



Developpez

Le Mag

Édition de février - mars 2012.

Numéro 38.

Magazine en ligne gratuit.

Diffusion de copies conformes à l'original autorisée.

Réalisation : Alexandre Pottiez

Rédaction : la rédaction de Developpez

Contact : magazine@redaction-developpez.com

Sommaire

Qt	Page 2
Access	Page 9
Android	Page 25
Eclipse	Page 34
Java	Page 41
Developpement Web	Page 45
DotNet	Page 66
Business Intelligence	Page 69
Liens	Page 71

Article Qt



Débogage d'une application Qt – Aperçu des outils

Cet article retrace et détaille les différents outils présentés durant les Qt Developer Days 2011, pouvant aider au débogage d'applications Qt et Qt Quick.

par **Alexandre Laurent**
Page 2



Article Eclipse

Introduction à Zest

Au travers d'un exemple simple, nous montrerons comment construire un graphe en utilisant cette boîte à outils.

par **Alain Bernard**
Page 34

Éditorial

Une fois n'est pas coutume, la rédaction se plie en quatre pour vous faire découvrir ou redécouvrir ces meilleurs articles, news, billet blog de ce nouveau numéro du magazine du club des professionnels de l'informatique.

Profitez-en bien !

La rédaction

Débogage d'une application Qt - Aperçu des outils de débogage pour Qt

Basé sur la présentation de Volker Krause, employé chez KDAB, menée durant les Qt Developer Days 2011 à Munich, cet article retrace et détaille les différents outils présentés durant la conférence, pouvant aider au débogage d'applications Qt et Qt Quick.

1. Introduction

Programmer est vraiment une tâche compliquée. Lors de l'écriture du code, il arrive très souvent que des oublis soient faits, ou que le programmeur intègre des erreurs. Certaines sont détectées par le compilateur directement, mais comme celui-ci ne peut pas deviner l'objectif final du code, les bogues se glissent dans les mailles du filet et provoquent des comportements inappropriés dans l'application. Selon les conséquences, il peut être plus ou moins facile de retrouver l'origine de l'erreur.

Comme le débogage peut se révéler très difficile, les programmeurs ont mis en place différentes techniques afin d'aider à la résolution des problèmes. La plupart d'entre elles consistent simplement à afficher les valeurs des variables pendant l'exécution même du programme, permettant de donner une idée plus juste de ce qu'il se passe réellement dans l'ordinateur.

Cet article présentera les différentes méthodes et outils existants afin de mieux comprendre le fonctionnement de nos applications et de pouvoir trouver la cause des bogues. Bien que l'article recense des outils de débogage génériques, nous nous pencherons plus spécialement sur les différents moyens de déboguer une application Qt.

2. La bonne vieille méthode

2.1. Affichage des messages dans la console

Il n'est pas possible d'écrire un article sur le débogage d'une application sans passer par la méthode la plus simple et la plus facile à mettre en place. Celle-ci consiste à afficher les valeurs des variables en utilisant la fonction basique du langage. En C++ nous utilisons `std::cout`, en C la fonction `printf()` ([Lien 01](#)).

Le principe est plus que simple. L'affichage de messages permet d'indiquer les différents morceaux de code que le programme exécute ou encore d'afficher les valeurs des variables afin de savoir si celles-ci sont correctes.

Toutefois, l'utilisation d'une fonction supplémentaire pose quelques problèmes lors des erreurs de segmentation dues à la corruption de la mémoire. Plus précisément, si le bogue change des valeurs en mémoire, le fait d'ajouter du code pour le débogage modifiera l'agencement de la mémoire. Du coup, après l'ajout de l'appel à la fonction, les conséquences du bogue peuvent être complètement différentes.

2.2. Affichage de messages avec Qt

2.2.1. Fonctions génériques

Qt propose des fonctions permettant de rapporter des messages lors de l'exécution du programme. Il en existe quatre, chacune correspondant à un niveau de sévérité différent :

- `QDebug()` est utilisée pour les sorties de débogage personnalisées : [Lien 02](#) ;
- `qWarning()` est utilisée pour des avertissements et des erreurs récupérables dans l'application : [Lien 03](#) ;
- `qCritical()` est utilisée pour des erreurs critiques et les erreurs systèmes : [Lien 04](#) ;
- `qFatal()` est utilisée pour tracer des erreurs fatales juste avant l'arrêt de l'application : [Lien 05](#).

Avec Qt, il est possible de modifier le comportement de ces fonctions afin de, par exemple, écrire les messages dans un fichier. Pour ce faire, il suffit de donner à Qt une fonction qu'il appellera lorsqu'il devra afficher les messages. Cette fonction de remplacement doit être renseignée avec `qInstallMsgHandler` : [Lien 06](#).

Vous pouvez éliminer les fonctions `QDebug()` et `qWarning()` de l'exécution en définissant les macros `QT_NO_DEBUG_OUTPUT` et `QT_NO_WARNING_OUTPUT`.

En définissant `QT_FATAL_WARNINGS`, les appels à la fonction `qWarning()` stoppe complètement le programme à l'identique de `qCritical()`.

2.2.2. Affichage détaillé des classes

Afin d'afficher des informations détaillées sur les classes et la hiérarchie des classes, Qt propose deux fonctions supplémentaires :

- `QObject::dumpObjectTree()` : affiche la hiérarchie de l'héritage de la classe ([Lien 07](#)) ;
- `QObject::dumpObjectInfo()` : affiche des informations sur les connexions signaux/slots de la classe ([Lien 08](#)).

3. Le code à votre rescousse : les assertions

3.1. Les assertions en C

Les assertions ne permettent pas vraiment de déboguer du code mais de détecter les erreurs plus rapidement. En effet, la méthode `assert()` ([Lien 09](#)) permet de stopper

l'application lorsque l'argument passé en paramètre est équivalent à zéro (considéré comme faux).

Avec cette fonction, il est très facile de s'assurer que les utilisateurs passent les bons arguments à la fonction ou encore que le programme est dans l'état approprié pour exécuter cette fonction (remplissage de précondition).

