. Nous pouvons alors charger la page main.qml :

```
QQmlApplicationEngine engine;
//Charger la classe
CBLEDiscover *bleDiscover = new CBLEDiscover();
engine.rootContext()->setContextProperty("BLED
iscover", bleDiscover);
//Charger le QML
engine.load(QUrl(QStringLiteral("qrc:/main.
qml")));
```

Au chargement de l'application, nous aurons donc la page main.qml qui s'affichera constituée d'un simple bouton. Pour gérer l'évènement du clique, il suffit d'utiliser le script suivant :

```
connectButton.onClicked: {
            BLEDiscover.start();
            pageLoader.source = "Relay.qml"
        }
```

Afin que la fonction start soit appelée, il est nécessaire qu'elle soit invocable **[16]** depuis le C++ :

```
Q_INVOKABLE void start();
```

L'objet **pageLoader** permet de charger une nouvelle page QML et est défini par :

```
Loader { id: pageLoader }
```

La page Relay.qml est alors chargée. Elle est composée de deux switchs qui permettront de piloter les deux relais. De la même manière, le changement d'état des switchs se gère de la façon suivante :

```
relay1.onClicked:{
        if (relay1.checked == true)
            BLEDiscover.switchRelay1(true)
        else
            BLEDiscover.switchRelay1(false)
    }
```

L'application est bientôt terminée. Nous avons vu comment insérer et appeler une fonction depuis le QML vers le C++. Il ne reste plus qu'à expliquer comment mettre à jour la parité QML avec le C++. Pour réaliser le lien, Qt propose l'utilisation des **Q_PROPERTY** **[17]** qui seront liées à des signaux de mise à jour. Par exemple, si nous voulons mettre à jour l'état d'un relais nous définirons la Q_PROPERTY suivante :

```
//Q_PROPERTY
Q_PROPERTY(bool relay1State MEMBER m_
relay1Changed NOTIFY relay1Changed)
//valeur membre
bool m_relay1Changed;
//signal à émettre lors d'une mise à jour
Q_SIGNAL void relay1Changed();
```

Et côté QML, la connexion au signal se fait de la manière suivante :

```
relay1 {
        checked: BLEDiscover.relay1State;
        font.pointSize:  7;
    }
```

4.4 Le rendu final

Pour avoir un aperçu du projet réalisé, voici ci-après le fruit du travail réalisé durant cet article. La première étape consiste au lancement de l'application (**.apk** disponible ici : https://github.com/bdx-iot/RelayGUI/blob/master/GLMF_200_GUI_BLE.apk)

L'étape suivante nous indique de nous connecter à notre périphérique (figure 14).

Fig. 13: Démarrage de l'application

Fig. 14: Connexion

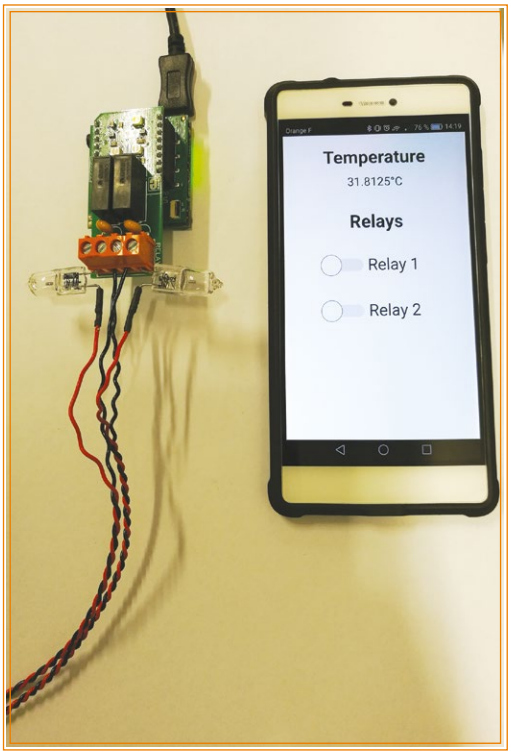


Fig. 15: IoT : L'environnement de test

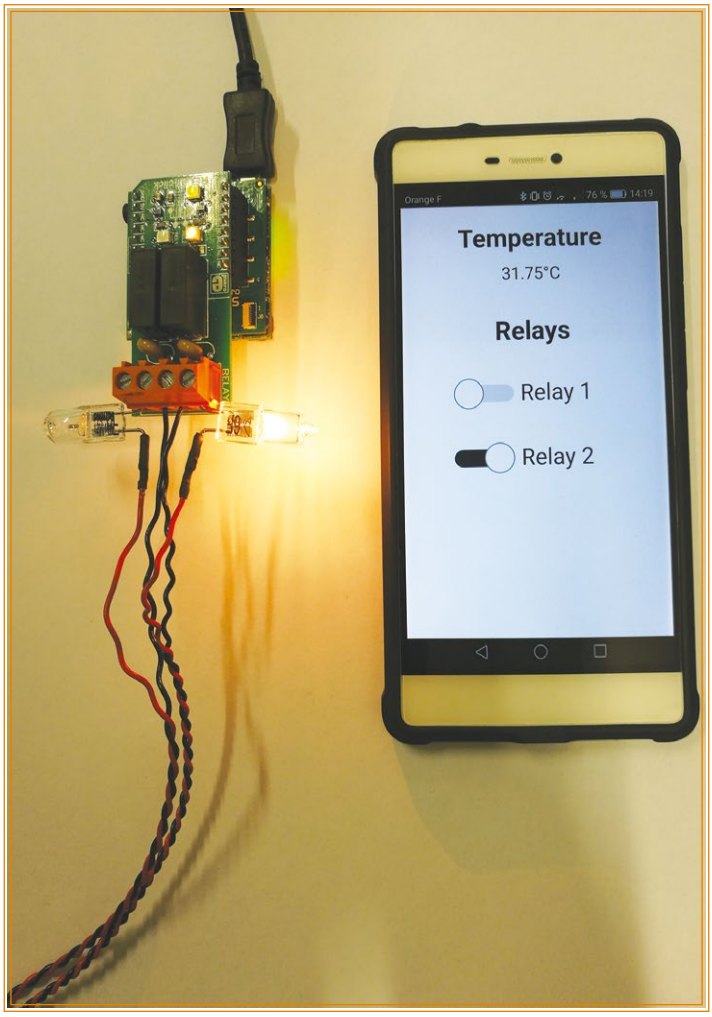


Fig. 16: IoT : Rendu globale avec RL1 activé

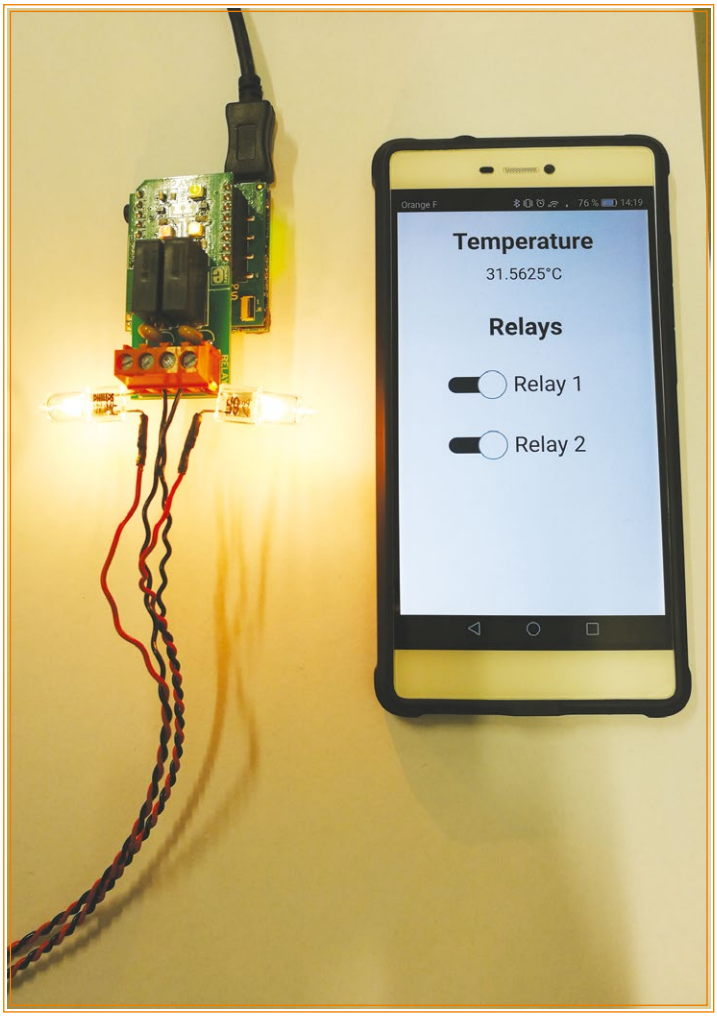


Fig. 17: IoT : Rendu globale

Une fois connectés, nous avons la satisfaction de pouvoir interagir avec nos relais, lire la température du capteur Altimètre (MPL3115), mais surtout, de pouvoir jouer avec une simple ampoule comme le montre notre environnement sur la figure 15, page précédente.

Agissons maintenant sur le relais 1 (figure 16).

Puis activons le 2ème relais (figure 17).

Rappelons une fois de plus que le lecteur pourra retrouver l'ensemble des sources sur le référentiel git suivant : https://github.com/bdx-iot. Se trouvera sur celui-ci :

- La couche « meta-iot » mise à disposition par les auteurs, comprenant :
- La recette dérivée du Kernel (gestion du fichier dts et fichier **defconfig**),
- La recette permettant la gestion des liens symboliques : https://github.com/bdx-iot/meta-iot/tree/master/recipes-iot/clicks-board-init
- La recette du « serveur BLE » et son script init associé (pour un démarrage automatique de l'application) : https://github.com/bdx-iot/meta-iot/tree/master/recipes-iot/service-relay
- Ainsi que d'autres recettes.
- Les sources de l'application serveur : https://github.com/bdx-iot/ServiceRelay
- Les sources de l'application client : https://github.com/bdx-iot/RelayGUI

CONCLUSION

Dans cet article, nous avons découvert le standard MikroBUS et nous avons ainsi pu le mettre en application sur notre plateforme cible. De par l'intégration, nous avons d'autre part, découvert des notions comme le device tree ou encore le langage QML.

Finalement, nous avons mis en pratique l'ensemble des notions pour créer un véritable objet connecté (jusqu'à la création de la partie client pour la gestion de celui-ci), le tout basé sur du Bluetooth Low Energy. Ceci n'étant qu'une introduction au vu du potentiel, le lecteur pourra s'inspirer et imaginer d'autres applications (domotique) via les modules Mikrobus de MikroElektronika.

En parlant de domotique, n'oublions pas que cette plateforme dispose d'une interface caméra, autre périphérique qu'il serait intéressant de mettre en œuvre afin de créer un environnement connecté pour la surveillance de nos enfants (vidéo/température/gestion de la veilleuse), ceci en ne négligeant pas l'aspect low energy de notre objet bien entendu. ■

LIENS

[1] Article « A la découverte de la WaRP7 » par Pierre-Jean TEXIER et Jean CHABRERIE, *Open Silicium n°20*

[2] Site d'Element14 : https://www.element14.com/community/docs/DOC-79058/l/warp7-the-next-generation-iot-and-wearable-development-platform

[3] Page NXP présentant l'architecture du i.MX7 : http://www.nxp.com/products/microcontrollers-and-processors/arm-processors/i.mx-applications-processors/i.mx-7-processors/i.mx-7solo-processors-heterogeneous-processing-with-arm-cortex-a7-and-cortex-m4-cores:i.MX7S

[4] Documentatio du projet Yocto (version 2.1 : krogoth) : http://www.yoctoproject.org/docs/2.1/mega-manual/mega-manual.html

[5] Documentation relative à l'outil repo : https://source.android.com/source/using-repo.html

[6] Couche distribution des auteurs : https://github.com/bdx-iot/meta-iot

[7] Ensemble des cartes « click » par MikroElektronika : http://www.mikroe.com/click/

[8] Documentation dans les sources du Noyau Linux sur l'utilisation des GPIO en espace utilisateur : https://www.kernel.org/doc/Documentation/gpio/sysfs.txt

[9] Driver gpio-exporter : https://patchwork.kernel.org/patch/6207321/

[10] Article « Périphériques découvrables, découverts ou à découvrir » par Pierre FICHEUX, *Open Silicium n°20*

[11] Article « Introduction au « device Tree »sur ARM » par Thomas PETAZZONI, *Open Silicium n°17*

[12] Documentation de la classe Qfile : http://doc.qt.io/qt-5/qfile.htmlhttp://doc.qt.io/qt-5/qfile.html

[13] Spécifications des services bluetooth : https://www.bluetooth.com/specifications/gatt/services

[14] Spécifications des caractéristiques bluetooth : https://www.bluetooth.com/specifications/gatt/characteristics

[15] Article « IoT sous Yocto, un cas d'utilisation » par Pierre FICHEUX, *Open Silicium n°20*

[16] Explication de la macro Q_INVOKABLE : http://doc.qt.io/qt-5/qobject.html#Q_INVOKABLE

[17] Explication de la macro Q_PROPERTY : http://doc.qt.io/qt-5/qobject.html#Q_PROPERTY

PROGRAMMATION D'APPLICATION
AUDIO EN C++ AVEC JUCE

ADRIEN ANSELME
[Développeur audio chez Springbeats.com]

MOTS-CLÉS : C++, JUCE, PROGRAMMATION AUDIO, INTERFACE GRAPHIQUE, MAO, BIT CRUSHER

Créer une application autonome pour traiter le son peut paraître long et fastidieux. Avec l'aide de la bibliothèque JUCE, c'est en vérité très simple. Pour le prouver, nous allons créer ensemble un « bit crusher » : un programme qui réduit la taille des échantillons sonores d'un fichier pour lui donner un effet de distorsion.

Si le monde de l'audio a longtemps été cantonné aux environnements **Windows** ou **Mac**, on observe de plus en plus d'activité dans ce secteur sur notre système d'exploitation fétiche. Même si la majorité des effets et instruments virtuels disponibles gratuitement ou commercialement n'incluent pas de version **GNU/Linux**, on commence néanmoins à voir des éditeurs de stations de travail audio-numériques commerciales distribuer leur logiciel sur les trois plateformes (comme **Tracktion [1]**, **Bitwig Studio**), ainsi que des outils de plus en plus efficaces pour faire tourner les versions Windows. Du côté des projets libres, il y a également de quoi faire puisque des applications comme **Audacity [2]**, **Ardour [3]** ou **Qtractor [4]** ont aujourd'hui atteint la maturité nécessaire pour être utilisée quotidiennement de manière stable. Ajoutez à cela la quantité de vidéos didactiques disponibles sur le net et vous obtenez les conditions idéales pour vous lancer dès aujourd'hui dans l'univers passionnant de la musique ! La figure 1 montre un aperçu de l'application que nous allons réaliser.

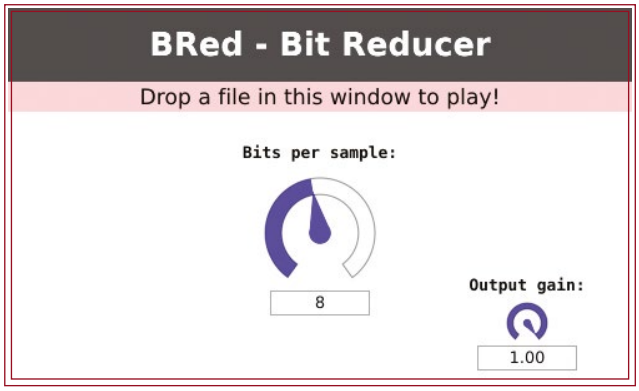


Fig. 1 : L'interface graphique finale de l'application de réduction de la taille d'échantillon que nous allons réaliser.

1. JUCE

Dans ce contexte déjà encourageant pour l'apprenti développeur audio, viennent se greffer des bibliothèques faites pour nous rendre les tâches obscures bien plus faciles et nous permettre de nous concentrer en priorité sur le travail du son en lui-même.

Découvrons ensemble la bibliothèque **JUCE [5]**, tellement répandue parmi les professionnels que vous avez peut-être déjà utilisé une application programmée avec JUCE sans même le savoir.

1.1 Historique

Tout a commencé par un développeur anglais isolé, Julian Storer, qui crée dans les années 2000 une station de travail audio-numérique, Tracktion, qu'il vend ensuite à Mackie. Fort de son expérience, il décide de créer dans la foulée JUCE (pour *Jules' Utility Class Extensions*), une bibliothèque de classes **C++** facilitant grandement la manipulation de flux et de formats audio et MIDI, et la création d'interfaces graphiques, rendant ainsi la vie bien plus aisée aux créateurs d'instruments virtuels et autres filtres. Comme **Qt**, JUCE bénéficie d'une double licence, permettant aux projets libres de l'utiliser sans contrepartie.

Après avoir récupéré les droits sur Tracktion, Julian réécrit son application en utilisant cette fois JUCE, et dans la foulée intègre l'équipe de la startup **ROLI**, fabricant de pianos au toucher tridimensionnel.

Pour nous, développeurs, c'est l'occasion d'utiliser une bibliothèque et des outils de niveau professionnel, puisque, comme nous allons le voir, la même base de code peut être à la fois portée sur toutes les plateformes desktop, dans tous les formats d'instruments virtuels reconnus, ainsi que sur les mobiles **Android** et **iOS**. Certains utilisateurs ont même reporté avoir créé des applications sur **Raspberry Pi**. Plutôt pratique.

1.2 Installation

La méthode conseillée pour se lancer sur un système vierge est d'abord d'installer toutes les dépendances nécessaires, puis de cloner le dépôt **GitHub** de JUCE :

```
$ apt-get install clang llvm libfreetype6-
dev libx11-dev libxinerama-dev libxrandr-dev
libxcursor-dev mesa-common-dev libasound2-dev
freeglut3-dev libxcomposite-dev libcurl4-gnutls-dev
$ git clone --depth 1 https://github.com/
JulianStorer/JUCE
```

Pour ce projet nous allons avoir besoin de certaines fonctionnalités de **C++11** ou **C++14**, il est donc recommandé d'utiliser une version récente de **clang** ou **GCC**.

Les aficionados de **Docker** pourront se baser sur mon container **`adanselm/docker-clang-juce`**.

La deuxième étape est de compiler l'outil de création de projet intitulé « Projucer » :

```
$ cd juce/extras/Projucer/Builds/LinuxMakefile
$ make CONFIG=Release
```

Notez que ce dernier peut être téléchargé indépendamment, et installer les modules pour vous, mais c'est une bonne pratique que de se baser directement sur le dépôt de JUCE, pour pouvoir naviguer librement parmi les différentes versions de la bibliothèque, y compris de plus anciennes si nécessaire.

1.3 Organisation

Depuis la version 2, la bibliothèque se divise en modules. L'outil **Projucer** nous permet de sélectionner ceux qui feront partie de notre projet. Il est également assez simple de créer ses propres modules à réutiliser dans plusieurs applications.

Voici quelques exemples : voir tableau page suivante.

Quand on débute, difficile de savoir quels éléments existent et comment s'en servir. Pour nous aider, trois sources principales d'information :

- Le forum de JUCE (forum.juce.com) ;
- Le Doxygen en ligne (juce.com/doc) ;
- Les nombreux exemples fournis dans le dépôt git.

`juce_core`	Les classes indispensables pour utiliser les chaînes de caractères, les conteneurs, la mémoire, les *threads*, etc.
`juce_graphics`	Comme son nom l'indique : les objets graphiques, images et fontes.
`juce_audio_basics`	Au cœur de notre application audio, il permet de gérer les tableaux d'échantillons sonores, les messages MIDI, etc.
`juce_audio_devices`	Les classes pour manipuler les périphériques d'entrée/sortie audio et MIDI.
`juce_audio_formats`	Les abstractions et codecs pour lire et écrire les formats wav, ogg, aiff, mp3, flac, etc.
`juce_audio_plugin_client`	Les classes nécessaires pour exporter un *plugin* aux formats VST, AudioUnit, AAX, RTAS,...
`juce_audio_processors`	Le pendant de `juce_audio_plugin_client`, il sert à lire ces différents formats d'extensions et donne les abstractions pour créer des processeurs de signal.
`juce_audio_utils`	Les composants d'interface graphique spécialisés dans l'audio (aperçu de forme d'onde, clavier de piano, ...)
`juce_gui_basics`	Les composants d'interface graphique les plus couramment utilisés.

Une fois que vous serez habitués à manipuler les classes principales, vous irez très certainement directement lire le code de la bibliothèque à la moindre interrogation, celle-ci étant un modèle de code lisible et documenté.

2. POSER LES FONDATIONS

Il est maintenant temps de mettre le pied à l'étrier. Pour ce faire, exécutez le Projucer :

```
$ ./build/Release/Projucer
```

Et créez un nouveau projet de type ***Audio Application*** (voir figure 2). Nous l'appellerons « BRed » (pour « Bit Reducer »). Après sélection du dossier où stocker le projet et confirmation du chemin d'accès aux modules (figure 3, ci-contre), le Projucer va créer pour nous quelques fichiers source contenant le point d'entrée principal de l'application, ainsi qu'une fenêtre de base (figure 4, ci-contre)).

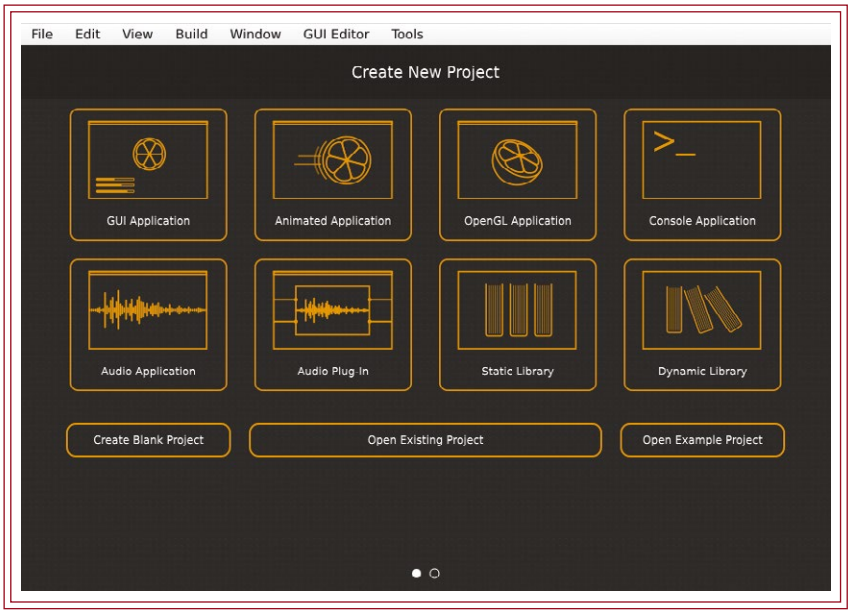


Fig. 2 : Le menu principal à l'ouverture de l'outil Projucer. Nous nous intéressons ici aux projets de type « Audio Application ».

Il devrait également créer une cible d'export « Linux Makefile » qui va générer pour nous le nécessaire pour compiler notre projet.

Si l'on sauvegarde notre projet en cliquant sur ***File > Save Project*** puis qu'on lance `make` depuis le répertoire `Builds/LinuxMakefile` pour le compiler, on devrait obtenir un binaire qui affiche... une passionnante fenêtre blanche... Patience ! C'est une première étape.

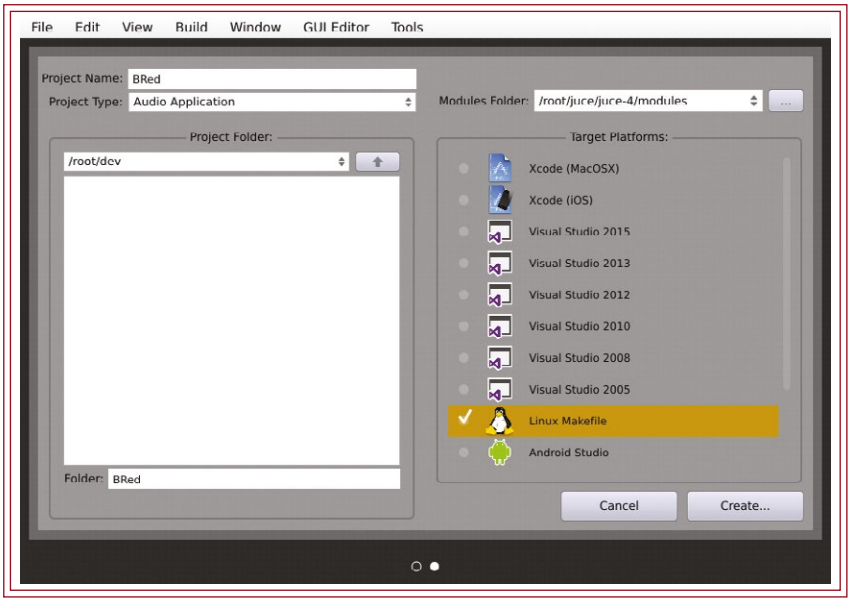


Fig. 3 : L'écran de création du projet. Vérifiez le chemin d'accès aux modules.