Comme il est indiqué dans la documentation, pour avoir accès à la fonction, il est nécessaire d'inclure le fichier `cassert` (ou `assert.h` en C). Voici un petit exemple d'un cas simple d'utilisation des assertions :

Exemple d'utilisation des assertions

```
#include <assert.h>

void foo(char* myString)
{
    assert(myString); // Si le pointeur est NULL
    (soit 0) l'assertion va être levée et le
    programme arrêté.
    // Ici, on est sûr que myString n'est pas un
    pointeur NULL, on peut donc l'utiliser.
    printf("%s\n",myString);
}

int main()
{
    foo(0);
    return 0;
}
```

Sortie de l'application

```
main: main.c:5: foo: Assertion `myString' failed.
Abandon
```

Si un pointeur nul est passé à cette fonction, le programme s'arrêtera et affichera un message clair dans la console. Si en plus, le programme est exécuté dans un débogueur celui-ci stoppera à la ligne de l'assertion et le débogage sera possible à partir de ce point.

Les assertions ne sont exécutées que dans les programmes compilés en mode de débogage. Notamment, le mode release des différents EDI définit la macro `NDEBUG` qui désactive les assertions. Lorsque cette variable est définie, les assertions sont inexistantes (aucun appel à une fonction, aucune perte en performances) donc, ne vous en privez pas.

3.2. Les assertions dans Qt

Généralement, les assertions Qt sont identiques aux assertions en C.

Malgré tout, quelques différences existent :

- deux macros sont définies : `Q_ASSERT` ([Lien 10](#)) et `Q_ASSERT_X` ([Lien 11](#)). La première a un comportement identique à `assert()`. La deuxième laisse la possibilité à l'utilisateur de définir le message à afficher lorsque l'assertion est levée ;
- pour avoir accès à ces macros, il faut inclure « QtGlobal » ;
- pour désactiver les assertions dans Qt, il faut définir la macro `QT_NO_DEBUG`.

4. Les outils généralistes

4.1. Les débogueurs

Les débogueurs sont des programmes utilisant les données de débogage du programme afin de pouvoir tracer le déroulement du programme pendant son exécution. Il est possible, à l'aide d'un débogueur, d'exécuter le programme étape par étape, de stopper le programme à un endroit précis (à l'aide de points d'arrêt), d'afficher les valeurs des variables, de s'arrêter sur le code provoquant un crash ou encore d'afficher la liste d'appels des fonctions.

Afin d'avoir de meilleurs résultats lors du débogage du code celui-ci doit être compilé en mode « debug ». Si on s'intéresse à ce qui se passe au niveau du compilateur, cela signifie que le programme est créé avec des informations de débogage (alourdissant la taille du programme final) qui renseigne le débogueur d'une multitude d'informations utiles pour tracer le programme.

4.1.1. gdb

`gdb` signifie « the GNU Debugger ». Ce programme est disponible sous les systèmes Unix-Like et existe aussi sous Windows dans la suite MinGW.

Bien qu'il dispose de toutes les fonctionnalités nécessaires pour le débogage d'un programme, il est toujours quelque peu rebutant de par son interface, purement console. Heureusement, il est souvent utilisé au travers d'une interface graphique comme nous allons le voir.

Voici un petit extrait des commandes de `gdb` :

Commande	Description
<code>gdb monProgramme</code>	Charger le programme dans <code>gdb</code>
<code>run</code>	Lancer le programme
<code>print maVariable</code>	Afficher une variable
<code>break monFichier.ext : numero_de_ligne</code>	Définir un point d'arrêt dans le fichier ' <code>monFichier.ext</code> ' à la ligne ' <code>numero_de_ligne</code> '
<code>bt</code>	Afficher la liste d'appel
<code>next</code>	Exécuter une ligne (sans entrer dans la fonction)
<code>step</code>	Exécuter une ligne (entrer dans la fonction)
<code>quit</code>	Quitter <code>gdb</code>

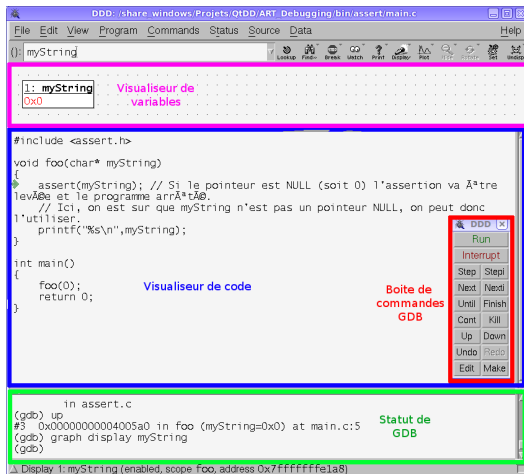
Afin que `gdb` puisse analyser correctement les structures de données de Qt, vous devez installer un script disponible dans le dépôt de KDE : [Lien 12](#). Pour l'installer vous devez renseigner le chemin menant au script, dans le fichier `~/gdbinit`. Voici un exemple :

```
source /chemin/vers/le/fichier/kde-devel-gdb
```

4.1.1.1. DDD

DDD est une surcouche offrant une interface graphique à `gdb`. Elle permet de donner des ordres à `gdb` avec la souris. Malheureusement, ce programme n'est disponible que pour

les systèmes Unix-like et n'est plus mis à jour. De plus, un désavantage notable est qu'il ne s'intègre pas à l'éditeur de code.



Pour apprendre à l'utiliser, vous pouvez consulter le tutoriel écrit par Hiko Sejuro : [Lien 13](#).

4.1.1.2. Les EDI

Tout EDI qui se respecte se doit d'intégrer un débogueur afin de permettre la recherche d'erreurs dans le code affiché par l'éditeur. Pour ne citer que l'un des plus connus : Code::Blocks ([Lien 14](#)), intègre un débogueur qui n'est autre que gdb (pour Windows, son port MinGW).

Ainsi, il est parfaitement possible de déboguer du code, par le biais d'une interface graphique qui reste constante dans tout le processus de création d'un programme. La fenêtre d'édition du code permettant aussi de déboguer le code.

4.1.1.3. QtCreator

QtCreator intègre un débogueur comme tout autre EDI. La particularité de QtCreator, c'est la présence d'un ajout permettant l'analyse complète des types de Qt dans le débogueur. Grâce à cela, il est beaucoup plus facile de déboguer les différents types présents dans Qt, tels que les QList.

QtCreator repose aussi sur gdb, lorsqu'il utilise la chaîne de développement G++ (ou MinGW) ou sur cdb avec le compilateur de Microsoft. Même si le débogueur peut différer, leur utilisation au sein de QtCreator reste la même.

4.1.2. Microsoft Visual Studio Debugger

Microsoft Visual Studio possède lui aussi un débogueur directement intégré dans la suite logicielle. Celui-ci est spécifique à Microsoft et rend le débogage très facile. De plus, il est possible d'avoir la résolution des types Qt grâce au module Qt pour Visual Studio : [Lien 15](#).

Pour plus d'informations sur cet outil, veuillez consulter le tutoriel de Laurent Gomilla : [Lien 16](#).

4.2. La surveillance de la mémoire

Une autre source de bogues est due à la gestion de la

mémoire. Certains peuvent être sans conséquence sur l'exécution du programme, tels que les fuites de mémoire, d'autres peuvent avoir des conséquences « aléatoires » sur le programme (ce qui rend très compliquée la recherche de l'origine du problème).

4.2.1. Valgrind

Valgrind est un outil disponible uniquement sous GNU/Linux et Mac OS. Valgrind permet de simuler l'exécution du programme afin de vérifier tous les accès mémoire effectués par celui-ci. À la fin de l'exécution, Valgrind affiche des informations sur les fuites de mémoire (pointeurs non référencés, mémoire non libérée) ainsi que tous les accès en dehors de la mémoire allouée pour le programme (donc, des erreurs de segmentation).

L'utilisation de Valgrind est simple :

Lancement d'un programme avec Valgrind

```
valgrind --leak-check=full --show-reachable=yes monProgramme
```

Pour connaître le récapitulatif des fuites de mémoire, il suffit de lire la dernière partie du rapport :

Récapitulatif des fuites de mémoire

```
LEAK SUMMARY:
==19814== definitely lost: 92 bytes in 4
blocks
==19814== indirectly lost: 25,664 bytes in 11
blocks
==19814== possibly lost: 8,176 bytes in 1
blocks
==19814== still reachable: 120,614 bytes in
1,348 blocks
==19814== suppressed: 0 bytes in 0 blocks
```

- definitely lost : indique la mémoire totalement perdue. Une fuite de mémoire est à corriger ;
- indirectly lost : indique la mémoire perdue dans les structures basées sur des pointeurs (par exemple, si une racine d'un arbre est définitivement perdue, les enfants sont indirectement perdus). Si vous corrigez une fuite définitivement perdue, il y a de fortes chances que les fuites indirectement perdues partent aussi ;
- possibly lost : indique une perte de mémoire dont Valgrind n'est pas totalement sûr (utilisation étrange des pointeurs) ;
- still reachable : indique une fuite dont vous avez toujours accès aux pointeurs (donc facilement corrigible) ;
- suppressed : indique les fuites de mémoire qui ont été supprimées.

Les erreurs (fuites de mémoire, accès en dehors de la mémoire allouée) sont représentées de la façon suivante :

Erreur dans Valgrind

```
==19814== 25,544 (56 direct, 25,488 indirect)
bytes in 1 blocks are definitely lost in loss
record 169 of 170
==19814== at 0x4C27CC1: operator new(unsigned
long) (vg_replace_malloc.c:261)
==19814== by 0x407C03:
NE::SDL_Engine::initAPI() (SDL_Engine.cpp:65)
```

```

==19814== by 0x4042D7: NE::NEngine::init ()
(NEngine.cpp:43)
==19814== by 0x444C92: main (main.cpp:49)

```

Comme vous pouvez le remarquer, les renseignements donnés sont précieux. On voit facilement l'origine de l'erreur.

Lors de l'utilisation de Valgrind avec un programme Qt, celui-ci rapporte de nombreux problèmes liés aux classes de Qt. Afin que ces messages faux positifs soient cachés, il est nécessaire d'utiliser l'intégration de Valgrind dans QtCreator.

Afin que Valgrind puisse faire son travail correctement, le programme doit contenir les informations de débogage.

4.2.2. Massif

Massif est un sous-outil pour Valgrind, permettant de surveiller les usages de la pile du programme.

Utilisation de massif

```
Valgrind --tool=massif monProgramme
```

L'outil générera un fichier volumineux. Par conséquent, sa lecture manuelle est du suicide. Il existe un visualiseur appelé « massif-visualizer » : [Lien 17](#).

Encore une fois, à cause du code de Qt, il est nécessaire de filtrer les sorties du programme. Pour celui-ci, il faut ignorer les allocations de mémoire Qt (qMalloc...). Pour ce faire, utilisez la commande suivante :

```
Valgrind --tool=massif --alloc-fn=qMalloc()
monProgramme
```

4.3. Les profilers

Les profilers sont des outils permettant d'analyser là où le programme passe le plus de temps. Ainsi, en connaissant mieux le comportement de son application, il est possible de faire des optimisations plus efficaces.

4.3.1. Callgrind

Callgrind est un sous-outil de la suite Valgrind. Celui-ci permet de connaître le temps d'exécution de chaque fonction. Ainsi, vous pouvez déterminer quelles parties de votre application sont critiques.

Utilisation de Callgrind

```
valgrind --tool=callgrind monProgramme
```

Afin d'éviter d'avoir des résultats embrouillés par le système de signaux/slots de Qt, il est préférable d'ignorer les fonctions internes suivantes :

- QMetaObject::activate* ;
- QMetaObject::metacall* ;
- *::qt_metacall*.

Pour ce faire, utilisez la commande suivante :

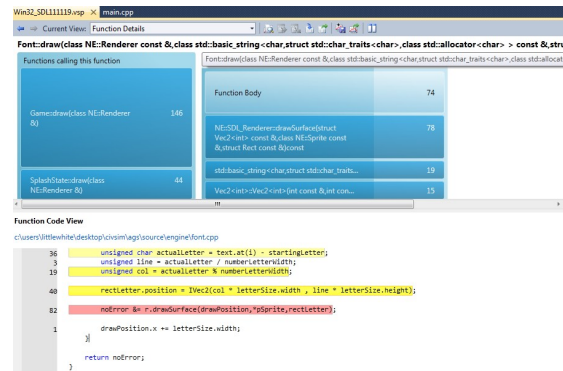
```
valgrind --tool=callgrind --fn-
skip=QMetaObject::activate* --fn-
skip=QMetaObject::metacall* --fn-
skip=*::qt_metacall* monProgramme
```

Afin de visualiser les résultats de Callgrind, il est conseillé d'utiliser KCachegrind, qui affichera graphiquement les différentes informations récoltées par Callgrind.

4.3.2. Microsoft Visual Studio Profiler

Microsoft propose un profiler (style de Callgrind) directement dans Visual Studio (uniquement pour les versions « Premium » et « Ultimate »). L'outil est accessible à partir du menu « Analyser » > « Démarrer le gestionnaire de performance » > « Échantillonnage CPU ».