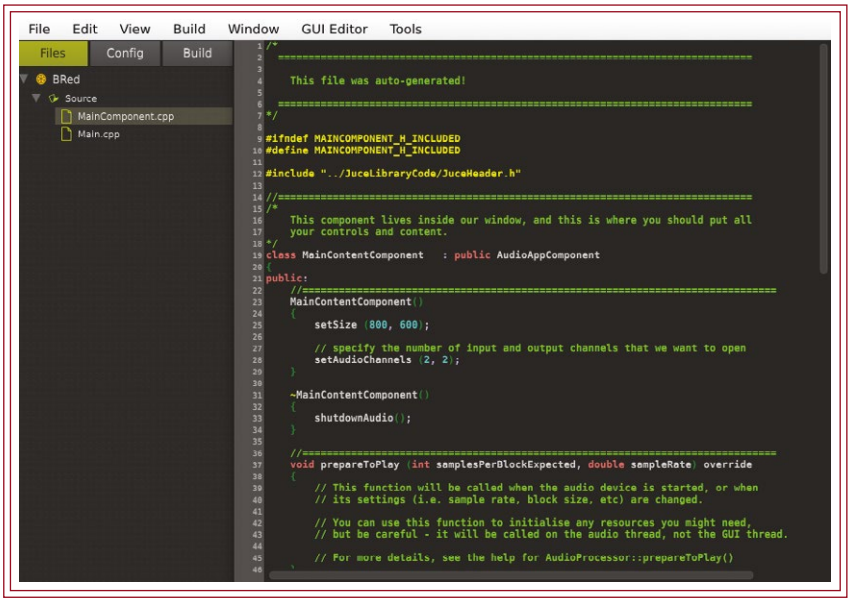


Fig. 4 : Le premier onglet de la fenêtre principale du projet nous montre les fichiers qu'il inclut. Ici, les fichiers créés par défaut correspondent au point d'entrée principal du programme et à un composant graphique vide.

2.1. Lire les fichiers audio

Le squelette d'application créé par le Projucer est déjà capable de manipuler de l'audio. Il va tenter d'ouvrir le périphérique audio par défaut et initialiser deux canaux d'entrée et de sortie par le biais de l'appel de à la méthode **`setAudioChannels(2, 2)`** du constructeur **`MainContentComponent()`**. Nous désirons sortir un signal stéréo vers nos enceintes, donc cette configuration nous convient... à condition d'avoir un système correctement configuré, bien évidemment.

La méthode **`void getNextAudioBlock(const AudioSourceChannelInfo& bufferToFill)`** de cette même classe nous permet donc de nourrir la carte son avec des données audio. En lisant le contenu d'un fichier (WAV, par exemple), et en remplissant le tableau contenu dans **`bufferToFill`** avec ces données, nous devrions entendre le son du média choisi, directement dans les sorties du périphérique audio par défaut.

Reste à donner à notre programme les moyens de lire les différents formats de fichier, et de donner à l'utilisateur la possibilité de glisser-déposer le fameux fichier dans la fenêtre de l'application.

Pour cela, ajoutons deux membres privés à notre classe **`MainContentComponent`** :

```
AudioFormatManager mFormatMgr;
ScopedPointer<AudioFormatReader
Source> mSource;
```

Le premier permet de créer les classes gérant le décodage du fichier. Il nécessite d'être initialisé dans notre constructeur, sans quoi il ne reconnaîtrait aucun format :

```
MainContentComponent()
{
    // Ajoute le support des
formats wav, aiff, ogg, flac,
etc.
    mFormatMgr.
registerBasicFormats();
    // ...
}
```

Le second correspond au flux de données venant du décodeur du fichier. Faisons d'abord comme si nous avions des données à envoyer à la carte son. Nous créerons cette source dans un second temps, lorsque nous ajouterons le support du « glisser-déposer ».

Pour transférer les données depuis la source, modifions les deux méthodes suivantes :

```
void prepareToPlay(int samplesPerBlockExpected,
double sampleRate) override
{
    if( mSource )
    {
        mSource->prepareToPlay(samplesPerBlock
Expected, sampleRate);
    }
}

void getNextAudioBlock(const
AudioSourceChannelInfo& bufferToFill) override
{
    // Vide le buffer d'abord (pour éviter la
sortie de bruits aléatoires)
    bufferToFill.clearActiveBufferRegion();
    if( mSource )
    {
        mSource->getNextAudioBlock(bufferToFill);
    }
}
```

Puis donnons à notre classe la capacité d'y déposer des fichiers depuis notre gestionnaire de fenêtres en la faisant hériter de **FileDragAndDropTarget** :

```
class MainContentComponent :
public AudioAppComponent, public
FileDragAndDropTarget
{
 public:
    // ...
```

Et en implémentant les deux méthodes virtuelles qui en découlent :

```
virtual bool isInterestedInFileDrag(const
StringArray& files)
{
    if(files.isEmpty())
      return false;

    const auto exts = mFormatMgr.
getWildcardForAllFormats().
removeCharacters("*");
    const File dropped( files[0] );
    if( dropped.existsAsFile() &&
dropped.hasFileExtension(exts) )
    {
      return true;
    }
    return false;
}
```

Cette première méthode récupère la liste des formats supportés par la classe **AudioFormatManager** dont nous avons créé une instance précédemment, et se sert de cette liste et de l'extension du fichier déposé par l'utilisateur pour décider si oui ou non ce fichier peut être accepté.

```
virtual void filesDropped(const
StringArray& files, int x, int y)
{
    const File dropped( files[0] );
    auto * reader = mFormatMgr.
createReaderFor(dropped);
    mSource = new
AudioFormatReaderSource(reader, true);
}
```

Ici nous créons effectivement la source qui pourra parcourir les données bloc par bloc, à partir du **reader**, qui lui s'occupe de décoder le format du fichier.

Si tout s'est bien passé, recompiler et exécuter l'application devrait nous permettre de glisser-déposer un fichier audio dans la fenêtre, et entendre le son qu'il contient sur notre périphérique principal.

2.2. Modifier le flux du signal

Ajoutons maintenant au projet une classe **BRedProcessor** du type **AudioProcessor**, dont le but sera de transformer le son, et son interface graphique du type **AudioProcessorEditor**.

LA CLASSE AUDIOPROCESSOR

Elle joue un rôle prépondérant dans JUCE en donnant l'interface d'une boîte noire, qui prend un signal audio et des messages MIDI en entrée, les traite, puis ressort également de l'audio et du MIDI.

Si vous faites de la synthèse sonore (donc un instrument), vous lirez les notes qui arrivent en MIDI et vous vous en servirez pour générer le signal audio en sortie.

Si vous créez un effet, vous modifierez simplement les données audio et/ou MIDI avant la sortie de la boîte noire.

Ici nous allons créer une application indépendante qui encapsulera un `AudioProcessor`. Si vous voulez générer un greffon de format VST ou autre qui pourra être exploité par une application musicale externe, cela ne change rien au niveau du traitement intrinsèque du signal. Créez simplement un projet du type « Audio Plug-In » et réutilisez vos classes de type `AudioProcessor` et `AudioProcessorEditor`.

La classe **AudioProcessor** présente des méthodes virtuelles à implémenter dans notre classe fille pour qu'elle soit fonctionnelle. Concentrons-nous d'abord sur **void processBlock(AudioBuffer<float>& buffer, MidiBuffer& midiMessages)**. C'est elle qui effectue les modifications voulues pour notre effet sur le flux du signal. Notez qu'elle opère à la fois sur les données audio et sur les données MIDI qui lui sont présentées.

Pour le moment, nous allons juste réduire le volume du signal audio entrant en y appliquant un coefficient prédéterminé :

```
void BRedProcessor::processBlock(AudioBuffer
<float>& buffer, MidiBuffer& midiMessages)
{
    buffer.applyGain(0, buffer.
getNumSamples(), 0.5f);
}
```

Nous réduisons ainsi l'intensité sonore de moitié.

ATTENTION !

Prenez garde, le *thread* audio n'attend pas !

Pour produire un signal, votre carte son demande continuellement des données à votre application, par le biais d'un tableau d'échantillons à remplir ou, plus simplement, modifier. Si vos opérations dans la méthode **processBlock()** d'un **AudioProcessor** prennent trop de temps, ou si elles restent bloquées pour se synchroniser avec un autre processus léger, les échantillons seront tout de même envoyés au driver de carte son, que vous ayez fini ou non, et vous obtiendrez un signal distordu, incomplet, avec des craquements audibles très désagréables.

En règle générale, on bannit complètement tout verrou et donc toute allocation mémoire dans cette partie du code. Par conséquent, mieux vaut savoir ce qui se passe derrière les appels de méthode que vous effectuez aveuglément.

Consultez le document *« Design Patterns for Real-Time Computer Music Systems »* **[6]** pour plus d'information.

Les autres méthodes de la classe sont très simples. Elles ne sont d'ailleurs pas toutes obligatoires. Référez-vous au code du projet ou à la documentation pour en savoir plus.

Ce qui nous intéresse avant tout, c'est d'intégrer cette classe d'effet dans le reste du programme.

Nous avons vu précédemment comment remplir le tableau d'échantillons avec le contenu d'un fichier. Il nous suffit maintenant d'avoir sous la main une instance de **BRedProcessor** (en ajoutant **BRedProcessor mProcessor;** comme variable de la classe **MainContentComponent**), et d'appliquer sa méthode **processBlock()** sur ce fameux contenu :

```
void getNextAudioBlock(const
AudioSourceChannelInfo& bufferToFill)
override
{
    if( mSource )
    {
        mSource->getNextAudioBlock(buffer
ToFill);
        mProcessor.
processBlock(*bufferToFill.buffer,
mMidiBuffer);
    }
}
```

Profitons-en également pour remplacer le contenu de notre fenêtre vide par l'interface graphique de notre **BRedProcessor**. Créons pour ce faire un pointeur sur cette interface dans la partie « private » de **MainContentComponent** :

```
private:
    ScopedPointer<AudioProcessorEditor>
mEditor;
```

Et initialisons-le dans son constructeur :

```
MainContentComponent()
: mEditor( mProcessor.createEditorIfNeeded() )
{
    // Ajoute l'éditeur dans l'arborescence
des sous-composants graphiques :
    addAndMakeVisible(mEditor);
    // ...
```

Enfin, pour que ce composant s'affiche correctement, il faut lui donner des coordonnées. C'est la méthode **resized()** qui s'occupe de placer les sous-composants à chaque fois que la fenêtre change de taille :

```
void resized() override
{
    if( mEditor )
    {
        mEditor->setBounds(0, 0,
getWidth(), getHeight());
    }
}
```

Vous pouvez maintenant utiliser le Projucer pour modifier l'apparence de **BRedProcessorEditor** graphiquement (en y ajoutant par exemple un composant de type **Label**

pour dessiner un titre en haut de la page). Jetez un œil au code du projet pour avoir un exemple. Profitez-en pour recopier les quelques lignes nécessaires pour que la destruction du **MainContentComponent** se fasse proprement à l'issue du programme.

2.3. Paramétrer l'effet

Jusqu'à présent, notre effet n'est pas très dynamique... La valeur du gain appliqué au signal est définie en dur dans le code !

Les dernières versions de JUCE proposent une classe qui va grandement nous faciliter la tâche pour gérer l'état de notre **AudioProcessor** : **AudioProcessorValueTreeState**. Pour l'utiliser, vous devez disposer d'un compilateur supportant **std::function**. Si vous obtenez une erreur de compilation, c'est qu'il faut mettre à jour clang ou gcc.

Créer une instance de cette classe dans notre **BRedProcessor** permet de stocker un arbre de propriétés qui peut être aisément importé ou exporté, soit en XML, soit en format binaire. Chaque propriété a un identifiant et une valeur de type variant, que l'on peut lier à un composant graphique.

Par conséquent, en ajoutant cette instance dans **BRedProcessor.h** :

```
private:
    AudioProcessorValueTreeState mState;
```

Nous pouvons désormais déclarer un paramètre nommé « outputGain » qui sera stocké dans l'état **mState**.

```
BRedProcessor::BRedProcessor()
: mState(*this, nullptr)
{
  mState.createAndAddParameter("outputGain",
"Output Gain", "%",

NormalisableRange<float>(.0f, 1.0f, 0.01f), 1.0f,
                                 [](float x){
return String(x); },
                                 [](const String&
s){ return s.getFloatValue(); } );
```

Les trois premiers paramètres de cette méthode correspondent à l'identifiant de la propriété, son nom complet et son suffixe. Vient ensuite la définition de l'intervalle des valeurs possibles : ici entre **0** et **1**, par incrément de **0,01**. Puis sa valeur par défaut : **1**. Et enfin les deux fonctions lambda qui servent à convertir la valeur numérique en chaîne de caractères et inversement, la chaîne de caractères en valeur numérique.

Il nous est donc maintenant possible de substituer la valeur du gain codée en dur par la valeur de cette propriété :

```
void BRedProcessor::processBlock(Aud
ioBuffer<float>& buffer, MidiBuffer&
midiMessages)
{
    float * pGain = mState.getRawParameter
Value("outputGain");
    buffer.applyGain(0, buffer.
getNumSamples(), *pGain);
}
```

Là où les choses deviennent intéressantes, c'est que nous allons pouvoir dynamiquement changer la valeur de cette propriété via notre interface graphique. Ajoutez d'abord un composant de type **Slider** à l'interface graphique de **BRedProcessorEditor** dans le Projucer. Appelons-le **mSlGain**.

Puis dans **BRedProcessorEditor.h** (une fois le projet sauvegardé dans le Projucer pour que le code soit mis à jour), ajoutez un tableau qui contiendra les objets faisant la liaison entre nos **Sliders** et nos différentes propriétés :

```
private:
    //[UserVariables]   -- You can
add your own custom variables in this
section.
    OwnedArray<AudioProcessorValueTree
State::SliderAttachment> mSlAttachments;
    // ...
```

ATTENTION !

Prenez garde de bien ajouter votre code entre les balises prévues à cette effet, sous peine de voir celui-ci disparaître à la prochaine sauvegarde du Projucer.

Et dans le fichier cpp :

```
    //[Constructor] You can add your own
custom stuff here..
    auto * sa = new AudioProcessorValueTree
State::SliderAttachment(state, "outputGain",
*mSlGain);
    mSlAttachments.add( sa );
```

Il faudra au préalable modifier les paramètres pris en compte par ce constructeur pour qu'on puisse lui passer l'**AudioProcessorValueTreeState** (nommé **state** ici) à la création.

Désormais, faire bouger le potard dans notre interface graphique modifiera directement la propriété correspondante dans l'état de **BRedProcessor**, qui sera lue à chaque passage dans la méthode **processBlock()**.

Pour peu que vous ayez bien passé ce fameux état en paramètre à l'instanciation de `BRedProcessorEditor`, vous avez désormais le contrôle total sur l'intensité sonore de votre effet modulo les quelques lignes supplémentaires nécessaires à un nettoyage correct de votre programme.

3. TRAITER LE SON

Maintenant que notre programme est capable de lire un fichier et qu'il donne la possibilité à l'utilisateur d'agir sur les paramètres de l'effet, il est finalement temps de distordre le son...

L'EFFET « BIT CRUSHER »

Ce traitement du son altère directement la façon dont le signal est encodé et fait appel à vos souvenirs d'écoliers. Lorsqu'on encode une forme d'onde correspondant à la variation de pression captée par nos oreilles, on est obligé dans le domaine numérique d'échantillonner le signal, c'est-à-dire de prélever la valeur du signal à intervalle régulier. Cet intervalle est fixe et donne notre fréquence d'échantillonnage. L'échelle sur laquelle se placent les valeurs que l'on relève est la taille de notre échantillon.

Plus ces échelles sont grandes (avec beaucoup de nuances dans les valeurs possibles, et un intervalle très petit), plus le signal pourra être recréé de manière fidèle et tromper notre oreille.

À contrario, réduire la palette de valeurs possibles créera une distorsion dans notre signal. C'est ce que l'on recherche ici.

En général, un effet de type « Bit Crusher » agit à la fois sur la fréquence d'échantillonnage et sur la taille des échantillons, pour donner un effet plus riche [7]. Ici on se contentera de modifier la taille des échantillons. On laissera de côté la modification de l'intervalle d'échantillonnage. Restons simples.

Maintenant que vous êtes rodés, vous pouvez répéter les étapes précédentes pour :

- ajouter un nouveau `Slider` à l'interface graphique de `BRedProcessorEditor` ;
- lier ce `Slider` à une propriété nommée *bitDepth* avec des valeurs comprises entre `1` et `16`, par incrément de `1` ;

• créer un paramètre dans **BRedProcessor** identifié par le nom *bitDepth*.

Et enfin, utiliser la valeur de ce paramètre dans le **processBlock()** pour triturer le son ! Voilà une implémentation naïve mais fonctionnelle de cet algorithme :

```
void BRedProcessor::processBlock(Aud
ioBuffer<float>& buffer, MidiBuffer&
midiMessages)
{
    float * pBitDepth = mState.getRawPara
meterValue("bitDepth");
    jassert(pBitDepth != nullptr);
    const float max = powf(2.0f,
*pBitDepth) - 1;
    for(int sample = 0; sample < buffer.
getNumSamples(); ++sample)
    {
        for(int chan = 0; chan < buffer.
getNumChannels(); ++chan)
        {
            const float oldValue =
buffer.getSample(chan, sample);
            const float newValue =
roundToInt(oldValue * max) / max;
        buffer.setSample(chan, sample,
newValue);
    }
}
// ...
```

On commence par calculer la valeur maximale (**max**) que peut prendre un échantillon en fonction du paramètre *bitDepth* donnant sa taille en bits. Puis, pour chaque échantillon et pour chaque canal (généralement gauche et droite) du tableau, on recalcule sa valeur proportionnellement à la nouvelle échelle.

En recompilant le programme, et en glissant un fichier son sur la fenêtre, vous devriez pouvoir dégrader le signal par simple jeu sur le nouveau potard pilotant le paramètre *bitDepth*.

CONCLUSION

Bien entendu, ce projet n'est qu'une ébauche très rapide, mais il permet déjà de se rendre compte qu'on peut arriver à un résultat probant en seulement quelques minutes. À vous ensuite d'itérer sur le code pour l'améliorer et lui donner de nouvelles fonctionnalités.

Voici quelques idées d'améliorations possibles :

• Jouer le son en boucle ;

• Remplacer la valeur fixe maximale de 16 bits par la vraie valeur lue dans le fichier audio via la classe **AudioSource** ou ses dérivés ;

• Permettre la dégradation de la fréquence d'échantillonnage en plus du nombre de bits par échantillon ;

• Améliorer l'interface graphique en personnalisant les méthodes de la classe **LookAndFeel** lues par chaque composant graphique ;

• En faire une extension utilisable par votre logiciel d'édition de musique favori (par exemple Audacity). Du coup, le son ne sera plus lu dans un fichier, mais directement transmis à notre **processBlock()** via le tableau d'échantillons reçu en paramètre d'entrée.

Comme vous avez pu le constater, la bibliothèque JUCE est plutôt riche. Prenez donc l'habitude de rechercher dans les classes et les exemples disponibles avant de coder une fonctionnalité de zéro. ■

RÉFÉRENCES

[1] Site officiel de Tracktion : http://www.tracktion.com

[2] Site officiel de Audacity : http://www.audacityteam.org

[3] Site officiel de Ardour : http://www.ardour.org

[4] Site officiel de QTractor : http://qtractor.sourceforge.net

[5] Site officiel de JUCE : http://www.juce.com

[6] DANNENBERG R., BENCINA R., *« Design Patterns for Real-Time Computer Music Systems »*, Septembre 2005 : http://www.cs.cmu.edu/~rbd/doc/icmc2005workshop/real-time-systems-concepts-design-patterns.pdf.

[7] Article « Bitcrusher » sur Wikipedia : https://en.wikipedia.org/wiki/Bitcrusher

NOTE

Le code est disponible sur : https://bitbucket.org/springbeats/bred.

C'est simple... c'est possible sur :

http://www.ed-diamond.com

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

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

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

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

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

LINUX MAGAZINE / FRANCE

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

Vos remarques :

.....

.....

VOICI TOUTES LES OFFRES COUPLÉES AVEC GNU/LINUX MAGAZINE !

POUR LE PARTICULIER ET LE PROFESSIONNEL ...

CHOISISSEZ VOTRE OFFRE !

Prix TTC en Euros / France Métropolitaine

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

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

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

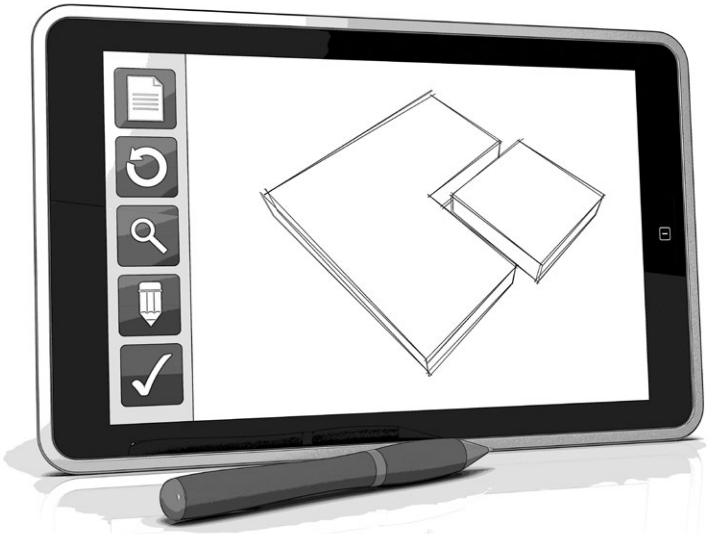
PRO OU PARTICULIER = CONNECTEZ-VOUS SUR :

http://www.ed-diamond.com pour consulter toutes les offres !

LA CRÉATION D'INTERFACES GRAPHIQUES AVEC WXPYTHON

OLIVIER BÉLANGER
[Développeur de logiciels audio]

MOTS-CLÉS : WXPYTHON, PROGRAMMATION, INTERFACE GRAPHIQUE, MULTI-PLATEFORMES, DÉVELOPPEMENT LOGICIEL



Cet article présente une introduction à la création d'interfaces graphiques avec WxPython, un port de la librairie WxWidgets pour le langage de programmation Python. La combinaison Python/WxPython permet d'écrire du code simple et efficace, offrant une apparence native sur toutes les plateformes majeures. Après un survol rapide des principaux éléments de langage propres à la création d'interfaces graphiques avec WxPython, un exemple concret sera présenté. Nous développerons, étape par étape, un jeu de mémoire aux logos de différentes distros linux.

NOTE

Cet article portant sur l'utilisation d'un module **Python [1]**, une connaissance de base du langage est assumée de la part du lecteur. Les tutoriels interactifs **[2]** disponibles sur le site web de Python constituent un bon point de départ!

Le code présenté dans cet article est disponible sur le compte GitHub de l'auteur : https://github.com/belangeo/memory

Le langage de programmation Python en lui-même ne permet pas la création d'interfaces graphiques. Par contre, il existe un certain nombre de modules donnant accès à des librairies conçues expressément pour le développement d'interfaces homme-machine. De nos jours, les deux librairies les plus courantes sont **pyQT** et **wxPython**. Elles offrent toutes deux une vaste gamme de classes permettant l'implantation des comportements standards d'un logiciel. Elles ont aussi l'avantage d'être multi-plateformes, c'est-à-dire que le même code va être exécuté correctement sous les systèmes d'exploitation les plus répandus (**Windows**, **OS X**

et **Linux**). Pour cette introduction au design d'interfaces graphiques, nous utiliserons la librairie wxPython **[3]**, qui doit être indépendamment installée sous la distribution courante de Python. Sous les systèmes à base **Debian**, la version 3.0 de WxPython, compatible avec Python 2 est disponible via le gestionnaire de paquet :

```
sudo apt-get install python-wxgtk3.0
```

Au moment d'écrire ces lignes, la version de WxPython (Phoenix) **[4]** compatible avec Python 3, doit être compilée manuellement sous les différentes distributions linux.

NOTE

Les lignes de code que vous rencontrerez dans cet article roulent sous Python 3, avec la version Phoenix de WxPython. Le code disponible sur Github fonctionne sous les versions 2 et 3 de Python sans modifications.

Cet article est divisé en deux sections. Dans un premier temps, une brève introduction à la bibliothèque illustrera les quatre étapes essentielles à la construction d'une interface graphique. Ces étapes sont :

- La gestion de la boucle d'exécution principale du logiciel ;
- Les conteneurs, c'est-à-dire la fenêtre et les panneaux où sera exposée l'interface ;
- Le contenu, c'est-à-dire les objets offrant une interaction avec l'utilisateur ;
- L'interaction entre l'interface et les fonctionnalités du programme, sous la forme de méthodes.