Après analyse, on obtient un résultat similaire à celui-ci :



L'outil est complet et très puissant. Au début il ne faut pas hésiter à farfouiller dans les différents panneaux présentés.

5. Configurer Qt pour aider au débogage

Qt dispose de nombreuses aides pour faciliter le débogage des applications. Pour activer ces différentes options, vous devez définir certaines variables d'environnement. De plus, il faut que Qt soit compilé en mode de débogage. Dans les parties suivantes, les variables les plus utiles vous seront décrites. Une liste complète est disponible à l'adresse suivante : [Lien 18](#).

5.1. Débugger le rafraîchissement des fenêtres

Si vous avez des ralentissements dus à l'affichage graphique, ou à des bogues d'affichage pour certains objets, les variables QT_FLUSH_PAINT, QT_FLUSH_PAINT_EVENT et QT_FLUSH_UPDATE vous seront utiles afin de savoir quelles sont les zones de vos fenêtres qui reçoivent les événements de dessin. Une fois cette option activée, vos fenêtres clignoteront indiquant les zones recevant les signaux de réaffichage.

5.2. Plus d'informations sur SQL

Il arrive bien souvent que le driver SQL ne soit pas chargé. Habituellement, Qt n'est pas très bavard sur la raison de l'échec de connexion au serveur SQL.

La variable QT_DEBUG_PLUGINS, demandera à Qt d'afficher des informations à propos du chargement des modules. Les causes suivantes provoquent l'affichage de messages :

- un mauvais chemin de recherche ;
- une incompatibilité des versions ;
- une incompatibilité des clés de construction.

Malgré tout, cela peut ne pas suffire à vous aider. Dans ce cas, votre dernier recours sera d'utiliser les options :

- LD_DEBUG ; pour GNU/Linux ;
- DYLD_PRINT_LIBRARIES ; pour Mac OS.

Comme vous pouvez le remarquer, il n'y a pas d'option pour les systèmes Windows.

5.3. Plus d'informations sur DBUS

Pour une application utilisant DBUS, il peut être intéressant de connaître les messages qui transitent. Pour ce faire, l'option QDBUS_DEBUG existe et provoquera un affichage complet de tous les messages interapplications.

5.4. Plus d'informations sur QWebKit

Tout d'abord, QWebKit n'est pas compilé en mode débogage même si Qt l'est. En effet QWebKit est tellement lourd en mode de débogage qu'il faut préciser à la compilation que l'on veut vraiment la version de débogage. Pour ce faire, ajoutez l'option -webkit-debug pendant la configuration.

L'option permettant d'afficher les messages d'information de QWebKit est : QT_WEBKIT_LOG.

5.5. Plus d'informations sur Phonon

Finalement, le module Phonon envoie des informations supplémentaires sur son état lorsqu'on utilise l'option PHONON_DEBUG ou encore PHONON_<backend>_DEBUG, où « backend » est le nom d'un plug-in de Phonon.

6. Les outils spécifiques à Qt

Finalement, pour le débogage d'applications Qt, des outils spécifiques ont été mis en place. Ceux-ci se décomposent en deux classes :

- les classes à utiliser dans le code ;
- les outils externes.

6.1. Classes Qt spécialisées dans le débogage

6.1.1. QScriptEngineDebugger

La classe QScriptEngineDebugger ([Lien 19](#)) fournit un débogueur qui peut être directement incorporé dans les applications qui utilisent Qt Script. L'activation du débogueur permet à l'utilisateur d'inspecter l'état du script et de contrôler son exécution.

Vous pouvez utiliser le débogueur de la manière suivante :

Utilisation de QScriptEngineDebugger

```
QScriptEngine engine;
QScriptEngineDebugger debugger;
debugger.attachTo(&engine);
```

Une fois cela fait, le débogueur sera appelé lorsque le script enverra une exception non rattrapée ou exécutera une fonction de débogage. D'autres interactions avec le script sont possibles et décrites dans la documentation : [Lien 20](#).

6.1.2. QWebInspector

La classe QWebInspector ([Lien 21](#)) permet de prendre le contrôle d'un outil d'inspection pour une QWebPage. L'outil permet d'afficher la hiérarchie de la page, ses statistiques de chargement et l'état courant de chaque élément. L'outil est surtout destiné aux développeurs web.

Le code suivant présente une méthode pour utiliser la classe :

Utilisation de QWebInspector

```
// ...
QWebPage *page = new QWebPage;
// ...

QWebInspector *inspector = new QWebInspector;
inspector->setPage(page);
```

6.2. Outils

6.2.1. GammaRay

GammaRay est un outil open source développé par KDAB, permettant d'examiner le cœur d'une application Qt : [Lien 22](#). Pour cela le programme utilise des méthodes d'injection afin de recueillir les informations pendant l'exécution. De plus, l'outil fournit une interface facilitant le parcours et l'analyse des structures internes de Qt.

Contrairement à un débogueur classique GammaRay donne une vue d'ensemble à votre programme. En elle-même, l'application comprend un visualiseur de propriétés, signaux, slots d'objets, un navigateur de contenu de QAbstractItemModel et de ses dérivés et un navigateur de scènes graphiques, un visualiseur de machine à états et un navigateur de QTextDocument. De plus, l'outil permet d'utiliser QScriptEngineDebugger et QWebInspector sans avoir à modifier le code.

Finalement, l'outil peut s'intégrer dans QtCreator.

7. Exemples de code

7.1. Shoot a bug

ShootABug est une classe permettant d'afficher des informations de débogage lorsque l'utilisateur clique sur le QObject qui en hérite.

Shoot a bug

```
class ShootABug: public QObject
{
    Q_OBJECT
public:
    bool eventFilter( QObject* recv, QEvent*
event )
    {
        if ( event->type() !=
QEvent::MouseButtonPress )
            return false; // pass it on
        QMouseEvent* mevent =
static_cast<QMouseEvent*>(event);
        if ( (mevent->modifiers() &
Qt::ControlModifier) &&
            (mevent->button() & Qt::LeftButton)
        ) {
```

```

// Ctrl + left mouse click.
qDebug("-----");
-----");
qDebug("Widget name : %s",
qPrintable( recv->objectName() ) );
qDebug("Widget class: %s", recv-
>metaObject()->className() );
qDebug("\nObject info:");
recv->dumpObjectInfo();
qDebug("\nObject tree:");
recv->dumpObjectTree();
qDebug("-----");
return true; // Block
}
return false;
}
};

```

8. Notes

Le problème des débogueurs et autres analyseurs est que le programme est fortement ralenti pendant l'exécution. Du coup, si un bogue apparaît à cause d'une mauvaise synchronisation des données, il se peut que celui-ci ne soit absolument pas visible lors du débogage. La conséquence dans une application Qt est que la boucle événementielle sera appelée moins souvent et les timers se déclencheront

Les blogs Qt

Adieu qmake, bienvenue qbs !

Outil ô combien utile à tout développeur Qt... et pourtant ô combien détesté ! Il fonctionne bien, mais n'est pas la partie la plus maintenable de Qt. On a déjà exploré ce que tout remplaçant potentiel de qmake doit être à même de faire ([Lien 28](#)) (et une liste de commentaires et questions à ce sujet : [Lien 29](#)). Notamment, toute une série d'outils actuellement disponibles ont été étudiés, aucun ne remplissait totalement ce cahier des charges (la discussion a bien continué pour Qt 5 sur la mailing list : [Lien 30](#)). C'est pourquoi un projet interne à l'équipe de développement de Qt a été lancé pour tester l'une ou l'autre idée, d'où un tout nouvel outil : la *Qt Build Suite*, qbs, à prononcer *cubes*.

C'est tout sauf qmake : pas de lien à la version de Qt, génération d'un graphe de compilation propre à partir de la description de haut niveau du projet, plus de génération de Makefile suivi d'un appel à make/nmake/gmake ou autre. En lieu et place, qbs agit comme un make parallèle, il appelle directement le compilateur, l'éditeur de liens et tout autre outil utile à la compilation du projet (à la manière de SCons ou Ant).

C'est un nouveau langage déclaratif : après QML, l'environnement Qt semble se mettre à la mode déclarative. En réalité, le langage utilisé par qbs est une version allégée de QML. Il fournit une représentation facile à utiliser pour l'EDI, tout en laissant la liberté d'écrire des expressions JavaScript. L'éditeur de fichier de projet devrait gérer lui-même toutes les listes de chaînes littérales et, pour les constructions plus compliquées, n'agir qu'en tant qu'éditeur de texte (ces expressions n'étant requises que si l'on a besoin de plus de puissance... et laissant alors littéralement tout faire, au vu de

moins.

9. Conclusion

Tout au long de cet article, nous avons vu les différents outils et méthodes permettant de faciliter la vie du programmeur. Certes ces outils sont efficaces, mais il est préférable d'adopter des habitudes de codage permettant d'éviter de créer des bogues. Nous avons vu les assertions, qui sont l'une d'entre elles, mais nous pouvons aussi citer les tests unitaires ([Lien 23](#)) qui permettent de détecter l'origine d'un bogue très tôt dans le processus de développement. De plus, il est possible de se faciliter la tâche en écrivant dans un journal les informations importantes de l'application, permettant ainsi de reproduire plus facilement les conditions provoquant un bogue.

10. Liens

- Diaporama de la présentation de Volker Krause : [Lien 24](#).
- Vidéo de la présentation de Volker Krause : [Lien 25](#).
- Documentation traduite de Qt : [Lien 26](#).

Retrouvez l'article d'Alexandre Laurent en ligne : [Lien 27](#)

l'extensibilité). Par exemple :

```

files: ["foo.h", "foo.cpp", "main.cpp"]
// éditable par l'EDI automatiquement
files: generateFileList().concat(['extra.cpp'])
// éditable uniquement à la main

```

Un premier exemple complet, l'habituel *Hello World* :

```

import qbs.base 1.0

CppApplication {
    name: "HelloWorld"
    files: "main.cpp"
}

```

Une description complète du langage est disponible dans la documentation : [Lien 31](#).

C'est extensible : alors qu'on cherche à tout prix à éviter avec qmake de générer du code ou de compiler des ressources, qbs fournit une syntaxe pour écrire des règles de transformation de fichier d'un certain type en un autre. On peut appeler des programmes pour le faire (comme rcc) ou exécuter directement la transformation en JavaScript. L'exemple simplifié qui suit montre comment transformer des fichiers `.pluginspec.in` en `.pluginspec` (comme ceux utilisés pour Qt Creator) :

```

Rule {
    ...
    prepare: {
        var cmd = new JavaScriptCommand();
        cmd.description = "generating " +
File: fileInfo.fileName(output.fileName);
        cmd.qtcreator_version =
product.module.qtcreator_version;

```

```

cmd.sourceCode = function() {
    var inf = new
TextFile(input.fileName);
    var all = inf.readAll();
    all = all.replace(new RegExp('\\\$\\\$\\$
$QTCREATOR_VERSION(?!\\w)', 'g'),
qtcreator_version);
    var file = new
TextFile(output.fileName, TextFile.WriteOnly);
    file.write(all);
    file.close();
}
return cmd;
}
}

```

C'est rapide pour des compilations incrémentales : qbs voit le projet de loin, comme un seul bloc, il ne doit pas se forker pour les sous-dossiers. Même si seulement une partie du projet est déjà compilée, le graphe complet de compilation est pris en considération, il n'y a plus d'appel récursif (qui devrait absolument être évité : [Lien 32](#)). Cela a pour effet secondaire de rendre les compilations incrémentales très rapides.

En modifiant un script de benchmarking pour supporter qbs ([Lien 33](#)), on peut comparer make et qbs, respectivement, pour une compilation incrémentale :

```

real 0m4.076s
user 0m2.556s
sys 0m1.952s

real 0m0.843s
user 0m0.724s
sys 0m0.112s

```

Vous voulez déjà tester ? Les sources sont d'ores et déjà disponibles sur Gitorious : [Lien 34](#). Actuellement, le projet est au stade expérimental, un terrain d'expérimentation pour de nouveaux concepts. qmake va encore résister un certain temps, personne ne sera forcé à abandonner qmake (bien que Qt va fort probablement passer à qbs dès que possible). Un point crucial reste à implémenter : l'adaptation à l'environnement de compilation (aussi appelée processus de configuration). Pour le moment, la seule manière de procéder est d'utiliser un outil externe qui génère un fichier JSON qui sera chargé par qbs. La solution envisagée actuellement est de rendre les tests de configuration utilisables par les modules (une implémentation qui ne sera donc plus entièrement déclarative, mais qui contiendra une bonne proportion de JavaScript).

Retrouvez ce billet blog de Thibaut Cuvelier en ligne : [Lien 35](#)

Positionner un formulaire par rapport au contrôle d'un autre formulaire

Dans un formulaire, on souhaite parfois apporter - sur demande de l'utilisateur - un complément d'information aux données affichées dans l'enregistrement actif.