En seconde partie de l'article, un exemple concret de jeu avec interface graphique sera présenté. Pour l'occasion, nous développerons, étape par étape, un jeu de mémoire aux logos de distros linux. Les éléments standards d'une application graphique, tels que la disposition des objets, la création d'une barre de menu, l'interaction avec l'utilisateur et la fenêtre « About... », seront illustrés.

1. INTRODUCTION À LA LIBRAIRIE WXPYTHON

1.1 La gestion de la boucle d'exécution

La gestion de la boucle principale se fait par le biais d'un objet `wx.App` ou d'un de ses dérivés. L'objet `wx.App`, qui initialise les comportements de base du logiciel, permet de mettre en place une application `wx` très rapidement. Un point important, qui vaut pour toute librairie d'interface graphique, est que cette dernière prendra éventuellement possession de la boucle d'exécution de Python, permettant ainsi au logiciel de rester actif pour une durée indéterminée. WxPython prend possession de la boucle à partir du moment où la méthode `MainLoop` est appelée sur un objet de la classe `wx.App`. Cet appel est en général à la dernière ligne du script principal puisque les lignes suivantes ne seront exécutées que lorsque la dernière fenêtre du programme sera détruite.

Le script suivant commence par importer la librairie `wx`, puis déclare un objet `wx.App`. Un objet de la classe `wx.App` doit absolument exister avant de commencer à construire des objets d'interfaces. Une fenêtre est ensuite créée (`wx.Frame`) et affichée à l'écran. Nous détaillerons les **conteneurs** au point suivant. Enfin, lorsque tout est en place, on appelle la méthode `MainLoop` sur notre objet `wx.App`. La boucle d'exécution passe alors aux mains de notre application graphique et attend de recevoir des événements pour exécuter diverses fonctions.

```
import wx
app = wx.App()
frame = wx.Frame(None, title='Simple
App', size=(250, 200))
frame.Show()
app.MainLoop()
```

La figure 1 montre la fenêtre obtenue.

Fig. 1 : Une application wx minimale.

1.2 Les conteneurs

La boucle d'exécution du programme est maintenant sous le contrôle de l'interface tant que celle-ci est active. Un conteneur a déjà été créé puisqu'une interface graphique ne peut vivre sans au moins une fenêtre affichée à l'écran. Nous utiliserons deux objets de la librairie WxPython en guise de conteneur. Le premier est `wx.Frame` et constitue le cadre principal de notre application, c'est-à-dire la fenêtre délimitant l'espace occupé par notre application à l'écran. Dans cette fenêtre, nous allons créer un ou plusieurs panneaux, avec l'objet `wx.Panel`, où seront affichés tous les éléments d'interface de notre programme.

Les objets fournis dans une librairie graphique constituent généralement une base à partir de laquelle nous pouvons

construire une interface avec ses fonctionnalités propres. La technique la plus répandue consiste à créer des classes dérivées de classes contenues dans la librairie. Nous allons donc définir notre propre classe **MyFrame**, avec **wx.Frame** comme classe parente, pour afficher le cadre de l'application. Pour qu'une classe enfant possède bien toutes les caractéristiques de la classe parente, la méthode **__init__** de la classe parente doit être appelée dans la méthode constructeur de la classe enfant. Une fois le cadre bien installé, nous allons créer un panneau, avec l'objet **wx.Panel**, qui servira à accueillir les objets d'interface de notre application. La méthode **SetBackgroundColour** permet de spécifier une couleur de fond à notre panneau.

```
import wx

class MyFrame(wx.Frame):
    def __init__(self, parent, title,
pos, size):
        wx.Frame.__init__(self, parent,
title=title, pos=pos, size=size)
        self.panel = wx.Panel(self)
        self.panel.
SetBackgroundColour("#FFFFFF")

app = wx.App()
frame = MyFrame(None, title='Simple App',
pos=(20, 20), size=(250, 200))
frame.Show()
app.MainLoop()
```

Fig. 2 : Une application avec un panneau de fond.

Le premier argument (**parent**) de pratiquement tous les objets de la librairie permet de spécifier le conteneur dans lequel sera placé l'objet défini. Cet argument doit être correctement spécifié pour obtenir un affichage conséquent de l'interface. Dans le script ci-dessus, le **parent** du panneau est **self**, c'est-à-dire la fenêtre principale du programme, tandis que le **parent** de la fenêtre est **None**, indiquant à WxPython que l'objet **frame** est bel et bien la fenêtre principale (*TopLevelWindow*) de l'application. Nous obtenons ici une fenêtre avec un panneau de fond (voir figure 2).

1.3 Le contenu

Le contenu comprend tous les objets graphiques affichés à l'intérieur des panneaux. Nous utiliserons dans cette introduction le texte statique (**wx.StaticText**), afin de souhaiter le bonjour au lecteur, et le menu déroulant (**wx.Choice**), qui servira éventuellement à changer la couleur de fond du panneau. Les paramètres d'initialisation varieront en fonction des différents objets, sauf pour le premier argument, qui correspond toujours au conteneur à l'intérieur duquel l'objet doit être affiché.

```
import wx

class MyFrame(wx.Frame):
    def __init__(self, parent, title,
pos, size):
        wx.Frame.__init__(self, parent,
title=title, pos=pos, size=size)
        self.panel = wx.Panel(self)
        self.panel.
SetBackgroundColour("#FFFFFF")
        label = wx.StaticText(self.panel,
label="Allo GLMF", pos=(90,40))
        choices = ["white", "red",
"green", "blue", "yellow"]
        choose = wx.Choice(self.panel,
choices=choices, pos=(85,70))
        choose.SetSelection(0)

app = wx.App()
frame = MyFrame(None, title='Simple App',
pos=(20,20), size=(250, 200))
frame.Show()
app.MainLoop()
```

Le résultat de ce code est visible en figure 3.

Fig. 3 : Ajout d'objets graphiques sur l'interface.

1.4 Interaction avec le programme

L'interaction entre la manipulation des objets et les modifications à apporter au programme s'effectue à l'aide d'un système d'événements auxquels sont associées des méthodes.

Les événements en WxPython sont des constantes sous la forme **wx.EVT_EVENTTYPE**. Par exemple, le type d'événement envoyé lors de la manipulation d'un bouton est **wx.EVT_BUTTON**. Pour le menu déroulant, ce sera **wx.EVT_CHOICE**.

On lie un événement à une méthode à l'aide de la méthode **Bind** de la classe **wx.EvtHandler** (classe parente de pratiquement tous les objets de la librairie) :

```
obj.Bind(event, handler)
```

La méthode appelée (**handler**) recevra en argument un objet dérivé de la classe **wx.Event** contenant les informations concernant l'action qui a généré l'événement.

Dans l'exemple ci-dessous (résultat en figure 4), la méthode **changeBackColour**, qui sera appelée lors de la manipulation du menu déroulant, prend donc deux arguments: l'objet lui-même (**self**) ainsi que l'événement généré (**evt**). Cette liaison entre un événement survenu à l'écran et l'appel d'une méthode du programme permet donc de modifier le processus en cours en fonction d'une manipulation effectuée par l'utilisateur. On appelle cela la programmation événementielle.

```
import wx

class MyFrame(wx.Frame):
    def __init__(self, parent, title,
pos, size):
        wx.Frame.__init__(self, parent,
title=title, pos=pos, size=size)
        self.panel = wx.Panel(self)
        self.panel.
SetBackgroundColour("#FFFFFF")
        label = wx.StaticText(self.panel,
label="Allo GLMF", pos=(90,40))
        choices = ["white", "red",
"green", "blue", "yellow"]
        choose = wx.Choice(self.panel,
choices=choices, pos=(85,70))
        choose.SetSelection(0)
        choose.Bind(wx.EVT_CHOICE, self.
changeBackColour)

    def changeBackColour(self, evt):
        self.panel.
SetBackgroundColour(evt.GetString())

app = wx.App()
frame = MyFrame(None, title='Simple App',
pos=(20,20), size=(250,200))
frame.Show()
app.MainLoop()
```

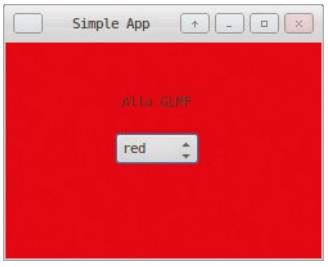


Fig. 4 : Interaction avec le programme.

Ceci conclu cette brève introduction au cycle de développement d'un programme avec interface graphique en WxPython. Pour chaque nouvelle fonctionnalité que l'on désire apporter au programme, il suffit de répéter les étapes 3 et 4, c'est-à-dire l'ajout d'un élément de contrôle graphique et son intégration au programme en liant un événement à une méthode.

2. DÉVELOPPEMENT D'UN JEU DE MÉMOIRE

Dans cette section, nous allons construire, étape par étape, un jeu de mémoire simple. Ce petit programme nous permettra de couvrir plusieurs des éléments standards d'une application avec interface graphique.

2.1 Création de la fenêtre principale

Dans un premier temps, nous allons créer et afficher une fenêtre vide, dans laquelle prendra place notre jeu. Au passage, une variable globale **PATH** est définie afin d'indiquer au programme où se trouvent les images à afficher sur les cartes du jeu.

```
import os, random, wx, wx.lib.buttons as buttons
from wx.adv import AboutDialogInfo, AboutBox

PATH = os.path.join(os.getcwd(), "images")

class MemoryFrame(wx.Frame):
    def __init__(self):
        wx.Frame.__init__(self, None,
title="Jeu de Mémoire")

if __name__ == "__main__":
    app = wx.App()
    frame = MemoryFrame()
    frame.Show()
    app.MainLoop()
```

2.2 Création de la barre de menu

La barre de menu est un élément incontournable de toute application avec interface graphique. On y retrouve plusieurs commandes nécessaires au bon fonctionnement du programme. Avec WxPython, on peut attacher une barre de menu à une fenêtre avec la méthode **SetMenuBar**. Dans la méthode **createMenuBar**, nous allons donc créer et attacher à la fenêtre principale un objet **wx.MenuBar**, dans lequel nous allons définir des menus avec **wx.Menu** et placer quelques items dans ces menus avec la méthode **Append**.

```
    def __init__(self):
        ...
        self.createMenuBar()

    def createMenuBar(self):
        menubar = wx.MenuBar()
        fileMenu = wx.Menu()
        fileMenu.Append(wx.ID_NEW,
"Nouvelle Partie\tCtrl+N")
```

```
        self.Bind(wx.EVT_MENU, self.
initGame, id=wx.ID_NEW)
        fileMenu.Append(wx.ID_EXIT,
"Quitter\tCtrl+Q")
        self.Bind(wx.EVT_MENU, lambda x:
self.Destroy(), id=wx.ID_EXIT)
        helpMenu = wx.Menu()
        helpMenu.Append(wx.ID_ABOUT, "À
propos de Memory")
        self.Bind(wx.EVT_MENU, self.
onShowAbout, id=wx.ID_ABOUT)
        menubar.SetMenus([(fileMenu,
"Fichier"), (helpMenu, "Help")])
        self.SetMenuBar(menubar)

    def onShowAbout(self, evt):
        info = AboutDialogInfo()
        info.SetVersion("1.0")
        info.SetName("Memory")
        info.SetDescription("Un petit jeu
de mémoire écrit avec WxPython.")
        info.SetCopyright("Olivier
Bélanger (2016)")
        AboutBox(info)
```

Notez le caractère tabulation **\t**, suivi d'une commande clavier, inséré dans le nom de certains items des menus. Cette technique permet de définir un raccourci clavier pour une commande menu. Le code ci-dessus présente aussi la méthode **onShowAbout**, appelée par la commande menu ***About Memory***. On affiche les informations données à un objet **AboutDialogInfo** à l'aide de la fonction **AboutBox**. Nous reviendrons sous peu sur le contenu de la méthode **initGame**, appelée lors de l'initialisation d'une nouvelle partie.

2.3 Création des cartes

Pour la création des cartes du jeu, nous utiliserons un bouton à deux états permettant de spécifier une image pour chaque état. La méthode **createButtons** crée tout d'abord une image noire opaque, commune à toutes les cartes. Elle génère ensuite une liste où le nom de chaque logo apparaît deux fois et finalement, la méthode **createButton** est utilisée pour créer autant de boutons qu'il y a de cartes à afficher sur le jeu.

```
    def __init__(self):
        ...
        self.createButtons()

    def createButton(self, id):
        button = buttons.
GenBitmapToggleButton(self, id, self.
back)
        button.Bind(wx.EVT_BUTTON, self.
onToggleButton)
        return button

    def createButtons(self):
        self.back = wx.Image(64, 64).
ConvertToBitmap()
        self.imgs = [f for f in
os.listdir(PATH) if f.endswith(".png")] * 2
        self.togs = [self.createButton(id) for
id in range(len(self.imgs))]
```

2.4 Création des panneaux et mise en place

Trois méthodes seront créées afin de placer les éléments visuels dans la fenêtre. La méthode **createLabelBox** génère une première boîte ajustable dont le contenu consiste simplement en deux chaînes de caractères. Elles serviront à indiquer le nombre de tentatives et le temps écoulé depuis le début de la partie. La méthode **createGameBox** construit l'élément principal de l'interface, c'est-à-dire la grille de cartes à jouer. L'objet **wx.GridSizer** nous rend la tâche facile puisqu'il permet de placer un groupe d'objets sur une grille dont le nombre de rangées et de colonnes peut être variable. La méthode **layout** rassemble les deux boîtes dans la boîte principale, ajuste la taille de la fenêtre et affiche le jeu à l'écran.

```
    def __init__(self):
        ...
        self.createLabelBox()
        self.createGameBox()
        self.layout()

    def createLabelBox(self):
        self.labelbox = wx.BoxSizer(wx.
HORIZONTAL)
        self.counted = wx.StaticText(self, -1,
"Coups joués:   0")
        self.elapsed = wx.StaticText(self, -1, "|
Temps écoulé: 00:00")
        self.labelbox.Add(self.counted, 0,
wx.ALL, 7)
        self.labelbox.Add(self.elapsed, 0,
wx.ALL, 7)

    def createGameBox(self):
        self.gamebox = wx.GridSizer(6, 6, 1, 1)
        self.gamebox.AddMany(self.togs)

    def layout(self):
```

```
        box = wx.BoxSizer(wx.VERTICAL)
        box.AddMany([self.labelbox, self.
gamebox])
        self.SetSizerAndFit(box)
        self.CenterOnScreen()
```

2.5 Initialisation d'une partie

À l'initialisation d'une nouvelle partie, on appelle la méthode **initGame**. Cette méthode effectue plusieurs actions. D'abord, la liste d'images est mélangée, les images sont réassignées aux boutons avec la méthode **SetBitmapSelected** et on assure qu'elles ne sont pas visibles avec la méthode **SetToggle(False)**. Ensuite, une nouvelle horloge est créée (**wx.Timer**) et toutes les variables nécessaires au bon fonctionnement du jeu sont remises à **0**.

```
    def __init__(self):
        ...
        self.initGame()

    def initGame(self, evt=None):
        random.shuffle(self.imgs)
        for i, img in enumerate(self.imgs):
            bitmap = wx.Bitmap(os.path.
join(PATH, img), wx.BITMAP_TYPE_PNG)
            self.togs[i].
SetBitmapSelected(bitmap)
            self.togs[i].SetToggle(False)
        self.timer = wx.Timer(self)
        self.Bind(wx.EVT_TIMER, self.
onTimer)
        self.clicked = []
        self.found = self.time = self.
trials = 0
        self.counted.SetLabel("Coups joués:
0")
        self.elapsed.SetLabel("| Temps
écoulé: 00:00")
```

2.6 Actions pendant le jeu

Chaque fois que le joueur clique sur une carte, la méthode **onToggleButton** est appelée. C'est ici que se déroule l'action du jeu. Premièrement, on vérifie si l'horloge est en marche et si elle ne l'est pas, on la démarre. À partir de ce moment, la méthode **onTimer** sera appelée à chaque seconde et mettra à jour le temps écoulé depuis le début de la partie. Le joueur ne peut cacher une carte lui-même, alors après avoir récupéré l'*id* de la carte sélectionnée, on vérifie si elle est face visible ou non. Si elle n'est pas visible, on la retourne automatiquement avec **SetToggle(True)** et on attend la prochaine action. Si le joueur a retourné une nouvelle carte,

Fig. 5 : Jeu de mémoire complété.

on ajoute celle-ci à la liste des cartes à comparer et si cette liste contient deux cartes, on compare le nom de l'image enregistrée pour chacune d'elles. Si les noms ne sont pas identiques, on appelle la méthode **hide** avec un délai d'une seconde (**wx.CallLater**) et on interrompt l'interaction entre l'utilisateur et l'interface avec la méthode **Disable**. Le délai écoulé, la méthode **hide** retourne les cartes, vide la liste des cartes à comparer et réactive le jeu en appelant la méthode **Enable** sur la fenêtre. Si les cartes comparées sont identiques, on les laisse face vers le haut et on augmente le compteur des images trouvées. Lorsque toutes les images sont visibles, l'horloge est arrêtée et la partie est terminée (voir figure 5).

```
    def onToggleButton(self, evt):
        if not self.timer.IsRunning():
            self.timer.Start(1000)
        id = evt.GetId()
        if self.togs[id].GetToggle():
            self.clicked.append((id, self.
imgs[id]))
            if len(self.clicked) == 2:
                if self.clicked[0][1] !=
self.clicked[1][1]:
                    wx.CallLater(1000,
self.hide)
                    self.Disable()
```

```
                else:
                    self.clicked = []
                    self.found += 1
                    if self.found ==
len(self.imgs) / 2:
                        self.timer.Stop()
                self.trials += 1
                self.counted.
SetLabel("Coups joués: %3d" % self.trials)
        else:
            self.togs[id].SetToggle(True)

    def hide(self):
        while self.clicked:
            self.togs[self.clicked.pop()
[0]].SetToggle(False)
        self.Enable()

    def onTimer(self, evt):
        self.time += 1
        t = "| Temps écoulé: %02d:%02d" %
(self.time / 60, self.time % 60)
        self.elapsed.SetLabel(t)
```

CONCLUSION

Dans cet article, nous avons présenté les bases essentielles au développement d'une application graphique avec la librairie WxPython. Nous avons notamment exploré le concept de programmation événementielle, à la source de la grande majorité des librairies d'interface graphique. Ces concepts fondamentaux ont ensuite été illustrés concrètement avec la création, étape par étape, d'un jeu de mémoire aux logos de différentes distros linux.

En espérant avoir donné le goût au lecteur de se lancer dans le développement d'interfaces graphiques... ■

RÉFÉRENCES

[1] Site officiel du langage Python : https://www.python.org/

[2] Tutoriels interactifs concernant le langage Python : http://www.learnpython.org/

[3] N. Rappin, R. Dunn, *« WxPython in Action »*, Manning Publication, 2006.

[4] Sources de WxPython Phoenix : https://github.com/wxWidgets/Phoenix

CRÉEZ VOTRE CLAVIER PROGRAMMABLE

TRISTAN COLOMBO

MOTS-CLÉS : CLAVIER PROGRAMMABLE, AUTOMATISATION, DIY, GAMING, PRODUCTIVITÉ

Qui n'a jamais rêvé de disposer de touches dédiées permettant d'automatiser des actions au clavier ou à la souris, le tout configurable simplement ? Nos claviers ne disposent pas forcément de suffisamment de touches et les logiciels des claviers de gaming ne sont pas compatibles avec Linux. Autant prendre un clavier à 10€ et en faire un clavier programmable !

Nous allons créer un clavier programmable en branchant sur notre ordinateur un second clavier « standard ». Nous en profiterons pour adopter une méthode de développement que je qualifierai de « R&D » et qui suivra l'évolution réelle de ce projet parti d'un test anodin : un clavier traînait sur le bureau alors que je travaillais sur mon ordinateur portable et je me suis dit qu'il pouvait être intéressant de bénéficier de touches supplémentaires... Voici donc les étapes qui m'ont permises de rendre ce clavier programmable.

1. RECONNAÎTRE LE SECOND CLAVIER... ET LE DÉSACTIVER

Pour commencer, nous allons avoir besoin de l'outil `xinput` qu'il faut installer au préalable :

```
$ sudo apt install xinput
```

Cet outil permet de lister nos périphériques :

```
$ xinput --list
⎡ Virtual core pointer                    id=2    [master pointer  (3)]
⎜   ↳ Virtual core XTEST pointer          id=4    [slave  pointer  (2)]
⎜   ↳ Logitech G500                       id=8    [slave  pointer  (2)]
⎜   ↳ Logitech G500                       id=9    [slave  pointer  (2)]
⎜   ↳ Logitech Logitech Illuminated Keyboard  id=11   [slave  pointer  (2)]
⎣ Virtual core keyboard                   id=3    [master keyboard (2)]
    ↳ Virtual core XTEST keyboard         id=5    [slave  keyboard (3)]
    ↳ Power Button                        id=6    [slave  keyboard (3)]
    ↳ Power Button                        id=7    [slave  keyboard (3)]
    ↳ Logitech Logitech Illuminated Keyboard  id=10   [slave  keyboard (3)]
    ↳ Eee PC WMI hotkeys                  id=12   [slave  keyboard (3)]
    ↳ Logitech USB Keyboard               id=13   [slave  keyboard (3)]
    ↳ Logitech USB Keyboard               id=14   [slave  keyboard (3)]
```

Normalement, si vous venez de brancher votre second clavier, celui-ci apparaîtra dans les dernières lignes. Nous pouvons noter que l'identifiant de la première occurrence est **13**. Demandons donc plus d'informations sur ce périphérique :

```
$ xinput --list-props 13
Device 'Logitech USB Keyboard':
        Device Enabled (146):   1
        Coordinate Transformation Matrix (148): 1.000000,
0.000000, 0.000000, 0.000000, 1.000000, 0.000000,
0.000000, 0.000000, 1.000000
        Device Product ID (267):        1133, 49948
        Device Node (268):      "/dev/input/event12"
```

Notez la présence du *Device Node*, nous en aurons besoin dans la suite.

Si vous vous placez dans un terminal et que vous appuyez sur des touches du second clavier... les lettres apparaissent à l'écran, ce qui est normal pour un clavier. Dans le cadre de notre utilisation, ce fonctionnement doit être désactivé. Pour cela, en utilisant l'identifiant récupéré précédemment (**13** pour moi), il faut exécuter :

```
$ xinput --set-prop 13 "Device Enabled" 0
```

Utilisez à nouveau votre second clavier : plus rien ne se passe ! Les paramètres passés à **xinput --set-prop** sont l'identifiant du *device*, la propriété à modifier (ici **Device Enabled**) et la valeur correspondant à l'action souhaitée (**0** pour désactiver et **1** pour activer). Ainsi, si vous souhaitez réactiver votre clavier vous pourrez taper :

```
$ xinput --set-prop 13 "Device Enabled" 1
```

2. RÉCUPÉRER AUTOMATIQUEMENT LES IDENTIFIANTS

Notre programme de gestion du clavier sera écrit en **Python** et nous devrons lui transmettre le *Device Node* pour interagir avec le clavier. De plus, l'identifiant et le *Device Node* peuvent changer d'un démarrage/débranchement-rebranchement à l'autre. Nous allons donc automatiser cette étape. Vous avez pu noter le nom de votre clavier fourni par la commande **xinput --list**... une simple commande shell nous permet alors de récupérer l'identifiant :

```
$ xinput --list | grep
"Logitech USB Keyboard" |
head -n1 | cut -f2 | cut
-d= -f2
```

Nous appliquons la même technique pour le *Device Node*. Le problème qui se pose est que, pour obtenir le *Device Node,* notre commande a besoin de connaître l'identifiant obtenu précédemment. Nous allons donc créer un petit script shell **run_keyboard.sh** :

```
01: #!/bin/bash
02:
03: KEYBOARD='Logitech USB
Keyboard'
04:
05: # Get ${KEYBOARD} id
06: id='xinput --list |
grep "${KEYBOARD}" |
head -n1 | cut -f2 | cut
-d= -f2'
07:
08: # Get ${KEYBOARD} Node
Device
09: device='xinput --list-
props ${id} |
grep 'Device Node' |
cut -f3 | sed 's/"//g'`
10:
11: # Disable ${KEYBOARD}
12: xinput --set-prop
${id} "Device Enabled" 0
```

N'oubliez pas de donner les droits en exécution :

```
$ chmod ugo+x run_keyboard.sh
```

Pour tester les valeurs d'**id** et de **device** il vous suffit d'ajouter des **echo** en fin de programme. Maintenant que nous avons désactivé le clavier, l'identifiant peut être transmis à notre script Python qui va prendre la main.