Ce tutoriel décrit comment ouvrir un second formulaire donnant plus de détails à un endroit qui dépend de la position d'un contrôle du premier.

Ceci est la suite du tutoriel Positionner un formulaire à un endroit déterminé : [Lien 36](#).

Nous ferons un pas de plus dans l'exploitation de la théorie exposée dans le tutoriel précédent.

Cette fois, en cliquant sur un contrôle d'un formulaire, nous allons ouvrir un second formulaire juste en dessous et nous rectifierons sa position si ce second formulaire déborde de l'écran.

1. Avant-propos

Considérez ceci comme un prétexte pour manipuler quelques propriétés des formulaires Access et apprendre à écrire quelques lignes de code.

Si vous êtes pressé et que votre intérêt porte sur la solution, voyez plutôt ces deux contributions :

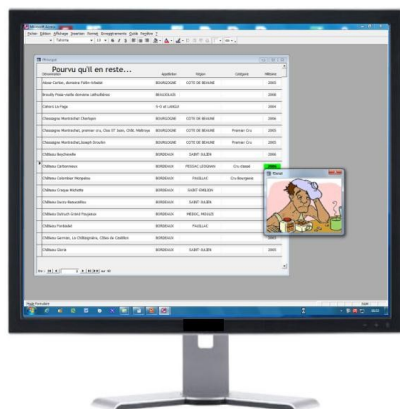
Argyronet a traité le sujet dans son tutoriel Comment positionner un formulaire ouvert depuis un autre, à droite de l'écran : [Lien 37](#) ;

Arkham46 dans cette contribution ([Lien 38](#)), donne tout le code pour positionner un formulaire sous un contrôle d'un autre formulaire.

La solution d'Arkham46 aboutit pratiquement au même résultat que nous et en moins de cent lignes, hors commentaires ! Et si vous ne comprenez pas tout ce que font les API qu'il utilise, ce n'est pas trop grave. Personnellement, je conduis une voiture depuis un demi-siècle et je n'ai toujours pas compris comment fonctionne un carburateur ! (Pour ceux qui aiment la précision : j'ai changé plusieurs fois de voiture durant ce laps de temps.)

2. Prérequis

Ce texte s'adresse à des utilisateurs Access débutants qui veulent s'initier à la programmation et **qui ont lu le premier tutoriel**.



3. L'objectif

Dans un formulaire, un double-clic sur un contrôle provoque l'ouverture d'un autre formulaire qui viendra s'afficher juste en dessous du contrôle cliqué...

... et si l'écran est trop petit, le second formulaire se déplace pour rester visible.

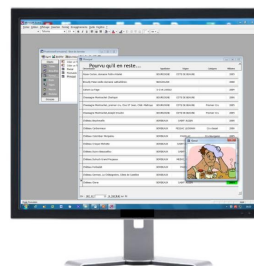


4. Plantons le décor

N'importe quel formulaire ferait l'affaire. Mais pour passer en revue toutes les possibilités, nous avons choisi un formulaire en continu, avec trois sections : *En-tête de formulaire*, *Détail* et *Pied de formulaire*.

Il nous fallait donc baser notre formulaire sur une table. Tant qu'à faire, autant prendre une liste de produits sympathiques, à consommer toutefois avec modération.

En voici un extrait :



Voici le formulaire *fPrincipal* qui nous servira d'exemple :

Dénomination	Appellation	Région	Catégorie	Millesime
Château Phélan Ségur 1/2	BORDEAUX	SAINT-ESTEPHE		2004
Château Flocq	BORDEAUX	POMEROL		2001
Château Roulier Montagne Saint-Emilion	BORDEAUX	SAINT-EMILION		2007
Château Roulier Montagne Saint-Emilion/vob	BORDEAUX	SAINT-EMILION		2008
Château Saint Pierre	BORDEAUX	SAINT-JULIEN		2005
Château Siran	BORDEAUX	MARGAUX		2003
Château Smith Haut Lafitte	BORDEAUX	PESSAC-LEODIGNAN		2000
Château Terrey Gros Cailloux, m.c.	BORDEAUX	SAINT-JULIEN	Cru Bourgeois	2003

Tous les champs de la table sont repris, leur nom est *zdt* suivi du nom du champ. Exemple, *zdtDENOMINATION* pour le champ *DENOMINATION* (« *zdt* » pour **z**one de **t**exte).

Un gadget : *zdtMILLESIME* est mis en évidence en vert dans la ligne de l'enregistrement actif. (Voir plus bas comment on y arrive.)

Ma première idée d'exemple était : si on double-clique sur le millésime, on lance une recherche Google sur les sites spécialisés et on affiche les appréciations des œnologues réputés dans un formulaire. C'est sans doute possible mais ça nous aurait conduits trop loin dans les explications. (Voir à ce sujet Arkham46 VBA et développement Web : [Lien 39](#).) Je vous invite à revoir le sujet... quand le débutant, à qui je m'adresse, aura acquis un peu plus de bouteille(s).

Ici, on fera plus simple, car en fait, n'importe quel formulaire **indépendant** fait l'affaire pour nos explications.

Dans l'exemple, il s'appelle *fDetail* et ne contient rien d'autre qu'une image, sans aucun rapport. Quoique !

Notre objectif : un double-clic sur le contrôle *zdtMILLESIME* de l'enregistrement en cours provoque l'ouverture de *fDetail* qui viendra se superposer à *fPrincipal* juste en dessous et aligné à gauche du contrôle double-cliqué. Ou ailleurs si manque de place.



5. Allons-y pas à pas, d'abord avec des mots

Si le formulaire *fDetail* est une fenêtre indépendante, pour afficher à l'endroit voulu, il « suffit » de fournir les bons paramètres à une instruction *Docmd.MoveSize droite, bas, largeur, hauteur*.

Dès lors, le problème se ramène à localiser le point sous la loupe :



en s'exprimant en twips, par rapport au coin supérieur gauche de notre écran.

5.1. Le paramètre « droite »



Relativement au coin supérieur gauche de l'écran, il se décompose en :

- la distance horizontale du coin supérieur gauche du formulaire *fPrincipal* ;
- l'épaisseur de la bordure gauche du formulaire *fPrincipal* ;
- la largeur de la coulisse du sélecteur d'enregistrement ;
- la distance entre cette coulisse et le bord gauche du contrôle ciblé.