3. UTILISER LE MODULE EVDEV POUR DÉTECTER LES TOUCHES UTILISÉES

Le module **evdev** (*EVent DEVice*) va nous permettre de scruter les événements du clavier. Pour l'installer, nous utilisons **pip3** :

```
$ sudo pip3 install evdev
```

Notre script s'appellera **keyboard.py**. Nous pouvons donc modifier **run_keyboard.sh** pour l'appeler :

```
...
14: # Python script call
15: python3 keyboard.py ${device}
```

Passons à l'écriture du script **keyboard.py** :

```
01: import evdev
02: from evdev import ecodes
03: import sys
04:
05: def getDevice():
06:     if len(sys.argv) != 2:
07:         print('Syntax : python3 keyboard.
py <device>')
08:         exit(1)
09:     return sys.argv[1]
10:
11: def readKeyboard(device):
12:     for event in device.read_loop():
13:         if event.type == ecodes.EV_KEY:
14:             if event.code == ecodes.KEY_A
and event.value == 1:
15:                 print('Touche A détectée')
16:
17: if __name__ == '__main__':
18:     dev = getDevice()
19:     device = evdev.InputDevice(dev)
20:     print('Connected to', dev)
21:
22:     readKeyboard(device)
```

Nous importons **evdev** en ligne 1, puis en ligne 2 **ecodes**, le sous-module d'**evdev** permettant de travailler avec des variables associées aux codes des touches et, enfin, **sys** en ligne 3 pour accéder aux paramètres transmis depuis la ligne de commandes. La fonction **getDevice()** des lignes 5 à 9 permet de s'assurer que l'utilisateur a bien transmis un paramètre et retourne ledit paramètre (qui doit être le *Device Node* envoyé par **run_keyboard.sh**). Dans les lignes 11 à 15 la fonction **readKeyboard()** réalise une boucle infinie et scrute les événements clavier (le **device** transmis en paramètre). Si un événement apparaît sur une touche alors son type est **ecodes.EV_KEY** (test de la ligne 13). Dans ce cas, en ligne 14, nous testons s'il s'agit d'une pression sur la touche **<A >** (**ecodes.KEY_A**). La valeur de l'événement, dans le cas d'un clavier, sera :

- **0** : touche relâchée ;
- **1** : touche enfoncée ;
- **2** : touche maintenue enfoncée.

Si l'utilisateur appuie sur **<A>** nous affichons donc un petit message (ligne 15). Le programme principal des lignes 17 à 22 se contente de récupérer le *Device Node* (ligne 18), d'initialiser un objet **InputDevice** sur ce *device* (ligne 19), d'afficher un message de log et de lancer la fonction **readKeyboard()** pour scruter les événements du clavier.

Pour exécuter ce code et avoir accès au *device* du clavier, il faudra le lancer en tant qu'administrateur :

```
$ sudo ./run_keyboard.sh
```

NOTE

Vous vous demandez peut-être pourquoi démarrer le script en super utilisateur ? Il faut savoir que tous les événements liés aux périphériques de saisie sont accessibles par evdev. Il faut donc éviter que tout le monde ait accès à **/dev/input** et puisse ainsi espionner l'utilisateur à l'aide d'un *keylogger* (même si dans les faits l'écriture d'un *keylogger* n'est guère complexe).

Vous remarquerez ensuite qu'un appui sur la touche **<A>** ne provoque aucun affichage... par contre avec la touche **<Q>** cela fonctionne ! Comme nous avons désactivé le clavier, celui-ci n'est plus reconnu en azerty. Il me paraît alors plus judicieux d'utiliser directement le code des touches. Vous pouvez appliquer la modification suivante pour afficher le code, le symbole et l'état des touches sur lesquelles vous appuierez :

```
...
11: def readKeyboard():
12:     for event in device.read_loop():
13:         if event.type == ecodes.EV_KEY:
14:             print('Code de touche :
{} - Symbole : {} - état {}'.format(event.code,
ecodes.KEY[event.code], event.value))
...
```

La liste **ecodes.KEY** contient les noms symboliques des touches dont le code est passé en index. Au lancement du programme vous obtiendrez quelque chose de semblable à :

ET AVEC UN JOYPAD ?

Il est possible d'employer exactement la même technique que celle décrite ci-dessus pour utiliser un Joypad (par exemple pour contrôler un robot). Par contre, pour découvrir son *Device Node* il faudra changer de logique car ce dernier n'apparaît pas dans la liste **xinput** :

```
$ lsusb
Bus 006 Device 005: ID 0cf3:3000 Atheros Communications, Inc. AR3011 Bluetooth (no
firmware)
Bus 006 Device 004: ID 046d:c318 Logitech, Inc. Illuminated Keyboard
Bus 006 Device 003: ID 046d:c068 Logitech, Inc. G500 Laser Mouse
Bus 005 Device 006: ID 046d:c21d Logitech, Inc. F310 Gamepad [XInput Mode]
...
```

On voit ici que mon joypad est un **Logitech F310** (et non, je ne suis pas sponsorisé par Logitech!). Nous allons maintenant rechercher le joypad dans **/proc/bus/input/devices** :

```
$ cat /proc/bus/input/devices
...
I: Bus=0003 Vendor=046d Product=c21d Version=4014
N: Name="Logitech Gamepad F310"
P: Phys=usb-0000:00:1a.0-1.4/input0
S: Sysfs=/devices/pci0000:00/0000:00:1a.0/usb5/5-1/5-1.4/5-1.4:1.0/input/input18
U: Uniq=
H: Handlers=event0 js0
B: PROP=0
B: EV=20000b
B: KEY=7cdb000000000000 0 0 0 0
B: ABS=3003f
B: FF=107030000 0
```

Nous voyons qu'il s'agit de **/dev/input/event0**.

En modifiant légèrement le code de **keyboard.py** (et en le renommant **joypad.py**) pour être plus générique, nous pourrons afficher les événements du joypad :

```
...
11: def readJoypad(device):
12:     for event in device.read_loop():
13:         print('Type evt : {} - Code : {} - état {}'.format(event.type, event.
code, event.value))
14:
15: if __name__ == '__main__':
...
20:     readJoypad(device)
```

Il n'y a plus qu'à exécuter le programme et à utiliser le joypad pour noter les événements que l'on souhaite intercepter :

```
$ sudo python3 joypad.py "/dev/input/event0"
Connected to /dev/input/event0
Type evt : 1 - Code : 308 - état 1
Type evt : 0 - Code : 0 - état 0
Type evt : 1 - Code : 308 - état 0
Type evt : 0 - Code : 0 - état 0
Type evt : 3 - Code : 17 - état -1
Type evt : 0 - Code : 0 - état 0
Type evt : 3 - Code : 17 - état 0
Type evt : 0 - Code : 0 - état 0
...
```

```
$ sudo ./run_keyboard.sh
Connected to /dev/input/event0
Code de touche : 16 - Symbole : KEY_Q - état 1
Code de touche : 16 - Symbole : KEY_Q - état 0
Code de touche : 16 - Symbole : KEY_Q - état 1
Code de touche : 16 - Symbole : KEY_Q - état 0
Code de touche : 17 - Symbole : KEY_W - état 1
Code de touche : 17 - Symbole : KEY_W - état 0
...
```

4. ASSOCIER UNE ACTION À UNE TOUCHE

Pour simuler une action sur le clavier dans le shell, on peut utiliser la commande **xdotool**, qu'il faut installer :

```
$ sudo apt install xdotool
```

Lors de l'installation, vous constaterez que cette commande utilise **libxdo3** qui sera également installé. L'usage de cette commande est simple :

```
$ xdotool key a
a$
```

Comme nous travaillons dans un programme en Python, pour appeler cette commande nous avons deux solutions :

1. Utiliser **os.system** :

```
os.system('xdotool key a')
```

2. Utiliser **subprocess.call** :

```
subprocess.call(['xdotool', 'key', 'a'])
```

Ce n'est pas très « propre ». La première solution que j'ai mise en place utilisait le module **python-libxdo** car ce dernier permet également la manipulation du curseur de la souris, mais jeme suis retrouvé bloqué lorsque j'ai voulu envoyer un signal d'appui sur une touche à partir de son code ou de son nom symbolique.

J'ai donc employé **evdev** où il faut utiliser un objet **Uinput** et surtout, penser à appuyer, puis relâcher la touche et enfin lancer la synchronisation pour que l'appui soit effectif. Cela se traduit par les lignes suivantes :

```
01: import evdev
02: from evdev import ecodes
03: import sys
...
11: def readKeyboard(device, ui):
12:     for event in device.read_loop():
13:         if event.type == ecodes.EV_KEY:
14:             if event.code == 16 and
event.value == 1:
15:                 print('Déclenche un
appui sur <F1>')
16:                 ui.write(ecodes.EV_KEY,
ecodes.KEY_F1, 1)
17:                 ui.write(ecodes.EV_KEY,
ecodes.KEY_F1, 0)
18:                 ui.syn()
19:
20:
21: if __name__ == '__main__':
22:     dev = getDevice()
23:     device = evdev.InputDevice(dev)
24:     print('Connected to', dev)
25:
26:     ui = evdev.UInput()
27:     readKeyboard(device, ui)
```

5. ENREGISTRER LES ACTIONS À RÉALISER SUR UNE TOUCHE

Programmer manuellement les actions à réaliser pour chaque touche du clavier serait bien trop long. Nous allons simplement enregistrer des séquences de touches qui seront associées à une touche et nous placerons ces informations dans un fichier qui sera lu au lancement du programme de gestion du clavier.

Nous supposerons que l'appui sur la touche **<²>** (code **41**) nous fera rentrer en mode « Enregistrement ». La séquence de touches sur lesquelles nous appuierons sera alors mémorisée jusqu'à un nouvel appui sur **<²>**. Il faudra alors appuyer sur une dernière touche qui sera la touche qui rejouera la séquence mémorisée.

Le format du fichier de sauvegarde sera un fichier ini de la forme suivante :

```
[symbole_touche_1]
type_action_1 = [valeur_1, ..., valeur_n]
...
[symbole_touche_m]
type_action_m = [valeur_1, ..., valeur_n]
```

type_action pourra valoir **key_down** (puis par la suite **mouse**, etc. pour l'intégration de la souris ou autres). On pourra également envisager des **key_up**, **key_pressed** et ainsi de suite.

Nous allons créer un nouveau fichier **key.py** qui contiendra des constantes correspondant aux touches du clavier que nous utilisons (un équivalent des **ecodes.KEY_x**) :

```
01: # Constants with key's codes
02: SQUARE = 41
03: RECORDING = SQUARE
```

Il serait intéressant d'avoir un repère visuel sur le clavier indiquant que nous sommes en mode enregistrement. Pour cela nous allons utiliser la led du verrouillage numérique par le biais de la méthode **set_led()** fournie par l'objet **InputDevice** d'**evdev**. Lorsque nous devrons indiquer la touche qui déclenchera l'exécution de la macro, nous éteindrons la led du verrouillage numérique et éclairerons celle du verrouillage majuscule. Au niveau de notre programme **keyboard.py**, tout cela va entraîner de nombreuses modifications que je commenterai au fil du code :

```
001: import evdev
002: from evdev import ecodes, UInput
003: import sys
004: import key
005: from configparser import
ConfigParser
006: import ast
007:
008: def getDevice():
009:     if len(sys.argv) != 2:
010:         print('Syntax : python3
keyboard.py <device>')
011:         exit(1)
012:     return sys.argv[1]
```

Nous retrouvons dans les lignes 1 à 12 les imports des différents modules qui seront utilisés avec les ajouts de **configparser** pour la gestion des fichiers ini et **ast** pour l'évaluation de code Python sous forme de chaînes de caractères (voir plus loin). La fonction **getDevice()** reste inchangée.

```
015: ########
016: ### Leds management
017:
018: def setLock(device, key,
action='off'):
019:     if action == 'off':
020:         device.set_led(key, 0)
021:     elif action == 'on':
022:         device.set_led(key, 1)
023:
024: def setNumLock(device,
action='off'):
025:     setLock(device, ecodes.LED_NUML,
action)
026:
027: def setCapsLock(device,
action='off'):
028:     setLock(device, ecodes.LED_CAPSL,
action)
029:
030: def resetLeds(device):
031:     setNumLock(device, 'off')
032:     setCapsLock(device, 'off')
```

La gestion de l'extinction et de l'éclairage des leds se fait grâce aux fonctions **setNumLock()** et **setCapsLock()** des lignes 24 à 28. Ces deux fonctions permettent une écriture plus élégante que celle employée dans **setLock()** des lignes 18 à 22 qu'elles utilisent pour mener à bien leur action. La fonction **resetLeds()** des lignes 30 à 32 permet d'éteindre toutes les leds et sera appelée au lancement du programme.

```
035: ########
036: ### Recording management
037:
038: def startRecording(device):
039:     print('Start recording macro')
040:     setNumLock(device, 'on')
041:     return True
042:
043: def stopRecording(device, recording_
buffer):
044:     print('Stop recording macro')
045:     print('Buffer:', recording_
buffer)
046:     setNumLock(device, 'off')
047:     setCapsLock(device, 'on')
048:     print('Hit a key to save the
buffer')
049:     return False
050:
051: def saveMacro(key, recording_buffer,
macros):
052:     macros[key] = {'key_down':
recording_buffer}
053:     print(macros)
054:     writeConfig(macros)
```

Viennent ensuite les fonctions permettant d'enregistrer les macros. **startRecording()** (lignes 38 à 41) allume la led de verrouillage numérique et renvoie la valeur **True** pour indiquer un enregistrement en cours. **stopRecording()** (lignes 43 à 49) éteint la led de verrouillage numérique et éclaire celle de verrouillage majuscule pour indiquer que le programme attend l'appui sur une touche pour affecter la macro. La fonction renvoie **False** pour indiquer la fin de l'enregistrement. Enfin, dans les lignes 51 à 54, **saveMacro()** copie la liste de touches présente dans **record_buffering** dans le dictionnaire macros (contenant toutes les macros actives) et appelle la fonction **writeConfig()** pour sauvegarder le contenu de macros dans le fichier de configuration ini.

```
060: ########
061: ### Configuration file management
062:
063: def readConfig(configFile='keyboardrc.ini'):
064:     macros = {}
065:     config = ConfigParser()
066:     config.read(configFile)
067:
068:     for keysym in config.sections():
069:         print(config[keysym])
070:         actions_list = {}
071:         for action in config[keysym]:
072:             actions_list[action] = ast.
literal_eval(config[keysym][action])
073:         macros[keysym] = actions_list
074:         print('Macro pour', keysym, '=>',
macros[keysym])
075:         print(macros)
076:
077:     return macros
078:
079: def writeConfig(macros,
configFile='keyboardrc.ini'):
080:     config = ConfigParser()
081:
082:     for keysym, action in macros.items():
083:         config[keysym] = action
084:
085:     try:
086:         with open(configFile, 'w') as fic:
087:             config.write(fic)
088:     except:
089:         print('Write error on config file',
configFile)
090:         exit(2)
091:
092:     return macros
```

La fonction **readConfig()** (lignes 63 à 77) permet de lire le fichier de configuration (par défaut **keyboardrc.ini**) et de placer son contenu dans le dictionnaire **macros**. Notez que **config.read()** ne renvoie aucun message d'erreur si le fichier est absent (c'est son fonctionnement normal permettant de spécifier plusieurs fichiers de configuration). En ligne 72, **ast.literal_eval()** permet de transformer une chaîne de caractères représentant une liste en liste Python. La fonction **writeConfig()** des lignes 79 à 92 écrit le contenu du dictionnaire **macros** dans le fichier de configuration.

```
095: ########
096: ### Keys management
097:
098: def pressKey(keysym, ui):
099:     ui.write(ecodes.EV_KEY, ecodes.
ecodes[keysym], 1)
100:     ui.write(ecodes.EV_KEY, ecodes.
ecodes[keysym], 0)
101:     ui.syn()
```

```
102:
103: def pressKeys(keysymList, ui):
104:     for keysym in keysymList:
105:         pressKey(keysym, ui)
```

La fonction **pressKey()** des lignes 98 à 101 est un raccourci pour un appui simple sur une touche qui est passée en paramètre (**keysym**). La fonction **pressKeys()** des lignes 103 à 105 permet de donner une liste de touches **keysymList** qui seront envoyées à **pressKey()** pour une simulation d'appui.

```
108: ########
109: ### Keyboard main management
110:
111: def readKeyboard(device, ui):
112:     recording = False
113:     recording_buffer = []
114:     attribute_key = False
115:     macros = readConfig()
116:
117:     for event in device.read_loop():
118:         if event.type == ecodes.EV_KEY:
119:             # Recording management with
key.RECORDING
120:             if event.code == key.RECORDING
and not attribute_key and event.value == 1:
121:                 if not recording:
122:                     recording =
startRecording(device)
123:                 else:
124:                     recording =
stopRecording(device, recording_buffer)
125:                     attribute_key = True
126:             # Keys management
127:             if recording and event.code !=
key.RECORDING and event.value == 1:
128:                 print('Appui sur', ecodes.
KEY[event.code])
129:                 recording_buffer.
append(ecodes.KEY[event.code])
130:             elif attribute_key and event.
code != key.RECORDING and event.value == 1:
131:                 attribute_key = False
132:                 saveMacro(ecodes.
KEY[event.code], recording_buffer, macros)
133:                 print('Macro saved in
<{}>'.format(ecodes.KEY[event.code]))
134:                 setCapsLock(device, 'off')
135:                 recording_buffer = []
136:                 print(macros)
137:             else:
138:                 for keysym, action in
macros.items():
139:                     if event.code ==
ecodes.ecodes[keysym] and event.value == 1:
140:
pressKeys(action['key_down'], ui)
```

Le mécanisme de **readKeyboard()** est globalement le même que celui employé précédemment. **recording** (ligne 112) permet d'indiquer si un enregistrement est en cours, **recording_buffer** (ligne 113) est la liste qui contient un enregistrement de touches en cours, **attribute_key** (ligne 114) indique si le système attend la saisie d'une touche pour lui affecter une macro en cours d'enregistrement (**True**) et enfin **macros** (ligne 115) appelle **readConfig()** pour lire les macros enregistrées dans le fichier de configuration ini. La suite se divise en quatre parties :

1. lignes 119 à 125 : démarrage et arrêt de l'enregistrement des touches d'une macro ;
2. lignes 126 à 129 : enregistrement d'une touche pour la macro courante ;
3. lignes 130 à 136 : affectation d'une touche à la macro courante ;
4. lignes 137 à 140 : pour l'ensemble des macros définies dans **macros**, si l'utilisateur appui sur une touche qui est une clé du dictionnaire alors on appelle **pressKeys()** pour déclencher l'exécution de la macro.

```
143: if __name__ == '__main__':
144:     dev = getDevice()
145:     device = evdev.InputDevice(dev)
146:     resetLeds(device)
147:     print('Connected to', dev)
148:
149:     ui = UInput()
150:     readKeyboard(device, ui)
```

Le programme principal est toujours le même à l'exception de l'extinction des leds en ligne 146 et du remplacement de **Xdo** par **Uinput**.

Vous aurez remarqué la présence de nombreux affichages dans ce code. Ces derniers permettent de s'assurer que tout fonctionne correctement et seront retirés en fin de développement (vous pouvez aussi choisir de les convertir dès maintenant en fichier de log, ce qui peut être une très bonne idée !).

6. PROTÉGER UNE MACRO CONTRE L'EFFACEMENT

La touche **<Tabulation>** (**KEY_TAB**) va nous permettre de protéger une macro enregistrée. La protection sera simplement une option **lock** qui sera associée à une macro dans le dictionnaire **macros** (et donc dans le fichier de configuration). La touche de verrouillage majuscule (**KEY_CAPSLOCK**) permettra de retirer une protection. Voici les modifications de **keyboard.py** :

```
...
59: def protectKey(key, macros):
60:     macros[key]['lock'] = True
61:     print(macros)
62:     writeConfig(macros)
63:
64: def unprotectKey(key, macros):
65:     if 'lock' in macros[key]:
66:         del macros[key]['lock']
67:         print(macros)
68:         writeConfig(macros)
```

Les fonctions **protectKey()** et **unprotectKey()** permettent respectivement l'ajout et le retrait d'une clé **lock** associée à une macro.

```
119: ########
120: ### Keyboard main management
121:
122: def readKeyboard(device, ui):
123:     recording = False
124:     recording_buffer = []
125:     attribute_key = False
126:     protect = False
127:     unprotect = False
128:     macros = readConfig()
129:
130:     for event in device.read_loop():
131:         if event.type == ecodes.EV_KEY:
...
139:             # Recording keys
140:             if recording and event.code !=
key.RECORDING and event.value == 1:
141:                 print('Appui sur', ecodes.
KEY[event.code])
142:                 recording_buffer.
append(ecodes.KEY[event.code])
143:             # Set attribute to a macro
144:             elif attribute_key and event.code
!= key.RECORDING and event.value == 1:
...
151:             # Protect a macro
152:             elif event.code == ecodes.KEY_TAB
and event.value == 1:
153:                 print('Hit a key to indicate
macro to protect')
154:                 setCapsLock(device, 'on')
155:                 protect = True
156:             elif protect and event.value == 1:
157:                 if ecodes.KEY[event.code] in
macros:
158:                     protectKey(ecodes.KEY[event.
code], macros)
159:                     print('Macro <{}> is
protected'.format(ecodes.KEY[event.code]))
160:                 else:
161:                     print('Macro not found')
162:                 setCapsLock(device, 'off')
```

```
163:                    protect = False
164:                # Unprotect a macro
165:                elif event.code == ecodes.KEY_
CAPSLOCK and event.value == 1:
166:                    print('Hit a key to indicate
macro to unprotect')
167:                    setCapsLock(device, 'on')
168:                    unprotect = True
169:                elif unprotect and event.value == 1:
170:                    if ecodes.KEY[event.code] in
macros:
171:                        unprotectKey(ecodes.KEY[event.
code], macros)
172:                        print('Macro <{}> is no more
protected'.format(ecodes.KEY[event.code]))
173:                    else:
174:                        print('Macro not found')
175:                    setCapsLock(device, 'off')
176:                    unprotect = False
177:                # Execute macros
178:                else:
179:                    for keysym, action in macros.
items():
180:                        if event.code == ecodes.
ecodes[keysym] and event.value == 1:
181:                            pressKeys(action['key_
down'], ui)
...
```

Le principe permettant de gérer l'ajout et le retrait d'une protection est le même que celui employé pour l'enregistrement d'une macro grâce à l'usage de deux variables booléennes **protect** et **unprotect**.

7. AJOUTER DES NOTIFICATIONS GRAPHIQUES

Dans l'optique d'un lancement de notre programme au démarrage (donc sans terminal), il pourrait être intéressant d'afficher des notifications. Il existe pour cela plusieurs solutions comme utiliser directement la commande **notify-send** avec un appel système ou bien le module **notify2**, ou encore la **libnotify** de **Gnome**. Le problème avec les deux dernières solutions est que les modules ne sont pas compatibles avec Python 3. Il ne nous reste donc plus que la première solution qui, bien que moins esthétique, présente l'avantage d'être compatible avec tous les environnements de bureau. Nous allons créer une fonction **notifySend()** qui réalisera l'appel système pour nous :

```
001: import evdev
...
007: import subprocess
...
016: ########
017: ### Leds and GUI management
...
035: def notifySend(title, message,
time=1500):
036:     subprocess.Popen(['notify-send',
'-t', str(time), title, message])
...
```

Les différents **print()** de notification à l'utilisateur seront ensuite remplacés par des appels du type :

```
...
055: def saveMacro(key, recording_buffer,
macros):
056:     if key in macros and 'lock' in
macros[key]:
057:         notifySend('Keyboard
Macros', 'Macro on <b>{}</b> is protected.
format(key))
...
```

Notez l'utilisation des tags html **<b>** pour réaliser un affichage en gras. La figure 1 montre un aperçu de la notification en protégeant la macro enregistrée sur la touche **<C>**.

Fig. 1 : Notification de protection de la macro affectée à la touche <C>

8. PORTER LE CODE EN ORIENTÉ OBJET

Notre petit code du début a bien grossi. Il est correctement découpé en fonctions mais une architecture orientée objet serait plus adaptée offrant un gain indéniable en lisibilité et en maintenabilité. Nous sommes partis sur la base de petits tests et nous arrivons à un mini projet. Ce cas de figure n'est finalement pas rare dans le développement d'un programme qui n'est pas planifié et en Python on ne va pas se lancer dans une structure orientée objet pour le moindre test. Il faut donc se rendre compte suffisamment rapidement de l'évolution du code pour passer à un changement de paradigme.

Je profiterai de cette remise en forme du code pour y intégrer un logger qui stockera les messages de débugage. Lors de cette étape, il serait bon de songer également à l'intégration de la documentation du code et des tests unitaires. Le manque de pages se faisant cruellement sentir, ces deux dernières étapes ne seront pas abordées ici et seule la structure du code orienté objet sera présentée (le code intégral étant disponible sur la page **GitHub** du magazine).

Tout ce qui touche à l'interface graphique au sens large se trouvera dans un répertoire **gui** dans lequel nous placerons la gestion des leds du clavier (**KeyboardLeds.py**) et l'affichage des notifications à l'écran (**Notify.py**). La classe principale sera **MacroKeyboard** (fichier **MacroKeyboard.py**) qui utilisera les deux classes précédentes. Enfin, le fichier **keyboard.py** sera adapté de manière à utiliser cette classe. Commençons par la structure des deux fichiers du répertoire **gui**. Voici le code de **Notify.py** :

```
01: import subprocess
02:
03: class Notify:
04:     TIME_FLASH = 500
05:
06:     def __init__(self, title, time =
'1500'):
07:         self.__time = time
08:         self.__title = title
09:
10:     def setTitle(self, title):
11:         self.__title = title
12:
13:     def send(self, message):
14:         subprocess.Popen(['notify-
send', '-t', str(self.__time), self.__
title, message])
15:
16:     def flash(self, message):
17:         subprocess.Popen(['notify-
send', '-t', str(Notify.TIME_FLASH),
self.__title, message])
```

Comme vous pouvez le constater, il s'agit d'une simple réécriture sous forme de méthodes des différentes fonctions que nous avions utilisées. L'attribut de classe **TIME_FLASH** permet de régler simplement en phase de développement le temps d'affichage des messages « flash ».

Voici maintenant le code abrégé de **KeyboardLeds.py** :

```
01: from evdev import ecodes
02:
03: class KeyboardLeds:
04:
05:     def __init__(self, device):
06:         self.__device = device
07:         self.__numLock = 'off'
08:         self.__capsLock = 'off'
09:
10:     def __setLock(self, key, action='off'):
...
16:     def __setNumLock(self, action='off'):
...
20:     def __setCapsLock(self, action='off'):
...
24:     def numLockOn(self):
...
27:     def numLockOff(self):
...
30:     def capsLockOn(self):
...
33:     def capsLockOff(self):
...
36:     def reset(self):
...
```

Cette classe possède trois attributs : **__device** qui est indispensable pour pouvoir manipuler les leds puis **__numLock** et **__capsLock** qui permettent de conserver l'état des leds. Les noms de méthodes sont suffisamment parlant pour indiquer leur action et leur usage rendra le code de **MacroKeyboard.py** plus compréhensible. Voici d'ailleurs son code abrégé :

```
001: import evdev
002: from evdev import ecodes, UInput
003: import key
004: from configparser import ConfigParser
005: import ast
006: from gui.KeyboardLeds import KeyboardLeds
007: from gui.Notify import Notify
008: import logging
009: import logging.config
010:
011: class MacroKeyboard:
012:
013:     def __init__(self, dev,
activeLog=False, configFile='keyboardrc.ini',
logFile='keyboardlogrc.ini'):
014:         self.__device = evdev.InputDevice(dev)
015:
016:         self.leds = KeyboardLeds(self.__
device)
017:         self.leds.reset()
018:
019:         self.notify = Notify('Keyboard
Macros')
020:         self.notify.send('Running and
connected to <b>{}</b>'.format(dev))
021:
022:         self.__activeLog = activeLog
023:         if self.__activeLog:
024:             logging.config.fileConfig(logFile)
025:             self.__logger = logging.
getLogger('root')
026:
```

```
027:          self.__recording_buffer = []
028:          self.__recording = False
029:          self.__attribute_key = False
030:          self.__protect = False
031:          self.__unprotect = False
032:
033:          self.__configFile = configFile
034:          self.__macros = {} # defined in readConfig()
035:          self.readConfig()
036:
037:          self.__ui = UInput()
038:
039:
040:      ########
041:      ### Logging management
042:
043:      def log(self, message, cat='info'):
044:          if self.__activeLog:
046:              if cat == 'info':
047:                  self.__logger.info(message)
048:              elif cat == 'debug':
049:                  self.__logger.debug(message)
050:              elif cat == 'debug':
051:                  self.__logger.debug(message)
052:              elif cat == 'error':
053:                  self.__logger.debug(message)
054:
055:      ########
056:      ### Recording management
057:
058:      def startRecording(self):
...
064:      def stopRecording(self):
065:          self.log('Buffer: {}'.format(self.__
recording_buffer))
...
071:      def saveMacro(self, key):
...
079:      def protectKey(self, key):
...
084:      def unprotectKey(self, key):
...
091:      ########
092:      ### Configuration file management
093:
094:      def readConfig(self):
...
108:      def writeConfig(self):
...
123:      ########
124:      ### Keys management
125:
126:      def pressKey(self, keysym):
...
131:      def pressKeys(self, keysymList):
...
136:      ########
137:      ### Keyboard main management
138:
139:      def read(self):
...
```

On retrouve ici les imports des nouvelles classes dans les lignes 6 à 9. De nombreux attributs apparaissent dans le constructeur et permettront de gérer les leds (lignes 16 et 17), les notifications (lignes 19 et 20), le logger (lignes 22 à 25), etc. Notez d'ailleurs que pour le logger nous utiliserons un fichier de configuration **keyboardlogrc.ini** ayant la structure suivante :

```
[loggers]
keys = root

[handlers]
keys = filehandler

[formatters]
keys = completeFormatter

[logger_root]
level = DEBUG
handlers = filehandler

[handler_filehandler]
class = handlers.RotatingFileHandler
args = ('/var/log/macroKeyboard.log', 2048, 5)
formatter = completeFormatter

[formatter_completeFormatter]
format = %(asctime)s - %(name)s - %(filename)s
- %(funcName)s (%(lineno)d) - %(levelname)s :
%(message)s
datefmt = %d/%m/%Y %H:%M:%S
```

La méthode **log()** des lignes 43 à 53 permet d'utiliser le logger plus simplement en fonction du type de sévérité du message que l'on souhaite employer. Un exemple d'utilisation est donné en ligne 65. Les signatures des autres méthodes permettent de voir les parties du code initial qui ont été transformées pour adopter une structure objet.

Pour finir, **run_keyboard.sh** restant inchangé, voici l'adaptation du code de **MacroKeyboard.py** :

```
01: from MacroKeyboard import MacroKeyboard
02: import sys
03:
04: def getDevice():
05:     if len(sys.argv) != 2:
06:         print('Syntax : python3 keyboard.
py <device>')
07:         exit(1)
08:     return sys.argv[1]
09:
10: if __name__ == '__main__':
11:     dev = getDevice()
12:     macroKeyboard = MacroKeyboard(dev,
activeLog=True)
13:
14:     macroKeyboard.read()
```

On importe la classe **MacroKeyboard** en ligne 1, on crée une instance de cette classe en ligne 12 en précisant que l'on souhaite activer les logs puis on appelle la méthode **read()** (ligne 14) qui va scruter les événements du clavier.

9. AJOUTER LA GESTION DE LA SOURIS

Notre projet est maintenant apte à recevoir des améliorations plus simplement. Pourquoi alors ne pas ajouter la possibilité d'enregistrer des macros permettant de manipuler la souris ? Une nouvelle touche (la touche **<Shift>** de gauche par exemple) lancera l'enregistrement et sauvegardera la position de la souris à chaque clic tout en mémorisant le bouton sur lequel l'utilisateur a appuyé (gauche ou droit). Le module le plus adapté pour manipuler la souris est **PyAutoGUI**, plus simple que **libxdo** :

```
$ sudo pip3 install pyautogui
```

Il reste à utiliser ce module dans **MacroKeyboard.py** :

```
...
010: import pyautogui
011:
012: class MacroKeyboard:
013:     MOUSE_BTN = {key.BUTTON_LEFT : 'left', key.BUTTON_RIGHT : 'right' }
014:
015:     def __init__(self, dev, dev_mouse, activeLog=False, configFile='keyboardrc.ini', logFile='keyboardlogrc.ini'):
016:         self.__device = evdev.InputDevice(dev)
017:         self.__device_mouse = evdev.InputDevice(dev_mouse)
...
035:         self.__recording_mouse_buffer = []
036:         self.__recording_mouse = False
037:
038:         self.__configFile = configFile
039:         self.__macros = {} # defined in readConfig()
040:         self.readConfig()
041:
042:         self.__ui = Uinput()
```

Nous devons enregistrer un second *device* pour la souris (ligne 17). Cela signifie que le programme **run_keyboard.sh** nous transmet l'information (déterminée de la même manière que pour le clavier). J'ai ajouté en ligne 13 un attribut de classe permettant d'établir une correspondance entre le code des boutons souris et leur nom. Cette correspondance utilise deux nouvelles entrées dans le fichier **key.py** (**BUTTON_LEFT = 272** et **BUTTON_RIGHT = 273**). L'enregistrement d'une macro souris suivra le même mode de fonctionnement que pour le clavier et nous retrouvons les mêmes types de variables (lignes 35 et 36).

```
...
095:     ########
096:     ### Mouse recording management
097:
098:     def startMouseRecording(self):
099:         self.log('Start mouse recording macro')
100:         self.notify.send('Start mouse recording macro')
101:         self.leds.numLockOn()
102:         self.__recording_mouse = True
103:         self.mouseRecording()
104:
105:     def mouseRecording(self):
106:         for event in self.__device_mouse.read_loop():
107:             if event.type == ecodes.EV_KEY:
108:                 # Click detection
109:                 if (event.code == ecodes.BTN_MOUSE or event.code == ecodes.BTN_RIGHT) and event.value == 1:
110:                     (mouse_x, mouse_y) = pyautogui.position()
111:                     self.notify.send('Mouse clic on button {} ({}, {})'.format(MacroKeyboard.MOUSE_BTN[event.code], mouse_x, mouse_y))
112:                     self.__recording_mouse_buffer.append((event.code, (mouse_x, mouse_y)))
113:                     self.log(self.__recording_mouse_buffer)
114:                 elif event.code == ecodes.BTN_MIDDLE and event.value == 1:
115:                     self.log('Stop mouse recording macro')
116:                     self.notify.send('Stop mouse recording macro\nHit a key to save the buffer')
117:                     self.leds.numLockOff()
118:                     self.leds.capsLockOn()
119:                     return
120:
121:     def saveMouseMacro(self, key):
122:         if key in self.__macros and 'lock' in self.__macros[key]:
123:             self.notify.send('Macro on <b>{}</b> is protected'.format(key))
124:         else:
125:             self.__macros[key] = {'mouse': self.__recording_mouse_buffer}
126:             self.log('Macros: {}'.format(self.__macros), cat='debug')
127:             self.writeConfig()
```

Là encore, on peut voir que le fonctionnement est très proche de l'enregistrement des macros clavier. Lors du déclenchement de l'enregistrement de la macro souris on lance

une surveillance… de la souris (ligne 106). C'est **event.code** qui contient le code associé au bouton sur lequel on a cliqué et **pyautogui** permet de récupérer la position de la souris (ligne 110). On ajoute ensuite l'action au *buffer* (ligne 112).

```
176:     ########
177:     ### Mouse management
178:
179:     def activeMouseMvt(self, mousemvt):
180:         btn, (mouse_x, mouse_y) = mousemvt
181:         pyautogui.moveTo(mouse_x, mouse_y)
182:         pyautogui.click(button=MacroKeyboard.MOUSE_
BTN[btn])
183:
184:     def activeMouseMvts(self, mousemvtsList):
185:         for mousemvt in mousemvtsList:
186:             self.activeMouseMvt(mousemvt)
```

Pour « rejouer » les actions de la souris nous utilisons encore PyAutoGUI : **moveTo()** pour déplacer la souris (ligne 181) et **click()** pour déclencher un clic où le paramètre **button** indique s'il s'agit du bouton **left** ou **right** (ligne 182).

```
189:     ########
190:     ### Keyboard main management
191:
192:     def read(self):
193:         for event in self.__device.read_loop():
...
240:                 # Mouse recording
241:                 elif event.code == key.RECORDING_
MOUSE and event.value == 1:
242:                     self.startMouseRecording()
243:                 elif self.__recording_mouse and
event.code != key.RECORDING and event.code != key.
RECORDING_MOUSE and event.value == 1:
244:                     self.saveMouseMacro(ecodes.
KEY[event.code])
245:                     self.notify.send('Macro saved in
<b>{}</b>'.format(ecodes.KEY[event.code]))
246:                     self.leds.capsLockOff()
247:                     self.__recording_mouse_buffer =
[]
248:                     self.log('Macros: {}'.
format(self.__macros), cat='debug')
249:                     self.__recording_mouse = False
250:                 # Execute macros
251:                 else:
252:                     for keysym, action in self.__
macros.items():
253:                         if event.code == ecodes.
ecodes[keysym] and event.value == 1:
254:                             if 'key_down' in action:
255:                                 self.
pressKeys(action['key_down'])
256:                             elif 'mouse' in action:
257:                                 self.activeMouseMvts
(action['mouse'])
```

Il a fallu modifier la façon dont les macros sont exécutées puisque la clé **mouse** est apparu dans le fichier **keyboardrc.ini**. On teste donc si l'action à effectuer est du type **key_down** ou **mouse** (lignes 254 à 257).

10. INTÉGRER LE LANCEMENT AUTOMATIQUE DU PROGRAMME AU DÉMARRAGE

Pour que le programme puisse se lancer automatiquement au démarrage il faut effectuer quelques manipulations. Voici les étapes à réaliser en supposant que l'utilisateur soit **user** et que le programme se trouve dans **/home/user/bin** :

1. Modifier **/etc/sudoers** pour lancer le programme avec **sudo** sans demande de mot de passe :

```
user        ALL = (root)
NOPASSWD:    /home/user/
bin/macroKeyboard/run_
keyboard.sh
```

2. Pour Gnome, dans le répertoire personnel de **user**, créer un fichier **/home/user/.config/autostart/keyboard_macro.desktop** :

```
[Desktop Entry]
Type=Application
Exec=sudo /home/user/
bin/macroKeyboard/run_
keyboard.sh
Hidden=false
NoDisplay=false
X-GNOME-Autostart-
enabled=true
Name[fr_FR]=MacroKeyboard
Name=MacroKeyboard
```

CONCLUSION

Nous sommes partis d'un petit code qui n'avait vocation qu'à rester un simple test puis notre code a grossi jusqu'à devenir un projet et il a fallu alors changer de paradigme. Nous n'avons pas adopté une démarche de développement classique ici puisque nous ne savions pas au départ vers quoi nous nous dirigions. Nous obtenons donc un programme « hybride » que je qualifierai de version « R&D » sur laquelle il y a encore un travail de refactorisation et de dépoussiérage à effectuer. Il reste également quelques problèmes mineurs à résoudre tels que la possibilité d'enregistrer des combinaisons de touches (comme **<Contrôle>** + **<c>**), la définition d'un temps d'attente variable entre les clics souris (pour attendre l'ouverture d'une application par exemple) ou encore quelques tests d'erreur.

Le clavier est tout à fait utilisable et avec un peu de temps/patience, en le customisant un minimum il peut devenir très pratique (voir figure 2).

Si vous souhaitez poursuivre le projet ou pour le moins développer des programmes en Python permettant de manipuler une souris, un clavier ou un joypad, je vous encourage à consulter les pages suivantes :

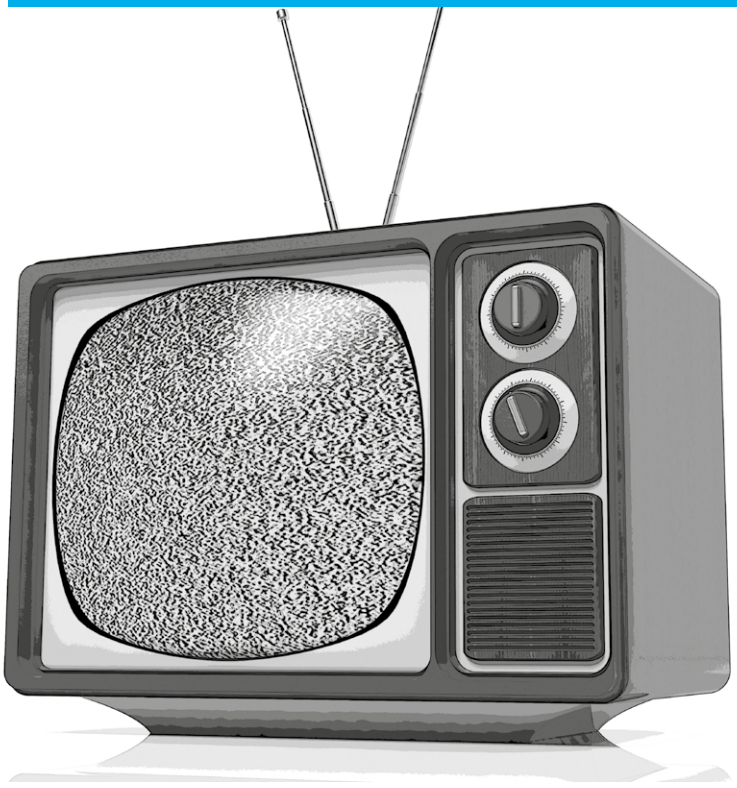
- Documentation d'evdev : https://python-evdev.readthedocs.io/en/latest/index.html
- Documentation de PyAutoGUI concernant la manipulation de la souris : http://pyautogui.readthedocs.io/en/latest/mouse.html ■

Fig. 2 : Clavier customisé pour l'usage du programme

CRÉEZ UNE APPLICATION POUR VOTRE TIZEN TV SAMSUNG

TRISTAN COLOMBO

MOTS-CLÉS : TV CONNECTÉE, TIZEN, HTML, JAVASCRIPT, JQUERY



Même les télévisions sont connectées ou « intelligentes » (smart TV) et proposent des marchés contenant de nombreuses applications. Samsung, avec ses Tizen TV, fournit aux développeurs un SDK permettant le développement d'applications natives ou Web. Je vous propose de tester cet outil en développant un petit jeu.

Le **Tizen TV SDK** permet de développer des applications pour les télévisions connectées de marque **Samsung** fonctionnant sous **Tizen** (les **Tizen TV**). Ce SDK permet aussi bien de travailler sur des applications natives que des applications Web. Pour les applications natives, il faut ajouter le SDK **NaCl** (*Native Client*) et le code sera en **C** ou **C++**. Pour les applications Web, on sera confronté à du classique **html** / **JavaScript**.

Dans cet article, je vous propose de tester le Tizen TV SDK en développant sous forme d'application Web un petit jeu qui sera un Flappy Bird Like.

1. INSTALLATION DU SDK

La première des choses à faire va être d'installer le SDK. Rien de bien compliqué ici, tout a été correctement pensé... et même pour une fois très bien pensé comme vous allez pouvoir le constater :

- Rendez-vous sur https://www.samsungdforum.com/TizenDevtools/SdkDownload et téléchargez le

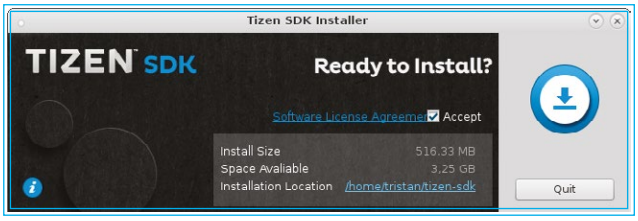


Fig. 1: Fenêtre d'installation du SDK.

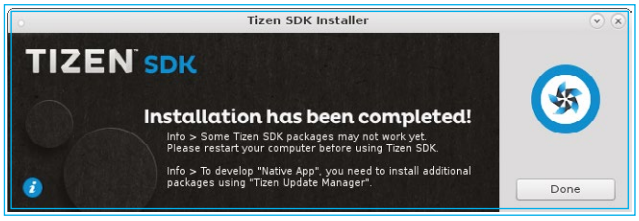


Fig. 2: Message de fin d'installation.

fichier correspondant à votre architecture. Pour moi ce sera **tizen-web-ide_TizenSDK_2.4.0_Rev7_ubuntu-64.bin** ;

• Rendez le fichier exécutable :

```
$ chmod +x tizen-web-ide_TizenSDK_2.4.0_
Rev7_ubuntu-64.bin
```

• Exécutez-le :

```
$ tizen-web-ide_TizenSDK_2.4.0_Rev7_
ubuntu-64.bin
```

Et c'est tout ! Vous aurez accès à une fenêtre vous indiquant les opérations à effectuer et permettant de modifier le chemin d'installation (voir figure 1). Lorsque l'installation sera achevée, vous aurez droit à un message vous indiquant clairement quoi faire (figure 2). Et comme les choses sont vraiment bien faites, suite à cette fenêtre une nouvelle fenêtre apparaîtra vous demandant si vous souhaitez exécuter le **Tizen Update Manager** (permettant notamment l'installation de NaCl, cf figure 2). Nous répondrons ***Oui*** et, en nous rendant dans l'onglet ***All Packages*** puis dans ***Extras >***

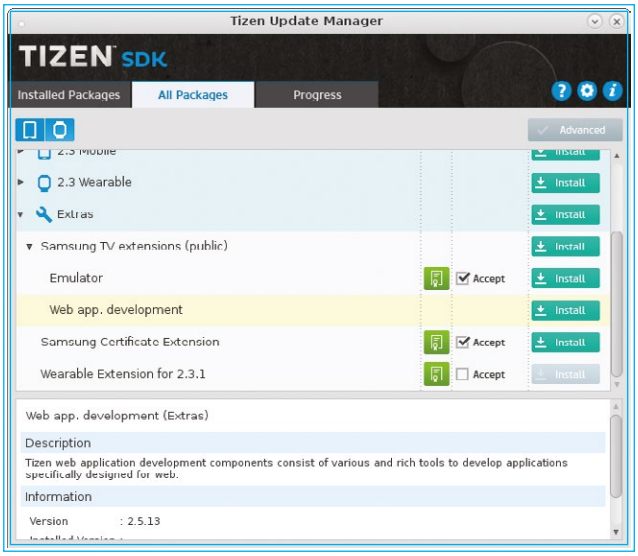


Fig. 3: Tizen Update Manager.

Samsung TV extensions (public), nous installerons ***Samsung Certificate Extension*** (après avoir accepté les licences) comme vous pouvez le voir en figure 3. Les dépendances seront automatiquement résolues, le logiciel en profitera pour mettre à jour ce qui doit l'être et le pourcentage de progression apparaîtra dans le dernier onglet (dans lequel vous aurez accès au détail des installations).

2. DÉCOUVERTE DU SDK

Samsung a pensé aux développeurs et le démarrage d'un projet se fait en utilisant des *templates*, ce qui permet d'avoir immédiatement une structure cohérente. Pour avoir une première vue de cette structure, nous allons créer rapidement un petit programme « Hello world » :

1. Créez un nouveau projet en lançant l'ide (*Integrated Development Environment*) depuis votre répertoire d'installation. L'éditeur n'est rien d'autre qu'un **Eclipse** adapté :

```
$ cd tizen-sdk/ide
$ ./eclipse
```

2. Créez un nouveau projet en cliquant sur ***File*** > ***New*** > ***Other...*** puis ***Tizen*** > ***Tizen Web Project***.

3. Sélectionnez ***Caph3.1-Empty Template for jQuery*** dans ***TV-SAMSUNG-PUBLIC-2.4*** (voir figure 4, page suivante). Ceci nous permettra d'utiliser le *framework* **Caph** de Samsung. Ce *framework* est dédié à tout ce qui touche à l'interface utilisateur : boutons, boîtes de dialogue, etc. **[1]**. Il est possible d'utiliser Caph avec **jQuery** ou **Angular.js**. J'ai choisi ici de partir sur un projet basé sur jQuery.

NOTE

Vous pouvez également choisir de créer une application basique, sans aucun *framework* pré-inclus (***Basic Application*** dans ***TV-2.4*** ou ***Basic Project*** dans ***TV-SAMSUNG-PUBLIC-2.4***).