5.2. Le paramètre « bas »



Relativement au coin supérieur gauche de l'écran, il se décompose en :

- la distance verticale du coin supérieur gauche du formulaire *fPrincipal* ;
- la hauteur de la bordure supérieure du formulaire *fPrincipal* ;
- à l'intérieur de *fPrincipal*, la position de la *section détail* actuellement active ;
- dans cette section, la position supérieure de contrôle ciblé ;
- la hauteur de ce contrôle.

5.3. Paramètres « Largeur » et « Hauteur »

Cela dépend des dimensions du formulaire *fDetail*. Fixons les idées : 4245 et 3495 twips, respectivement.

5.4. Reste à vérifier qu'il y aura assez de place à l'écran



En d'autres termes que les distances entre notre point visé et les bords droit et inférieur de l'écran permettent de loger le formulaire *fDetail*.

Si ça déborde à droite, il faudra corriger le paramètre *droite* actuel :

- ajouter la largeur du contrôle *zdtMILLESIME* ;
- soustraire la largeur de *fDetail*.

Donc, aligner le côté droit de *fDetail* sur le côté droit de *zdtMILLESIME*.

Si ça déborde en dessous, il faudra corriger le paramètre *bas* actuel :

- soustraire la hauteur du contrôle *zdtMILLESIME* ;
- soustraire la hauteur de *fDetail*.

Donc, placer le bord inférieur de *fDetail* sur le bord supérieur de *zdtMILLESIME*.

6. Récapitulons les données nécessaires et voyons comment les obtenir

Id Item	Comment le lire en twips ?
• La distance horizontale du coin supérieur gauche du formulaire <i>Principal</i>	$FormPrincipal.CurrentSectionLeft$
• l'épaisseur de la bordure gauche du formulaire <i>Principal</i>	$FormPrincipal.CurrentSectionLeft$
• la largeur de la coulisse de l'effecteur d'enregistrement	$FormPrincipal.CurrentSectionLeft$
• la distance entre cette coulisse et le bord gauche de <i>zdtMILLESIME</i>	$FormPrincipal.zdtMILLESIME.Left$
• la distance verticale du coin supérieur gauche du formulaire <i>Principal</i>	$FormPrincipal.CurrentSectionTop$
• la hauteur de la bordure supérieure du formulaire <i>Principal</i>	$FormPrincipal.CurrentSectionTop$
• à l'intérieur de <i>Principal</i> , la position de la section 'détail' actuellement active	$FormPrincipal.CurrentSectionTop$
• dans cette section, la position supérieure de <i>zdtMILLESIME</i>	$FormPrincipal.zdtMILLESIME.Top$
• la hauteur de <i>zdtMILLESIME</i>	$FormPrincipal.zdtMILLESIME.Height$
• la largeur du contrôle <i>zdtMILLESIME</i>	$FormPrincipal.zdtMILLESIME.Width$
• la largeur de l'écran	$FormPrincipal.zdtMILLESIME.Width$
• la hauteur de l'écran (barre des tâches exclue)	$FormPrincipal.zdtMILLESIME.Width$

- Les expressions en vert n'ont pas de secret pour

ceux qui ont lu le précédent tutoriel sur ce sujet.

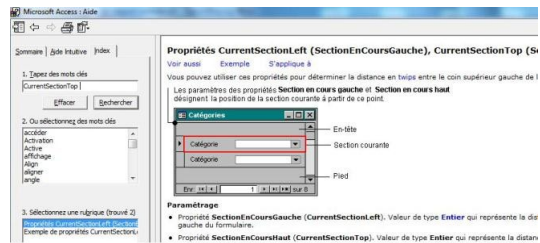
- Pour les expressions en orange, le réflexe



offre une explication bien documentée.

Je ne vais pas la reproduire ici. Tapez <F1> pour ouvrir l'aide Access et ensuite le mot-clé dans la zone de recherche.

Vous obtiendrez quelque chose comme ceci :

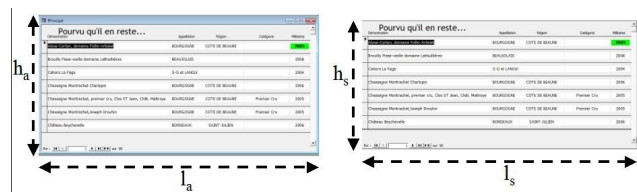


- Pour les parties en rouge, on n'a pas de fonction ni de propriété. On va devoir ruser !

7. Trouver la dimension des bordures

Monsieur de La Palice aurait dit un quart d'heure avant de mourir (donc quand il vivait encore) :

l'espace occupé par les bordures, c'est la différence entre les mesures du formulaire avec bordures et ses mesures sans bordure.



Ce n'est pas aussi bête qu'il y paraît !

Une bordure latérale = la moitié de $(l_a - l_s)$.

La bordure inférieure = la bordure latérale.

La bordure supérieure = $(h_a - h_s) -$ bordure inférieure.

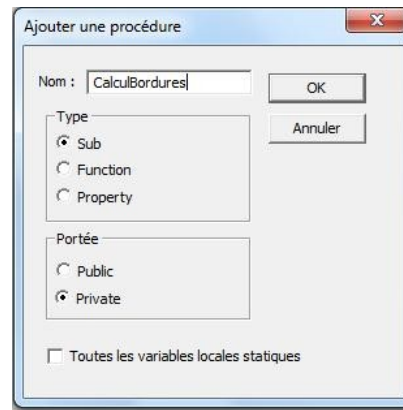
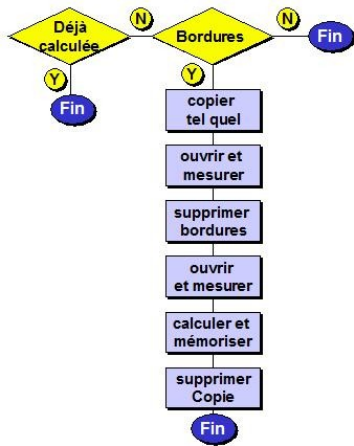
7.1. Suivons la piste

Nous allons nous organiser pour ne faire les calculs qu'une seule fois par session (le fait d'ouvrir la base de données).

Nous allons mémoriser la dimension des bordures dans des variables globales.

Pour relever les mesures, nous allons créer une copie du formulaire *fPrincipal* et l'ouvrir successivement avec et sans bordures. Nous pourrions ainsi réaliser notre objectif.