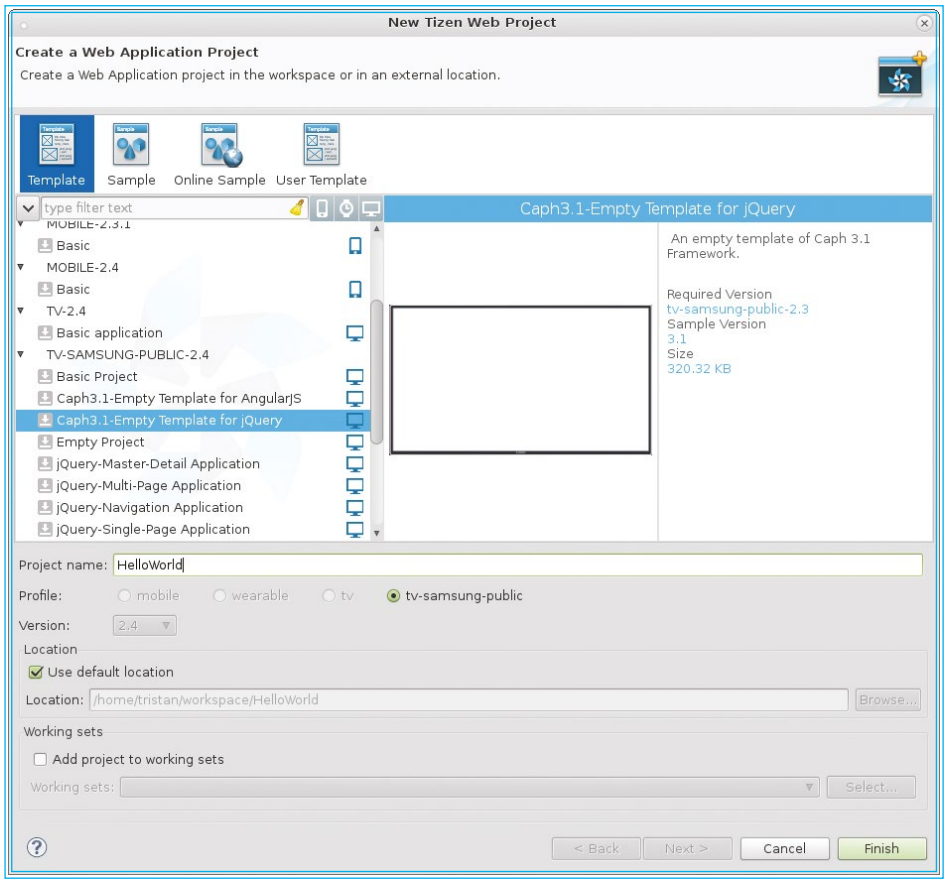


Fig. 4: Choix du modèle de projet Tizen TV.

La structure de notre projet est désormais visible dans la perspective ***Project Explorer*** comme le montre la figure 5. Le fichier **config.xml** contient les informations sur la configuration du projet. Une fois le fichier ouvert, celles-ci sont visibles et modifiables grâce à des onglets ou directement en manipulant le fichier xml (onglet ***Source***). Pour l'instant nous n'y toucherons pas.

Le fichier le plus important est ensuite **index.html** puisque c'est lui qui contient notre application... et pour l'instant il est vide (si ce n'est la structure de base) :

```
01: <!DOCTYPE html>
02: <html>
03: <head>
04:
05:     <!-- include CAPH 3.1.0 default package -->
06:     <link href="lib/caph/3.1.0/caph.min.css" rel="stylesheet" type="text/
stylesheet">
07:
08:     <!-- include jQuery file (you can use jQuery in your environment) -->
09:     <script src="lib/caph/3.1.0/bower_components/jquery/dist/jquery.min.js"
type="text/javascript"></script>
10:
11:     <!-- include hammer js for touch feature (uncomment to enable touch
feature) -->
12:     <!-- <script src="lib/caph/3.1.0/hammerjs/hammer.min.js" type="text/
javascript"></script>  -->
13:
14:     <!-- include the CAPH Package for jquery -->
15:     <script src="lib/caph/3.1.0/caph-jquery.min.js" type="text/javascript"></
script>
16:
17: </head>
18: <body>
19: </body>
20: </html>
```

Pour commencer, nous pouvons corriger ce code en retirant les attributs **type** ! En html5 (donc depuis quelques années déjà), le type par défaut de **<link>** est **"text/css"** et celui de **<script>** est **"text/javascript"**. La balise **<meta>** permettant d'indiquer un encodage utf-8 est absente et il faut l'ajouter si nous souhaitons utiliser des polices accentuées sans manipulations excessives.

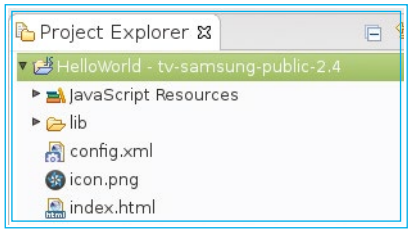


Fig. 5: Structure du projet

Nous allons simplement créer une page (on devrait plutôt parler d'écran ici) bleue. Sur cet écran nous afficherons une boîte de dialogue contenant un petit message et demandant d'appuyer sur le bouton **<Ok>** de la télécommande (eh oui, sur une TV il est plus simple d'utiliser une télécommande...). Cette dernière action provoquera l'effacement de la boîte de dialogue.

Nous commençons par le fichier **index.html** :

```
01: <!DOCTYPE html>
02: <html>
03:   <head>
04:     <meta charset="utf-8" />
05:     <meta name="viewport"
content="width=device-width,
initial-scale=1.0, maximum-
scale=1.0">
06:
07:     <link href="lib/
caph/3.1.0/caph.min.css"
rel="stylesheet" />
08:     <link href="style/hello.
css" rel="stylesheet" />
09:
```

```
10:     <script src="lib/caph/3.1.0/bower_
components/jquery/dist/jquery.min.js"></script>
11:     <script src="lib/caph/3.1.0/caph-jquery.min.
js"></script>
12:
13:     <script src="helloWorld.js"></script>
14:   </head>
15:
16:   <body>
17:     <div id="dialogHello" class="caph-dialog">
18:       <div class="caph-dialog-title"
style="color:#f00;">HELLO WORLD</div>
19:       <div class="caph-dialog-content"
style="color:#b2bec7;">
20:         Un petit programme de test
21:       </div>
22:       <div class="caph-dialog-content"
style="color:#0090ff;">
23:         Appuyez sur le bouton "OK" de votre
télécommande
24:       </div>
25:     </div>
26:   </body>
27: </html>
```

En ligne 4, vous pouvez noter l'ajout de la balise **<meta>** pour spécifier l'encodage utf-8 et en ligne 5 l'ajout de la définition du *viewport* pour ne pas être impacté par des résolutions d'écrans différentes. J'ai ajouté une feuille de style personnalisée dans **style/hello.css** qui indique simplement un arrière-plan bleu pour **<body>** :

```
body {
 background: #71c5cf;
}
```

Le code JavaScript se trouvera dans **helloWorld.js** (voir ligne 12). Pour définir la boîte de dialogue, j'utilise Caph **[2]** :

- une ** ** de classe **caph-dialog** qui va contenir toute la boîte (lignes 16 à 24) ;
- une ** ** de classe **caph-dialog-title** pour le titre (ligne 17) ;
- et des ** ** de classe **caph-dialog-content** pour le contenu de la boîte (lignes 18 à 20 et 21 à 23).

Il faut ensuite écrire le code de **helloWorld.js** pour indiquer le comportement de la boîte de dialogue et intercepter la pression sur le bouton **<Ok>** de la télécommande :

```
01: function configureDialogHello() {
02:     var dialogHello = $('#dialogHello').
caphDialog({
03:         center: true,
04:         focusOption: {
05:     depth: 1
06:    }
07:     });
08:
09:     return dialogHello;
10: };
11:
12: $(document).ready(function () {
13:     var dialogHello = configureDialogHello();
14:     dialogHello.caphDialog('open');
15:
16:    $.caph.focus.activate(function(nearestFocusableFin
derProvider, controllerProvider) {
17:     controllerProvider.addBeforeKeydownHandler
(function(context, controller) {
18:      if (context.event.keyCode === $.caph.focus.
Constant.DEFAULT.KEY_MAP.ENTER) {
19:       dialogHello.caphDialog('close');
20:      return false;
21:     }
22:    });
23:    });
24: });
```

La fonction **configureDialogHello()** permet simplement d'indiquer les options choisies pour l'affichage de la boîte de dialogue (ligne 1 à 10). En valeur de retour, on récupère l'objet « boîte de dialogue » (ligne 9).

La suite du code s'exécute lorsque tous les éléments de la page sont chargés (**$(document).ready()** de la ligne 12). On commence par configurer et afficher la boîte de dialogue par la méthode **caphDialog('open')** (lignes 13 et 14). Dans les lignes 16 à 23 on scrute les événements de la télécommande : en cas d'appui sur le bouton **<Ok>** (**caph.focus.constant.DEFAULT.KEY_MAP.ENTER**) on ferme la boîte de dialogue (ligne 19).

NOTE

Pour connaître les codes associés aux touches de la télécommande vous pouvez simplement insérer la ligne suivante après la ligne 17 :

```
console.log(context.event.keyCode) ;
```

Nous verrons dans la suite comment visualiser cet affichage.

Pour tester votre application, une fois vos fichiers enregistrés, effectuez un clic droit sur le nom de votre projet dans la perspective ***Project Explorer*** et sélectionnez ***Run As*** > ***Tizen Web Simulator Application (Samsung TV)***. Le simulateur va alors s'ouvrir et exécuter votre programme (voir figure 6, page suivante). Notez qu'il est possible de changer de type de télécommande en cliquant sur les signes **<** et **>** qui apparaissent à gauche et à droite de la télécommande lorsque

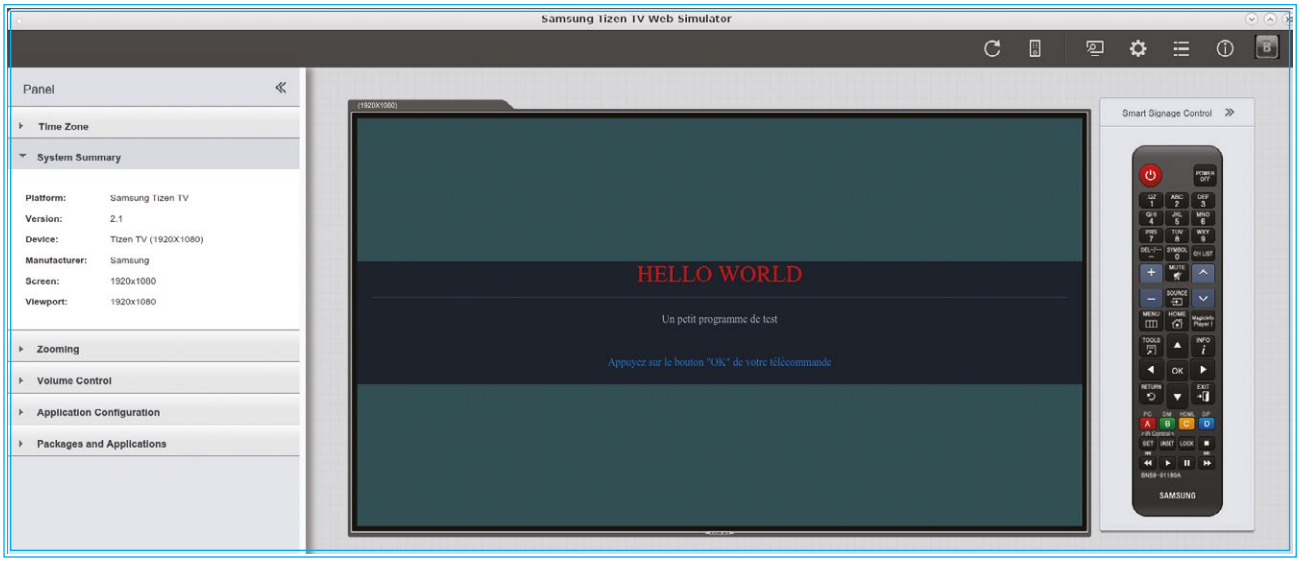


Fig. 6: Test du programme dans le simulateur.

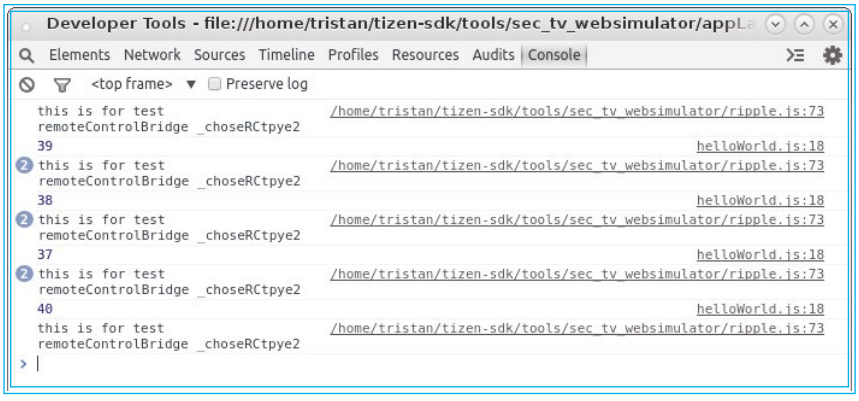


Fig. 7: Console de débuggage.

l'on survole celle-ci avec la souris. En cliquant sur la troisième icône en haut et à droite du simulateur (un écran avec une loupe), vous aurez accès à la console de débuggage. C'est notamment là que vous pourrez voir les messages des `console.log` comme le montre la figure 7. La roue crantée en haut et à droite de la console permet de régler les différents paramètres.

Dans le panneau de gauche du simulateur, vous aurez accès à diverses informations sur votre application. L'onglet ***Application Configuration*** pourra être utile pour corriger des erreurs de configuration (voir figure 8). Ainsi, dans notre exemple, on nous signale que la valeur de la variable `tizen:application` est incorrecte. Elle peut être corrigée dans le fichier `config.xml`, soit de manière graphique dans l'onglet ***Tizen*** proposé par Eclipse, soit directement dans le code source (onglet ***Source***). Pour que la modification soit prise en compte, il vous faudra relancer le simulateur.

NOTE

Vous n'êtes pas obligé de relancer le simulateur à chaque modification : appuyez sur le bouton **<Return>** de la télécommande pour accéder au menu des applications installées.

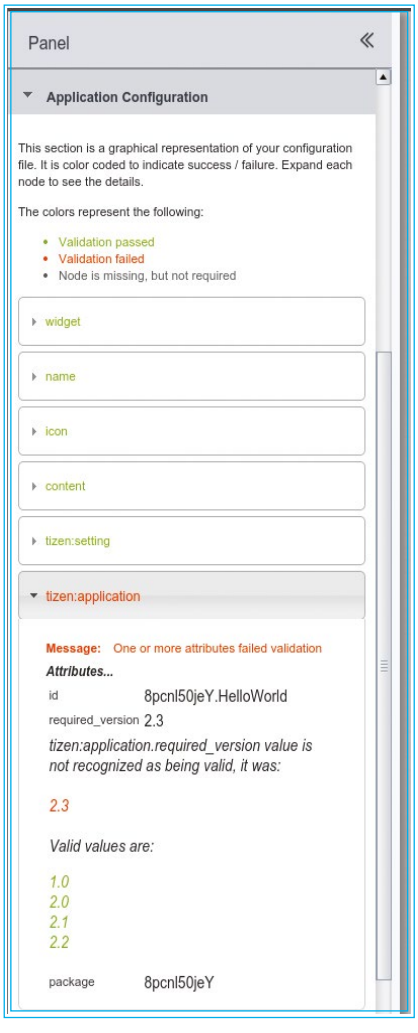


Fig. 8: Onglet Application Configuration du simulateur.

SIMULATEUR OU ÉMULATEUR ?

Le SDK propose un simulateur et un émulateur. L'émulateur sera utile pour les développements natifs mais nécessitera éventuellement une reconfiguration de votre carte graphique si vous n'utilisez pas les drivers propriétaires (sinon l'émulateur refuse de démarrer).

Dans le cas d'un développement Web le simulateur, plus léger et plus rapide, est largement suffisant et moins complexe à mettre en place (il n'y a rien à faire).

3. UN PINGOUIN VOLANT SUR LA BANQUISE

Maintenant que nous avons vu que le développement sous Tizen TV n'était pas plus complexe que le développement d'une application Web, nous allons écrire notre mini Flappy Bird-Like. Pour cela nous allons utiliser le *framework* de jeu **Phaser [3]**, de manière à ne pas avoir à réinventer la roue (éventuellement carrée qui plus est...).

3.1 Installation de Phaser

On commence bien entendu par «installer » Phaser. S'agissant d'un *framework* JavaScript, il suffit de récupérer le code et de placer celui-ci dans le bon répertoire. Nous allons créer un répertoire **`js`** dans notre projet et celui-ci contiendra tous nos fichiers JavaScript.

```
$ git clone https://github.com/
photonstorm/phaser.git
$ cp phaser/build/phaser.min.js ~/
workspace/FlappyPingus/js
$ cp phaser/build/phaser.map ~/workspace/
FlappyPingus/js
```

3.2 La base du jeu : le pingouin volant

Pour commencer, nous allons simplement afficher un pingouin et contrôler son vol en appuyant sur le bouton **<Flèche Haut>**.

3.2.1 Les ressources

Pour notre jeu, nous aurons besoin de ressources graphiques et audio. Pour ne pas obscurcir les explications sur le code, j'ai préféré commencer par organiser ces ressources dans le projet.

Tout d'abord, nous allons avoir besoin d'une image de pingouin volant. J'ai utilisé une image de 100 pixels de côté que j'ai placé dans **`img/pingus_1.png`**. Il est difficile de trouver des *sprites* libres de droits et vraiment jolis ; j'ai donc utilisé ce que j'ai trouvé, n'ayant pas le temps de dessiner moi-même un pingouin...

Ensuite, il est possible d'utiliser ses propres polices de caractères pour afficher des textes à l'écran, ce qui permet d'avoir un effet un peu plus « jeu » qu'avec les traditionnels **Arial**, **Times New Roman**, etc. La documentation de Phaser propose de nombreux exemples très bien documentés **[4]** mais pour l'emploi d'une police externe, éventuellement modifiée par nos soins, il y aura quelques manipulations à effectuer :

- Détenir le fichier ttf correspondant à la police souhaitée. Dans le cadre de cet exemple, j'ai téléchargé la police **Revalia** sur http://www.1001fonts.com/revalia-font.html;
- Phaser va attendre un fichier png et un fichier fnt... Pour convertir le ttf, il suffit de se rendre sur le site http://kvazars.com/littera, d'importer la police en cliquant sur ***Select Font*** dans le menu de gauche, éventuellement de modifier la police (couleur, ombre, etc.), puis de cliquer sur le bouton ***Export*** en haut de la page.

Pour cette police personnalisée, j'ai créé un répertoire **`font`** dans lequel j'ai placé **`font.png`** et **`font.fnt`**.

Enfin, un jeu sans son est un peu triste... dans le répertoire audio j'ai placé un fichier **`jump.wav`** qui sera utilisé à chaque « battement d'ailes » du pingouin. Libre à vous ensuite d'ajouter une musique en boucle durant le jeu, une musique de fin de jeu, un son à la mort du pingouin, etc.

3.2.2 Le code

Tout va se dérouler dans le fichier JavaScript **`main.js`**. Nous allons repartir du fichier, obtenu pour notre « Hello World » en modifiant seulement l'écran d'accueil (modification du texte de **`index.html`**). Je commenterai le code au fur et à mesure :

```
01: function configureDialogHello() {
...
10: };
```

La fonction **`configureDialogHello()`** n'est pas modifiée.

```
12: function gameStart() {
13:     var game;
14:     var gameHeight = window.innerHeight;
15:     var gameWidth = window.innerWidth;
16:     var score = 0;
```

La fonction **gameStart()** va contenir toute la logique du jeu. On définit les principales variables simples dans les lignes 13 à 16 : **game** sera une instance de **Phaser.Game**, **gameHeight** et **gameWidth** sont la hauteur et la longueur de l'écran de jeu (on récupère la taille totale de l'écran) et le score permettra de calculer le score du joueur.

```
17:     var mainState = {
18:         preload: function () {
19:           game.load.bitmapFont('myfont',
'fonts/font.png', 'fonts/font.fnt');
20:             game.load.image('pingus', 'img/
pingus_1.png');
21:             game.load.audio('jump', 'audio/
jump.wav');
22:         },
```

mainState est un tableau associatif où chaque élément est une fonction représentant un état du jeu (on est dans de la « semi-POO »). Dans les lignes 18 à 22, **preload** indique la liste des éléments à charger : la police **font.png** que l'on appellera **myfont** (ligne 19), l'image **pingus_1.png** qui sera nommée **pingus** (ligne 20) et le son **jump.wav** qui sera identifié par **jump**.

```
23:         create: function () {
24:             game.stage.backgroundColor =
'#71c5cf';
25:             this.pingus = game.add.
sprite(gameWidth / 2, gameHeight / 4, 'pingus');
26:
27:             game.physics.startSystem(Phaser.
Physics.ARCADE);
28:             game.physics.arcade.enable(this.
pingus);
29:             this.pingus.body.gravity.y = 1000;
30:
31:             this.jumpSound = game.add.
audio('jump');
32:
33:             var upKey = game.input.keyboard.
addKey(38);
34:             upKey.onDown.add(this.jump, this);
35:         },
```

create permet la création du jeu. On définit la couleur du fond d'écran en ligne 24 (même couleur que dans la css car la zone de jeu Phaser n'applique pas le style), puis on ajoute le *sprite* du pingouin au milieu de l'écran en longueur et aux trois quarts de la hauteur (ligne 25). On active le modèle **ARCADE** du moteur physique **[5]** (ligne 27) et on applique une gravité de **1000** au pingouin (lignes 28 et 29). En ligne 30 on ajoute le son **jump** au jeu et enfin les lignes 33 et 34 indiquent de lancer la fonction **jump()** lors de l'appui sur le bouton **<Flèche Haut>** (code **38**).

```
36:         update: function () {
37:           console.log('('+this.pingus.x+',
'+this.pingus.y+')');
38:             if (this.pingus.y < 0 || this.
pingus.y > gameHeight) {
39:               var looseText = game.add.
bitmapText(gameWidth / 2, gameHeight / 2 - 50,
'myfont', 'Bravo, votre score est de', 64);
40:               looseText.anchor.x = 0.5;
41:               looseText.anchor.y = 0.5;
42:               game.add.text(gameWidth /
2, gameHeight / 2 + 50, score, {font: '150px
Arial'});
43:               game.add.text(gameWidth /
2, gameHeight - 50, 'Appuyez sur OK pour une
nouvelle partie...', {font: '30px Arial'});
44:
45:               var okKey = game.input.keyboard.
addKey($.caph.focus.Constant.DEFAULT.KEY_MAP.
ENTER);
46:                 okKey.onDown.add(this.
restartGame, this);
47:             }
48:         },
```

Comme son nom l'indique, **update** est la fonction de mise à jour. La ligne 37 est une ligne de débuggage permettant d'afficher les coordonnées du pingouin. Si le pingouin sort de l'écran par le bas ou par le haut (ligne 38), on affiche un texte en utilisant la police bitmap **myfont** que nous avons définie (lignes 39 à 41). Pour le score il faudra utiliser la police **Arial** (ligne 42)... car la police que j'ai choisie ne contient pas de chiffre.

```
49:         jump: function () {
50:             this.pingus.body.velocity.y
= -350;
51:             this.jumpSound.play();
52:         },
```

jump sera appelé pour chaque bond de l'oiseau : on modifie la vélocité sur les ordonnées (déplacement vertical) et on joue le son **jumpSound**.

```
53:         restartGame: function () {
54:             game.state.start('main');
55:         },
56:     }
```

restartGame permet de relancer le jeu.

```
58:     game = new Phaser.Game(gameWidth,
gameHeight, Phaser.CANVAS, '');
59:     game.state.add('main', mainState);
60:     game.state.start('main');
61: }
```

Les lignes 58 à 60 permettent de définir l'instance de **Phaser.Game** en indiquant la taille de la fenêtre de jeu puis en associant les fonctions de **mainState** (ligne 54). La ligne 60, déjà vue pour **restartGame**, lance le jeu.

```
65: $(document).ready(function () {
66:     var dialogHello = configureDialogHello();
67:     dialogHello.caphDialog('open');
68:
69:    $.caph.focus.activate(function(nearestFocus
ableFinderProvider, controllerProvider) {
70:     controllerProvider.addBeforeKeydownHandler
(function(context, controller) {
71:       console.log(context.event.keyCode);
72:       if (context.event.keyCode === $.caph.
focus.Constant.DEFAULT.KEY_MAP.ENTER) {
73:         dialogHello.caphDialog('close');
74:         gameStart();
75:       return false;
76:     }
77:    });
78:    });
79: });
```

La seule modification de ce morceau de code est l'appel de la fonction **gameStart()** lors de l'appui sur le bouton **<Ok>**. Vous pouvez maintenant exécuter le programme dans le simulateur et vous serez en mesure de faire « voler » le pingouin jusqu'à ce que vous perdiez (voir figure 9).

3.3 Un pingouin plus « réaliste »

Il faut avouer que notre pingouin, en plus d'être laid, n'est pas très réaliste lorsqu'il essaye de reprendre de l'altitude. Nous allons modifier très légèrement notre code pour ajouter une animation lors de l'appui sur le bouton **<Flèche Haut>**. Pour cela, au lieu d'utiliser un simple sprite nous allons créer une *spritesheet* représentant toutes les étapes de l'animation. Disposant de deux images **pingus_1.png** et **pingus_2.png**, le plus simple est d'utiliser **ImageMagic** pour créer la *spritesheet* **flying_pingus.png** :

```
$ convert pingus_1.png pingus_2.png
pingus_1.png +append flying_pingus.png
```

Pour plus de « réalisme » nous effectuerons une légère rotation sur notre pingouin pour qu'il regarde vers le haut lorsqu'il monte et qu'il regarde vers le bas lorsqu'il descend. Ces modifications se traduisent sur le code de la manière suivante (en rouge) :

```
...
12: function gameStart() {
...
17:     var mainState = {
18:         preload: function () {
19:           game.load.bitmapFont('myfont',
'fonts/font.png', 'fonts/font.fnt');
20:             game.load.
spritesheet('flyingPingus', 'img/flying_pingus.
png', 100, 100, 3);
```

Fig. 9 : Le pingouin est sorti de l'écran : la partie est finie.

```
21:             game.load.audio('jump', 'audio/jump.wav');
22:         },
23:         create: function () {
24:             game.stage.backgroundColor = '#71c5cf';
25:             this.pingus = game.add.sprite(gameWidth / 2,
gameHeight / 4, 'flyingPingus');
26:             this.pingusFly = this.pingus.animations.add('fly');
...
36:         },
37:         update: function () {
...
49:             if (this.pingus.angle < 20)
50:                 this.pingus.angle += 1;
51:         },
52:         jump: function () {
53:             this.pingus.body.velocity.y = -350;
54:             game.add.tween(this.pingus).to({angle: -20}, 100).
start();
55:             this.pingus.animations.play('fly', 30, 1);
56:             this.jumpSound.play();
57:         },
58:         restartGame: function () {
59:             game.state.start('main');
60:         },
61:     }
...
```

Phaser permet d'ajouter cette animation avec un minimum de modifications : en ligne 20 on charge une *spritesheet* où les sprites ont une taille de **100** x **100** pixels et sont au nombre de **3** (donc taille globale de l'image de 300 x 100 pixels). On utilise donc désormais cette *spritesheet* et non plus l'image **pingus_1.png** d'où la modification de la ligne 25 avec le remplacement de **'pingus'** par **'flyingPingus'** (nom de la *spritesheet*). Nous créons également une animation appelée **fly** en utilisant toutes les images de la *spritesheet* (ligne 26), d'où l'absence de paramètres **[6]**. Cette animation est seulement créée et non exécutée donc tant que l'on ne lance pas l'animation, seul le premier sprite de la *spritesheet* sera affiché (on retrouve donc le comportement de notre code précédent). Cette animation est exécutée lors du saut par un appel à la méthode **play()** en ligne 55. On indique ici que l'on désire que l'animation soit jouée à **30** fps et **1** seule fois (pour une animation en boucle, utilisez la valeur **true**).

En ce qui concerne la rotation du sprite, nous modifions l'attribut **angle** qui lui est attaché en créant une nouvelle animation associée à la fonction **jump()** (ligne 54). Nous indiquons ici d'appliquer une rotation de **-20**° en **100** ms.

Enfin, un dernier petit raffinement si vous souhaitez modifier le centre de la rotation du sprite (par défaut il s'agit du coin supérieur gauche) pour faire plus Flappy Bird, vous pouvez ajouter la ligne suivante dans la fonction **create()** pour introduire un décalage de 20% vers la gauche et 50% vers le bas.

```
this.pingus.anchor.
setTo(-0.2, 0.5);
```

3.4 Ajout d'obstacles

Nous allons maintenant ajouter les obstacles qui seront constitués par des barres de glace. Nous reprendrons des

NOTE

Dans la commande **convert**, l'argument **-append** réalise une fusion verticale des images et **+append** une fusion horizontale. Cette technique peut être utilisée pour réaliser simplement n'importe quelle *spritesheet* à partir d'un nombre quelconque d'images. La figure 10 illustre ce procédé dans le cas de notre pingouin volant.

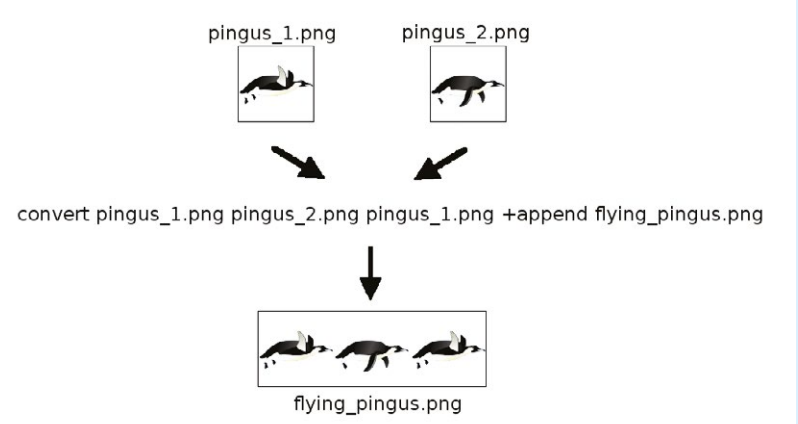


Fig. 10: Création de la spritesheet flying_pingus.png à partir de deux images grâce à ImageMagic.

NOTE

Il est également possible de créer l'animation dans la fonction **create()** puis de l'exécuter ensuite dans **jump()** :

```
...
23:          create: function () {
...
32:              this.jumpSound = game.
add.audio('jump');
33:              this.jumpAnimation =
game.add.tween(this.pingus);
34:              this.jumpAnimation.
to({angle: -20}, 100);
...
38:          },
...
54:          jump: function () {
55:              this.pingus.body.
velocity.y = -350;
56:              //game.add.tween(this.
pingus).to({angle: -20}, 100).start();
57:              this.jumpAnimation.
start();
58:              this.pingus.animations.
play('fly', 30, 1);
...
```

sprites de 100 x 100 pixels que nous empilerons pour former les barres. Nous déterminerons de manière aléatoire un passage dans ces barres. La logique sera donc de créer une barre complète (un mur) et de percer une issue en retirant un certain nombre de blocs (plus le nombre de blocs sera important, plus le jeu sera facile). Les modifications à apporter au code seront encore assez limitées bien que plus importantes que précédemment :

```
...
012: function gameStart() {
...
017:      var dead = false;
018:      var mainState = {
019:          preload: function () {
...
022:              game.load.image('ice', 'img/
ice.png');
023:              game.load.audio('jump',
'audio/jump.wav');
024:          },
```

Nous ajoutons une variable booléenne **dead** en ligne 17 pour être informé de l'état du jeu : si le pingouin est mort alors le vol sera interdit. Dans **preload()**, en ligne 22, nous chargeons l'image **ice.png** qui contient le bloc de glace de 100 x 100 pixels qui sera utilisé pour la construction des murs.

```
025:          create: function () {
026:              score = 0;
027:              dead = false;
...
034:              this.iceWall = game.add.group();
035:              this.nBlocks = Math.floor(gameHeight
/ 100);
...
048:              this.labelScore = game.add.
text(gameWidth / 2, 5, score, {font: '60px Arial',
fill: '#fff'});
049:
050:              this.timer = game.time.events.
loop(2000, this.addIceWall, this);
051:          },
```

Dans la fonction **create()**, le **score** est mis à zéro et la variable **dead** est passée à **false** de manière à pouvoir lancer plusieurs parties consécutives (lignes 26 et 27). Nous créons ensuite un groupe de sprites nommé **iceWall** (ligne 34). Pour l'instant ce groupe ne contient rien, nous le remplirons au cours de la partie.

En ligne 35 nous déterminons le nombre de blocs de glace qu'il est possible d'afficher sur l'écran (combien de blocs de **100** pixels de haut tiennent dans la hauteur **gameHeight**). En ligne 48 on affiche le score dans un nouvel objet **labelScore** en haut et au milieu de l'écran. Enfin, en ligne 50 on crée un timer : toutes les 2 secondes (**2000** ms), on exécute la fonction **addIceWall()** définie dans la suite. Cette fonction affichera un nouveau mur de glace à droite de l'écran et le mur se déplacera vers la gauche.

```
052:          update: function () {
053:              console.log('('+this.pingus.x+',
'+this.pingus.y+')');
054:              if (this.pingus.y < 0 || this.
pingus.y > gameHeight) {
055:                  this.loose();
056:              }
...
059:              game.physics.arcade.overlap(this.
pingus, this.iceWall, this.loose, null, this);
060:          },
```

J'ai externalisé le traitement en cas de défaite (ligne 55). Le code a été déplacé dans une fonction **loose()**. Pour vérifier si le pingouin rentre dans un mur, on utilise le moteur physique : si **pingus** rencontre **iceWall** alors on exécute **loose()**. Notez que le test de collision fonctionne sur nos sprites complets et que le pingouin est une image de 100 x 100 pixels avec fond transparent... Le moteur signalera donc une collision dès que les deux images sont physiquement en contact (même là où nous ne voyons rien). Pour un meilleur rendu, il faudrait utiliser une image dont les dimensions soient

les plus proches possible du sprite qu'elle englobe.

```
061:          jump: function () {
062:           if (! dead) {
063:                   this.pingus.body.velocity.y = -350;
...
067:                   this.jumpSound.play();
068:           }
069:          },
...
```

La fonction **jump()** est désactivée si le pingouin a heurté un mur (**dead** est alors à **true**).

```
073:          addIceBlock: function (x, y) {
074:              var iceBlock = game.add.sprite(x, y,
'ice');
075:              this.iceWall.add(iceBlock);
076:              game.physics.arcade.enable(iceBlock);
077:              iceBlock.body.velocity.x = -200;
078:              iceBlock.checkWorldBounds = true;
079:              iceBlock.outOfBoundsKill = true;
080:          },
```

L'ajout d'un (et un seul) bloc de glace en position **(x, y)** se fait à l'aide de **addIceBlock()** : on crée le sprite en position **(x, y)** et on l'ajoute au groupe **iceWall** (lignes 74 et 75). Dans les lignes 76 et 77 on utilise ensuite le moteur physique sur le bloc créé en lui appliquant une vélocité sur les abscisses de **-200** (scrolling horizontal vers la gauche). Pour finir, on indique au moteur de détruire le mur dès qu'il n'est plus visible à l'écran (lignes 78 et 79).

```
081:          addIceWall: function () {
082:              var hole = Math.floor(Math.random() *
(this.nBlocks - 3)) + 1;
083:              var correction = 0;
084:
085:              for (var i = 0; i < this.nBlocks; i++) {
086:                  if (i != hole && i != hole + 1) {
087:                      this.addIceBlock(gameWidth, i *
100 + 5 + correction);
088:                  }
089:                  if (i == hole + 1) {
090:                    correction = gameHeight - this.
nBlocks * 100 - 15;
091:                  }
092:              }
093:
094:              score += 1;
095:              this.labelScore.text = score;
096:          },
```

La construction des murs de glace est effectuée par la fonction **addIceWall()**. On commence par déterminer de manière aléatoire le numéro de bloc qui sera retiré sachant que le premier (indice **0**) et le dernier doivent rester. On élargira ensuite le trou en supprimant également le bloc **hole + 1** (à vous de modifier le calcul de **hole** en ligne 82 et les tests suivants si vous souhaitez élargir l'ouverture pour rendre le jeu plus simple).

NOTE

Au redémarrage d'une partie, le timer accélère (murs de plus en plus proches) et le jeu ralentit. Mais si après la ligne 110 on redémarre automatiquement le jeu par **this.restartGame()**, tout fonctionne correctement. J'ai effectué différents tests notamment en désactivant **okKey** dans **restartGame()** (en ayant passé la variable en global), rien ne change et le simulateur ne ralentit pas toujours sur le même mur. Défaut du simulateur ou problème du code ?

Dans les lignes 85 à 92 on effectue simplement une boucle sur le nombre de blocs constituant le mur et on supprime (on plutôt on n'ajoute pas) les blocs d'indice **hole** et **hole + 1**. Les blocs se situant après **hole + 1** (donc après le trou) seront un peu plus éloignés grâce au calcul de la **correction** (**gameHeight** a peu de chances d'être un multiple **100** et donc on agrandit le trou de manière à ce que le dernier bloc du mur se trouve en bas de l'écran). Enfin, dans les lignes 94 à 95 on gère le score et on l'affiche. Ici le score est incrémenté d'une unité à l'apparition d'un nouveau mur. Pour incrémenter le score à chaque franchissement d'un mur, il faudrait ajouter une fonction scrutant la position du pingouin par rapport aux murs.

```
097:          loose: function () {
098:           dead = true;
099:
100:          game.time.events.remove(this.timer);
101:            this.iceWall.forEach(function (p)
{p.body.velocity.x = 0;}, this);
102:
103:          var looseText = game.add.
bitmapText(gameWidth / 2, gameHeight / 2 - 50,
'myfont', 'Bravo, votre score est de', 64);
...
110:            okKey.onDown.add(this.restartGame,
this);
111:          },
112:      }
...
```

La fonction **loose()** reprend l'affichage du message de fin de partie et on ajoute la mise à **true** de la variable **dead** en ligne 98, la suppression du timer en ligne 100 et l'immobilisation des murs (passage de la vélocité sur les abscisses à **0**) en ligne 101.

Le résultat final est visible en figure 11.

À PROPOS DU SIMULATEUR

Le simulateur a parfois un comportement complètement irrationnel (souvent au démarrage et après quelques heures d'utilisation). Avant de vous arracher les cheveux sur un code qui se met à dysfonctionner sans raison, je vous recommande fortement de commencer par fermer le simulateur puis de le relancer !

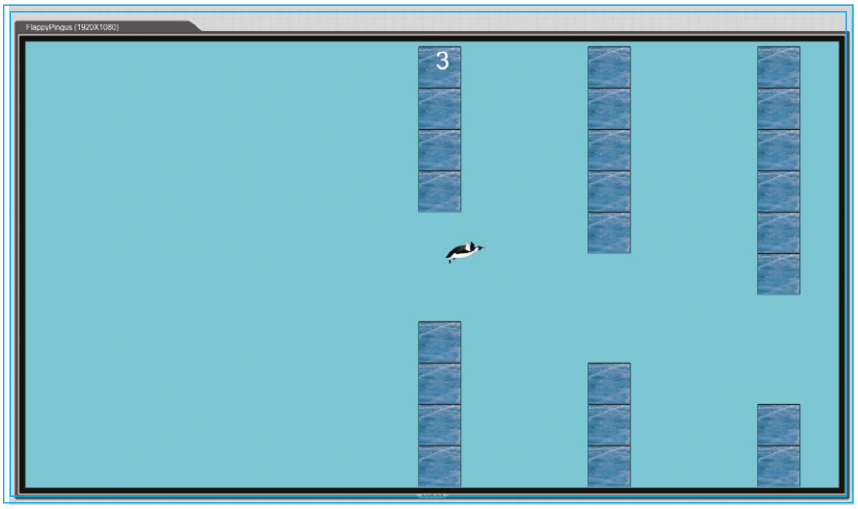


Fig. 11: Vesion finale de Flying Pingus.

NOTE

Un problème étrange est apparu lors de la première tentative de mise à jour du *firmware* : l'entrée ***Mise à jour du logiciel*** était désactivée. Pour la réactiver, j'ai modifié la langue du système dans ***Paramètres*** > ***Système*** > ***Paramètres Expert*** > ***Langue*** (passage du français à l'anglais). J'avoue que la manipulation était tout à fait fortuite. Ce qui est « amusant », c'est que maintenant c'est le menu de changement de langue qui est désactivé et qui affiche « Cette fonction n'est pas disponible ».

4. TESTER L'APPLICATION SUR LA TV

La procédure permettant de tester l'application directement sur la télévision est assez lourde. Elle est détaillée dans cette partie.

4.1 Passage en mode développeur

Pour commencer, vous devez passer en mode développeur. Pour cela, allez dans le menu ***APPS*** et tapez le code **12345** (saisie silencieuse). Une nouvelle fenêtre apparaîtra : validez le choix ***Activé*** de manière à obtenir l'affichage ***Developer mode : On***. Indiquez ensuite l'adresse IP de l'ordinateur relié à la TV et validez le tout (chez moi ça ne sert strictement à rien, car l'adresse n'est pas enregistrée...). Vous devrez rebooter pour que les modifications soient prises en compte (il faut débrancher la prise électrique).

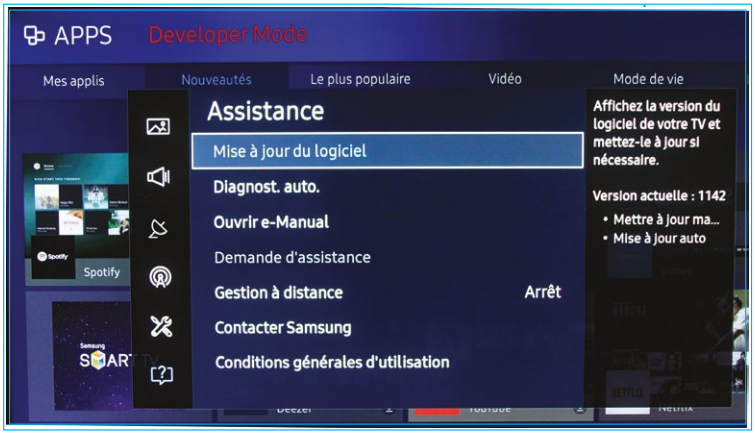


Fig. 12 : Mise à jour du firmware.

4.2 Mise à jour du firmware

D'après la documentation **[7]**, vous devez vous assurer que la version du *firmware* installé sur votre TV est au moins égale à **1400**. Pour cela, rendez-vous dans le menu ***Paramètres*** > ***Plus...*** > ***Assistance*** > ***Mise à jour du logiciel*** (voir figure 12). Chez moi, la version est **1142**... et il n'y a pas de mise à jour disponible (même en passant par un téléchargement chez Samsung **[8]**). On va donc tester comme ça.

Récupérez ensuite l'adresse IP de votre TV en vous rendant dans ***Paramètres*** > ***Réseau*** > ***État du réseau*** > ***Param. IP***. Dans la suite j'utiliserai **192.168.0.13** (IP obtenue dans le champ ***Adresse IP***).

4.3 Connexion avec la TV depuis Eclipse

Dans l'éditeur Eclipse, dans la vue ***Connection Explorer***, cliquez sur l'icône ***Remote Device Manager*** (voir figure 13). Dans la fenêtre qui apparaît (voir figure 14, page suivante), cliquez sur ***New*** puis donnez un nom à la connexion et indiquez l'adresse IP obtenue précédemment (le port par défaut est **26101**).

Cliquez ensuite sur la connexion que vous venez de créer (toujours dans la fenêtre Remote

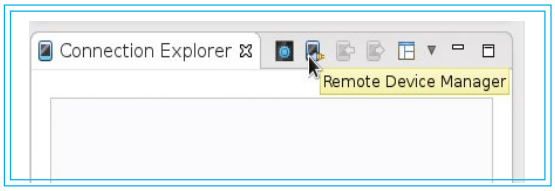


Fig. 13 : Lancement du Remote Device Manager.

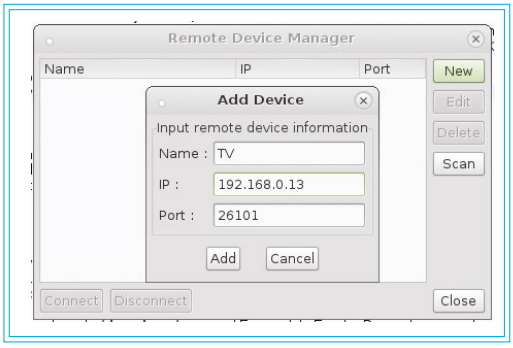


Fig. 14 : Création de la connexion.

Device Manager) et établissez la connexion en cliquant sur le bouton ***Connect***. Une nouvelle ligne apparaîtra dans la vue ***Connection Explorer*** comme vous pouvez le voir en figure 15.

Fig. 15 : Connexion établie avec la TV.

4.4 Création d'un certificat

Toutes les applications doivent être dotées d'un certificat valide. Pour pouvoir installer notre application sur la TV nous allons donc devoir en créer un.

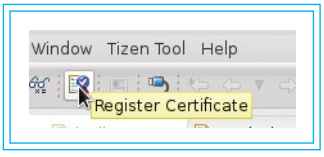


Fig. 16 : Création d'un certificat.

Cliquez sur l'icône ***Register Certificate*** dans la barre d'outils (voir figure 16). Dans la nouvelle fenêtre qui apparaît, sélectionnez ***TV*** en tant que *Device Type*. Donnez ensuite un nom à votre certificat en cliquant sur le bouton ***New*** associé à l'étape 2 *Create Security Profile*. Dans l'étape 3 *Create an author Certificate*, cliquez sur ***Create new certificate*** puis complétez les différents champs (nom et mot de passe obligatoires). Validez en cliquant sur ***Request***.

On vous demandera alors de signer avec un compte Samsung. Cliquez sur ***Create New Account*** dans la fenêtre qui s'affiche (si vous n'avez pas de compte, bien entendu). Toute la procédure est intégrée dans Eclipse (voir figure 17), ce qui est appréciable au vu du nombre d'opérations à effectuer depuis que nous avons commencé... Validez ensuite les conditions d'utilisation (attention, certaines cases ne concernent que du mailing...).

Vous reviendrez ensuite sur la fenêtre de création du « certificat auteur ». Cliquez à nouveau sur ***Request*** puis remplissez cette fois votre ID (adresse mail) et mot de passe avant de cliquer sur ***Sign In***. Un message vous indiquera que le certificat a été placé dans le répertoire **`tizen-sdk-data/keystore/nom_certificat`** (dans l'exemple utilisé ici, ayant nommé mon certificat TV_GLMF, celui-ci se trouve dans **`tizen-sdk-data/keystore/TV_GLMF`**).

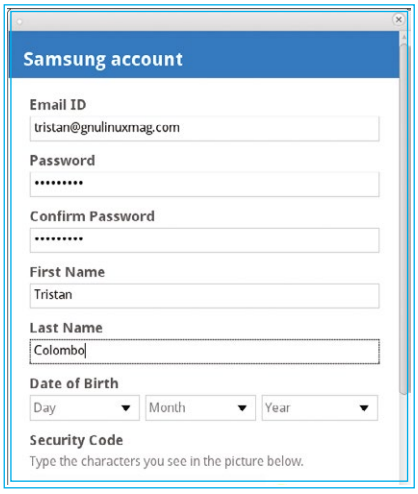


Fig. 17 : Création d'un compte Samsung depuis Eclipse.

Passez ensuite à la quatrième et dernière étape avec la création du « certificat distributeur ». Cliquez sur ***Create new certificate*** pour accéder à la fenêtre de création du certificat présentée en figure 18, page suivante. Indiquez un mot de passe et ajoutez un périphérique en cliquant sur le bouton ***Add*** (noté ***Added*** sur la figure, car déjà ajouté et entouré en rouge). Cliquez sur ***Request*** puis validez à nouveau avec votre compte Samsung.

Achevez la procédure en cliquant sur le bouton ***Ok*** de la fenêtre *Samsung Tizen Certificate*.

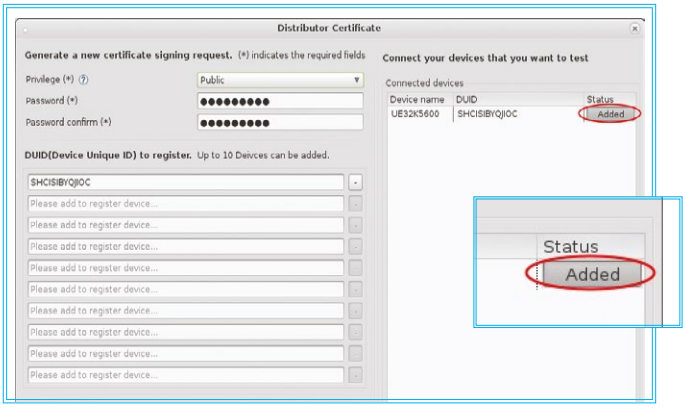


Fig. 18 : Création du « certificat distributeur ».

4.5 AUTORISER L'INSTALLATION D'APPLICATIONS SUR LA TV

Retournez maintenant dans la vue ***Connection Explorer*** et effectuez un clic droit sur la connexion créée en 4.3 pour sélectionner ***Permit to install applications*** (voir figure 19).

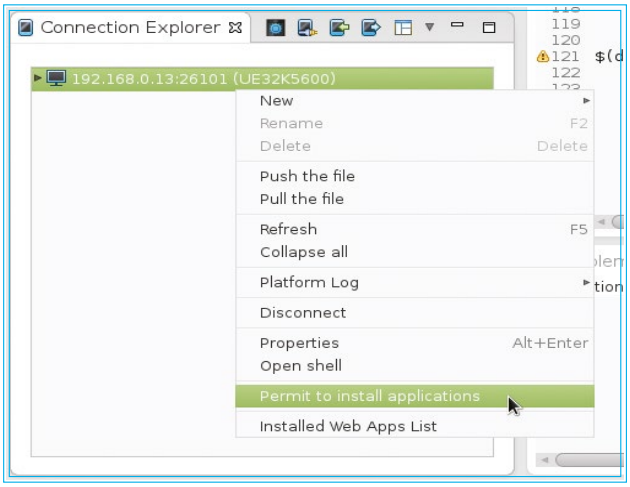


Fig. 19 : Autorisation d'installation d'applications sur UE32K5600 (la télé...).

Vous devriez avoir le message suivant qui s'affiche : « Succeeded to upload a certificate ».

4.6 LANCEZ VOTRE APPLICATION SUR LA TV (ENFIN...)

Effectuez un clic droit sur le répertoire de votre projet et sélectionnez ***Run As*** > ***Tizen Web Application***... et là le miracle s'accomplit enfin : votre application est sur la télévision (voir figure 20, page suivante) ! Celle-ci sera ensuite accessible depuis le menu ***APPS*** > ***Mes applis***.

Vous noterez qu'il n'était pas inutile de tester l'application sur la TV, car celle-ci ne se comporte pas exactement de la même manière que l'émulateur : il reste quelques bugs à corriger.

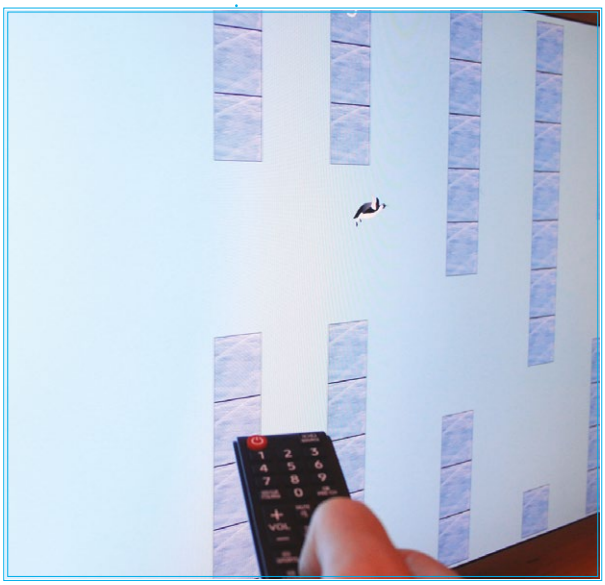


Fig. 20 : Flappy Pingus s'exécute enfin sur la TV !

CONCLUSION

Cet article a été l'occasion de deux découvertes : le SDK Tizen TV et le *framework* Phaser. Vous avez pu voir que transférer le programme sur une véritable Tizen TV n'était pas de tout repos... mais ce n'est rien à côté de la procédure permettant de distribuer une application **[9]** ! Vous comprendrez sans doute pourquoi il n'y a pas beaucoup d'applications disponibles sur le « market »...

Mise à jour du SDK Tizen

Pour mettre à jour votre SDK, rendez-vous dans le répertoire d'installation, dans le répertoire `update-manager` et lancez la commande :

```
$ ./update-manager.bin
```

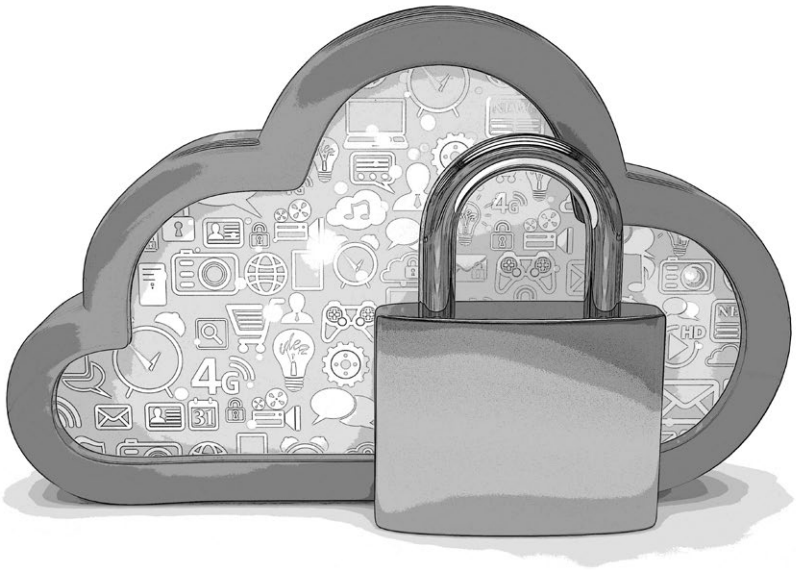
On peut noter que le SDK a été particulièrement soigné et qu'il n'est pas désagréable à utiliser sous Eclipse (rebaptisé Tizen SDK), même lorsque l'on est adepte de **Vim**. Le simulateur est pratique, mais pas forcément très stable au vu des tests réalisés et surtout une application testée sur l'émulateur n'aura pas nécessairement le même comportement sur la TV, ce qui est vraiment pénalisant. D'un point de vue technique, je me suis orienté sur un développement web et le *framework* Phaser, que je n'avais jamais utilisé, s'est avéré particulièrement compréhensible, bien documenté et efficace. Il y a peut-être encore des choses à creuser de ce côté...

Comme vous avez pu le lire, je trouve que le travail de Samsung au niveau du SDK a été particulièrement bien pensé (des outils en ligne de commandes sont même présents dans **`tools/ide/bin`**). Par contre, il faut reconnaître que le changement de périphérique (ie passage de l'ordinateur à la TV pour les tests ou la distribution) est une véritable machine à gaz ! Si l'on rajoute à cela des incohérences de la documentation (sur le numéro de version du *firmware* par exemple) et des entrées du menu de la TV qui s'activent/désactivent sans lien direct, on peut se dire que l'on se trouve face à un produit intéressant... mais qui manque encore un peu de maturité ! ■

LE CSRF DÉMYSTIFIÉ ET BLOQUÉ

SÉBASTIEN GIORIA
[OWASP France Leader & Evangéliste]

MOTS-CLÉS : WEB, CSRF, MOD_CSRF, CSRFGUARD, DJANGO, RUBY ON RAIL



Dans cet article, nous allons détailler la vulnérabilité appelée CSRF qui est l'une des plaies les plus courantes que l'on remonte dans les audits de sécurité tout en proposant diverses solutions en fonction du contexte de l'application Web que vous utilisez.

Le *Cross Site Request Forgery* (CSRF), encore appelé XSRF en anglais, est une vulnérabilité courante dans les applications Web (principalement mais cela peut s'appliquer aussi aux applications mobiles, client-serveur, etc.). Toutes les études que nous pouvons trouver en parlent et la classent parmi les plus courantes :

- Le Top10 **OWASP** la classe à la 8ème position **[1]** ;
- Le **Web Application Security Consortium** la classe 9ème position sur 49 vulnérabilités **[2]** ;
- La dernière étude sur les sites Web de la société **WaveStoneFr** (en 2016) la classe en 3ème position des failles découvertes **[3]**.

Pourtant il peut être très simple dans certains cas de corriger cette vulnérabilité. Plusieurs solutions s'offrent en fonction du contexte. Cela peut être parfois même réglé par un simple clic dans une boîte de dialogue. Nous allons voir dans cet article les différentes solutions qui s'offrent à nous que cela soit de manière « infrastructure ou configuration » ou de manière « dans le gras du code ».

1. CSRF KÉSAKO ?

CSRF: *Cross Site Request Forgery*. En bon français on traduit cela par « Falsification de requêtes intersites ».

Si nous analysons cette traduction, il est facile de comprendre en partie le mécanisme de cette vulnérabilité. Il s'agit donc depuis un site http://www.pirate.com de pouvoir exécuter une requête sur le site http://www.bank.com (par exemple).

Ceci étant dit, je ne suis pas sûr que tout le monde ait tout compris par cette description. Il est à mon sens nécessaire de décomposer l'attaque pour que tout le monde suive correctement. Un bon exemple est nécessaire. Prenons donc le principe suivant :

- Je suis un Internaute, j'ai accès au site de ma banque ;

- Je suis un Internaute qui surfe sur d'autres sites ;
- Je suis un Pirate et je connais le site de la banque de l'Internaute ;
- Je suis un pirate et je peux avoir accès à un forum ou un site Web où va venir l'Internaute.

Première Étape :

Je suis un internaute et j'ai donc accès au site de ma banque. Ce matin, j'ai décidé d'aller voir le solde de mon compte pour savoir si je peux acheter des **Raspberry Pi** (modèle que vous voulez).

Je me rends sur le site de ma banque, je rentre mes identifiants et je consulte mon solde. Le serveur Web de ma banque va donc m'envoyer un élément de suivi de ma session applicative (la plupart du temps un cookie).

Seconde étape :

J'ai beaucoup de chance, j'ai 42 000€ sur mon compte... Ça va me permettre d'acheter pas mal de Raspberry Pi.

Je surfe et trouve sur des forums spécialisés des discussions sur des prix canon pour des Raspberry Pi. Je me rends donc sur ces sites et consulte les articles.

Troisième étape :

Le pirate qui a réussi à m'attirer sur ses articles du forum a bien sûr placé un élément dans le forum/la page Web qui est de la forme :

```
<img src="https://www.mabanque.com/424242/virement/
emettre?cptE=42&cptD=4242&montant=4200" />
```

Pas de chance, automatiquement cette url est chargée par le navigateur (et l'image affiche une jolie croix)... Et là c'est potentiellement le drame !

1. J'ai de la chance : ma banque est sécurisée et me demande un élément supplémentaire pour passer le virement (un code SMS, une case de carte de code, etc.). Ouf... le virement n'est pas parti !
2. Je n'ai pas de chance (et là je vous recommande de changer de banque) : ma banque ne demande pas d'élément supplémentaire pour valider le virement, et le virement part...

Voilà, le CSRF c'est aussi simple que cela.

2. CSRF OK, MAIS POURQUOI CELA FONCTIONNE ?

Revenons au fonctionnement typique du navigateur, de HTTP et de l'HTML.

Une page HTML est constituée de différents éléments, qui peuvent être :

1. « Internes » à la page ;
2. « Externes » à la page mais situés sur le même serveur ;
3. « Externes » à la page et situés sur un autre serveur.

Sur le point 1, cela peut présenter des risques, mais cela n'est pas le point que nous allons expliciter. Si le contenu est externe à la page, le navigateur qui interprète le code HTML effectue, surtout sans aucun contrôle, une requête HTTP à destination du serveur qui contient l'élément (dans le cas du vol explicité, vers www.mabanque.com) et éventuellement interprète le résultat retourné. Mais le pire dans tout cela, c'est que le navigateur va envoyer vers le serveur externe l'ensemble des éléments dits « identifiants » à ce dernier. Et c'est ce dernier point qui va permettre dans le scénario évoqué de passer le virement.

Un identifiant qu'est-ce que cela peut être ? Eh bien dans le monde HTTP, un identifiant cela peut être :

- le *cookie* de session ;
- un *cookie* quelconque (pourvu qu'il soit rattaché au serveur) ;
- un *login*/*password* HTTP (le fameux *login*/*password* encodé en base64) ;
- un certificat SSL/TLS d'authentification.

Dans le cas du scénario évoqué, il y a fort à parier que l'identifiant sera le cookie de session.

L'utilisateur ayant été identifié précédemment, le cookie de session sera envoyé automatiquement pour éviter que l'utilisateur soit obligé de retaper son *login* et son mot de passe lors de son prochain passage sur le site Web. Et donc aussi dans le scénario que nous avons évoqué, ce qui permettra au virement de passer.

3. QUELLES SONT LES SOLUTIONS À NOTRE DISPOSITION ?

Maintenant que nous avons vu pourquoi le CSRF existe et comment il fonctionne, nous allons essayer de le corriger.

Empêcher un CSRF est parfois très simple, mais peut dans certains cas nécessiter de profonds changements dans votre application. Dans la suite, nous n'évoquerons que les cas classiques, les cas « spécifiques » devront être réfléchis par l'architecte de votre application.

Tout d'abord il faut savoir que le CSRF peut se corriger dans le code ou via des *frameworks*, voire directement par le serveur application. Nous allons essayer dans la suite de décrire sur les serveurs les plus courants ainsi que sur les langages les plus utilisés qu'elles sont les solutions disponibles.

3.1 Les solutions dans la partie infrastructure (serveurs Web, serveurs Applicatifs)

3.1.1 Sur Apache

Il existe un module **Apache** dédié au contre des CSRF. Il s'agit de **mod_csrf** (http://mod-csrf.sourceforge.net). La documentation étant assez bien faite et claire, je ne peux que vous conseiller d'aller regarder le lien où l'on trouve le module.

3.1.2 La configuration des filtres de TOMCAT

Certains serveurs application **Java** sont livrés avec une protection basique anti-CSRF, c'est le cas de **TOMCAT** (à travers le filtre de protection CSRF intégrée **[4]**). Le fonctionnement est tout simple :

1. Définition du filtre dans le **web.xml** :

```
<filter>
    <filter-name>CsrfFilter</filter-name>
    <filter-class>org.apache.catalina.filters.
CsrfPreventionFilter</filter-class>
    <init-param>
      <param-name>entryPoints</param-name>
      <param-value>/apasbesoindanticsrf</param-value>
    </init-param>
  </filter>
```

2. *Mapping* des URLs qui auront besoin d'anti-CSRF (ici toutes en dehors de **/apasbesoindanticsrf**) :

```
<filter-mapping>
    <filter-name>CsrfFilter</filter-name>
    <url-pattern>/*</url-pattern>
  </filter-mapping>
```

TOMCAT renverra ensuite des codes HTTP 403 dans le cas où le jeton n'est pas bon.

3.2 Les solutions dans la partie code...

Ici la solution va être propre à chacun des langages. Nous allons regarder quelques langages courants (et éventuellement les *frameworks* que l'on peut utiliser) par ordre alphabétique (Java, PHP, Python, **Ruby**).

3.2.1 Aidons Duke avec la libellule...

En Java, une seule bibliothèque est à vous recommander. Utilisez l'**OWASP CSRFGuard [5]**. Elle fonctionne comme un filtre Java et va donc s'installer dans le **web.xml** de manière super simple. Elle est libre, mise à jour régulièrement et ne nécessite pas d'autres module Java pour fonctionner.

1. Déclarer le **CSRFGuard** comme un **Listener** dans le **web.xml** :

```
 <listener>
            <listener-class>org.owasp.
csrfguard.CsrfGuardServletContextListener
</listener-class>
       </listener>
       <listener>
            <listener-class>org.owasp.
csrfguard.CsrfGuardHttpSessionListener</
listener-class>
       </listener>
       <context-param>
             <param-name>Owasp.CsrfGuard.
Config</param-name>
             <param-value>WEB-INF/Owasp.
CsrfGuard.properties</param-value>
       </context-param>
       <context-param>
             <param-name>Owasp.CsrfGuard.
Config.Print</param-name>
             <param-value>true</param-value>
       </context-param>
```

2. Activer sur les urls le **CSRFGuard** :

```
<filter>
<filter-name>CSRFGuard</filter-name>
<filter-class>org.owasp.csrfguard.
CsrfGuardFilter</filter-class>
</filter>
<filter-mapping>
<filter-name>CSRFGuard</filter-name>
<url-pattern>/*</url-pattern>
</filter-mapping>
```

3. Configurer les propriétés que vous souhaitez (temps de vie du jeton, manière d'injecter le jeton, etc.).

Et c'est fini, votre application Java est protégée.

3.2.2 Bien sûr, avec des frameworks on arrive aussi a se protéger facilement

Si vous utilisez un *framework* du marché. Certains ont des solutions dans la configuration. C'est le cas de **Spring [6]**. Depuis la version 3.2, le CSRF est activé par défaut.

Il faudra ensuite s'assurer que dans tous les formulaires le jeton est bien présent, par exemple :

```
<form method="post" action="/do/something">
  <sec:csrfInput />
  Name:<br />
  <input type="text" name="name" />
  ...
</form>
```

3.2.3 Pour les éléPHPants, cela peut être simple, ou pas...

Au niveau de PHP, plusieurs solutions s'offrent à vous en fonction des *frameworks* utilisés. Si votre développement est basé sur **Zend**, **Symfony**, ou **Cake** alors la solution consiste à correctement le configurer pour activer l'envoi automatique des jetons anti-CSRF.

Pour Zend, il faut ajouter les modules qui vont bien dans le code et créer le jeton qui va placer un élément caché dans le formulaire :

```
use Zend\Form\Element;
use Zend\Form\Form;

$csrf = new Element\Csrf('csrf');

$form = new Form('my-form');
$form->add($csrf);
```

Dans Symfony, par défaut, la protection est activée si vous utilisez des formulaires **[7]** (cf http://symfony.com/doc/current/reference/configuration/framework.html) ! Néanmoins si vous souhaitez le modifier (*timing*, nom ou autre) c'est toujours possible pour chacun des formulaires via la fonction **configureOptions** **[8]** (http://symfony.com/doc/current/form/csrf_protection.html).

Néanmoins, vous souhaitez parfois activer cette protection sans pour autant activer le module de formulaire (cas des API de type REST par exemple). Dans ce cas rien de plus simple !

```
public function myFonction()
{
    if ($this->isCsrfTokenValid('token_
id', $submittedToken)) {
        echo 'valid';
    }
}
```

Au niveau de Cake, il faut charger le composant CSRF qui se chargera lui de faire le travail via un *cookie* :

```
public function initialize()
{
    parent::initialize();
    $this->loadComponent('Csrf');
}
```

Par contre il faudra ensuite toujours utiliser le composant **FormHelper** si vous souhaitez correctement mettre en œuvre la protection par la suite.

3.2.4 Ciel, je n'ai pas de framework PHP !!!!

Nous sommes en 2016, cela n'est pas possible !! Enfin, peut-être que votre code est un portage d'une sombre application développée par un stagiaire d'une agence de communication qui vous a fait votre site Web il y a 10 ans... Et dans ce cas, peu de solutions simples sont disponibles.

Le mieux sera sûrement d'utiliser une bibliothèque tierce comme le **OWASP PHP CSRF Guard** (https://www.owasp.org/index.php/PHP_CSRF_Guard) ou la librairie Anti-CSRF de **ParaGonie** (https://github.com/paragonie/anti-csrf).

Il n'est pas recommandé de refaire un mécanisme de sécurité comme l'anti-CSRF soi-même...À coup sûr, vous allez installer un bug dans le code, même s'il parait simple (d'autant plus que le code du stagiaire de 2006 doit réellement être difficile à lire).

3.2.5 Moi le perfectionniste par contre, c'est simple

En **Django** la solution est incluse aussi dans le *framework* par défaut. Si le jeton n'est pas valide, il sera retourné une page 403 au navigateur. Django utilise les *cookies* et un champ caché dans les formulaires pour cette protection.

Il suffit d'utiliser de la manière suivant l'élément **csrf_token** pour que la protection fonctionne :

```
<form action="" method="post">{% csrf_
token %}
```

Il n'y a pas plus simple :).

3.2.6 Mettons notre code a l'abri du CSRF sur de bons rails

Le *framework* **Ruby on Rail** est assez pratique pour développer des applications Web sécurisées. Il comporte énormément d'éléments simples à intégrer, voire inclus. Pour ce qui est de l'anti-CSRF, la solution sera extrêmement simple et indolore pour la plupart des sites Web.

Ajouter la ligne suivante à votre *controler* :

```
protect_from_forgery with: :exception
```

Cela ajoutera automatiquement un jeton de sécurité et générera une exception dans le cas où le jeton n'est pas valide après la requête.

Il est possible en surchargeant correctement l'exception d'effectuer d'autres choses lors de la gestion de l'exception (logs, déconnexion utilisateur, etc.). Exemple :

```
rescue_from ActionController::InvalidAuth
enticityToken do |exception|
 sign_out_user # method déconnectant
l'utilisateur et détruisant la session.
End
```

CONCLUSION

Comme vous avez pu le voir, il existe des tonnes de solutions pour vous protéger du CSRF (et il en existe même dans des langages un peu plus sales que l'on a pas abordé...). Il est bon de noter que la plupart du temps ces solutions sont quasiment *plug-and-play*, voire pour beaucoup déjà intégrées dans les *frameworks* ou serveurs applicatifs directement.

Il n'y a donc plus d'excuses pour que votre site Web se retrouve pris en otage par de joyeux pirates voulant dépouiller Madame Michu ! ■

RÉFÉRENCES

[1] OWASP Top10 2013 – Le CSRF : https://www.owasp.org/index.php/Top_10_2013-A8-Cross-Site_Request_Forgery_(CSRF)

[2] CSRF Dans le classement du WASC : http://projects.webappsec.org/w/page/13246919/Cross%20Site%20Request%20Forgery

[3] Étude de la société WaveStone FR sur la sécurité des sites Web : https://www.wavestone.com/app/uploads/2016/10/Benchmark-Web-Security.pdf

[4] Documentation sur le filtre anti-CSRF de TOMCAT : https://tomcat.apache.org/tomcat-8.0-doc/config/filter.html#CSRF_Prevention_Filter)

[5] Page du projet OWASP CSRF Guard : https://www.owasp.org/index.php/Category:OWASP_CSRFGuard_Project

[6] Documentation anti-CSRF de Spring : http://docs.spring.io/spring-security/site/docs/current/reference/html/csrf.html

[7] Configuration de Symfony : http://symfony.com/doc/current/reference/configuration/framework.html

[8] Anti-CSRF dans Symfony : http://symfony.com/doc/current/form/csrf_protection.html

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 et privé 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

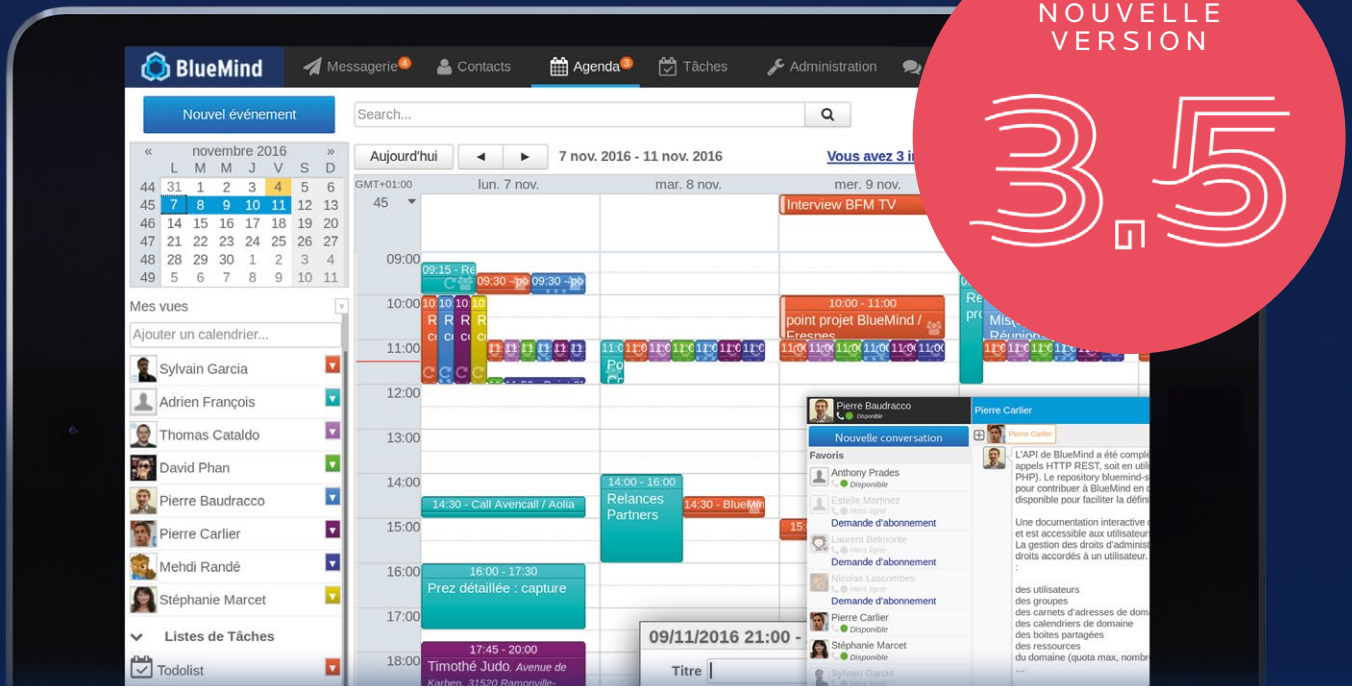
200... BRAVO ! BLUEMIND ACCOMPAGNE LINUXMAG DANS SON RENOUVEAU !

BlueMind

SOLUTION OPENSOURCE
PROFESSIONNELLE DE MESSAGERIE
COLLABORATIVE

LIBÉREZ VOTRE MESSAGERIE

EMAILS
AGENDAS
CONTACTS
TÂCHES
CHAT
MOBILES
DOCUMENTS & PIÈCES JOINTES
ENVOI FICHIERS VOLUMINEUX
...
...



NOUVELLE VERSION
3.5

FRANCAIS / NOMBREUSES RÉFÉRENCES / ERGONOMIQUE / ÉVOLUTIF / ÉCONOMIQUE

Découvrez l'écosystème BlueMind et toutes les fonctionnalités sur
www.bluemind.net