Voici l'algorithme que nous allons appliquer :



7.2. Traduisons en code

Nous affecterons les dimensions des bordures dans une variable globale d'un type que nous définissons dans le module *mFonctions*. Nous aurons aussi besoin de mémoriser les coefficients de conversion pixels => twips :

```
Type Bordure
  Laterale As Long
  Inferieure As Long
  Superieure As Long
End Type
Type Pixel2Twip
  Horizontal As Long
  Vertical As Long
End Type
Global gBordure As Bordure
Global gBordureDispo As Boolean
Global gPixel2Twip As Pixel2Twip
```

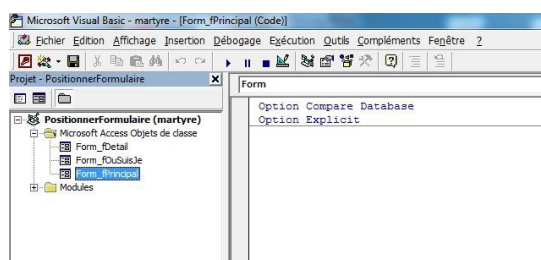
Par défaut, *gBordure.Laterale*, *gBordure.Inferieure* et *gBordure.Superieure* valent zéro et *gBordureDispo* vaut False (Faux).


Dans le module de code du formulaire *fPrincipal* nous créons une procédure.

Enfonchez simultanément <Alt> et <F11> (on note <Alt-Key-F11>) pour ouvrir l'éditeur de code.

<Ctrl-Key-R> pour afficher l'explorateur de projet.

Dans ce dernier, vous cliquez sur **Form_fPrincipal**. Votre écran devrait afficher :

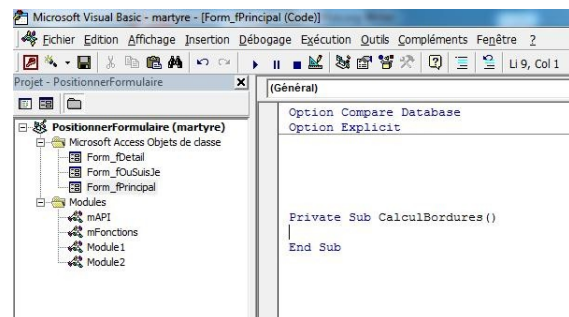


Cliquez sur le dérouleur de l'icône  , il vient ceci . Cliquez sur Procédure...

Nous baptisons notre procédure : « CalculBordures ».

Nous choisissons de donner une portée « privée » à notre procédure. C'est-à-dire qu'elle sera disponible uniquement dans le périmètre du formulaire.

Cliquez sur OK, votre écran affiche :



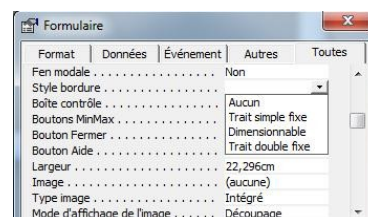
Le reste du code sera inséré là où votre curseur clignote. Entre la déclaration de la procédure et son indicateur de fin.

Dans ce code, la propriété « Style Bordure » s'écrit « .BorderStyle ».

Pour trouver la correspondance entre les noms en français et la syntaxe de la propriété, procédez comme ceci :

- affichez les propriétés du formulaire (ou de l'état) ;
- cliquez sur la propriété choisie, elle s'affiche en surbrillance ;
- enfoncez <F1>, l'aide s'affiche et vous donne le nom anglais.

Si une liste déroulante offre un choix de valeurs, cette dernière s'exprime dans le code au moyen d'un nombre. Ce nombre est en fait le rang de la ligne dans la liste en commençant par zéro.



Par exemple :
à « Aucun » correspond « 0 » ;
à « Dimensionnable » correspond « 2 ».

La syntaxe serait :

Forms!LeFormulaire.BorderStyle = 0 pour exprimer qu'il ne faut pas de bordure.

On définit des variables pour mémoriser les dimensions des fenêtres (*rectForm*) et les résultats intermédiaires (*lLargeAvec*, *lHautAvec*, *lLargeSans* et *lHautSans*, le « l » rappelle qu'elles sont de type entier long) :

```
Dim rectForm As RECT
Dim lLargeAvec As Long
Dim lHautAvec As Long
Dim lLargeSans As Long
Dim lHautSans As Long
```

Prenez dès le début de bonnes habitudes de nommage. Si vous aimez la rigueur, voyez les Conventions typographiques de Argyronet : [Lien 40](#).

On affecte la valeur True à la variable *gBordureDispo*, pour signifier que les données relatives aux bordures seront disponibles. Cela nous évitera de recommencer tous les calculs dès que les dimensions seront connues.

S'il s'avère que le formulaire n'a pas de bordure, on laisse telles quelles les dimensions initiales (zéro) :

```
gBordureDispo = True
If Me.BorderStyle = 0 Then Exit Sub
```

Dans le cas contraire (une valeur de *BorderStyle* ≠ 0), on crée une copie du formulaire que l'on nomme « |bordure| ». Ce nom est arbitraire, la seule contrainte, c'est qu'aucun formulaire ne se nomme déjà ainsi dans la base de données. C'est pour cette raison que je choisis de décorer son nom avec des « pipes » (« | », <Alt> et <124> sur le pavé numérique), ce caractère étant rarement utilisé, un formulaire homonyme est peu probable.

On ouvre ce formulaire en mode caché. Bien qu'invisible à l'écran, la fenêtre existe et nous pouvons la mesurer au moyen de notre fonction *PositionFormPx* que nous connaissons depuis le tutoriel précédent :

```
DoCmd.CopyObject , "|Bordures|", acForm,
CurrentProject.AllForms(Me.Name).Name
DoCmd.OpenForm "|Bordures|", , , , , acHidden
lLargeAvec = PositionFormPx("|Bordures|").Largeur
lHautAvec = PositionFormPx("|Bordures|").Hauteur
```

Nous refermons.

Nous allons maintenant modifier la copie en enlevant les bordures.

Nous l'ouvrons en mode création (caché), pour affecter la valeur 0 à sa propriété *BorderStyle*. Et nous passons en mode formulaire pour mesurer sans bordures :

```
DoCmd.Close acForm, "|Bordures|"
DoCmd.OpenForm "|Bordures|", acDesign, , , ,
acHidden
Forms!["|Bordures|"].BorderStyle = 0
DoCmd.OpenForm "|Bordures|", , , , , acHidden
lLargeSans = PositionFormPx("|Bordures|").Largeur
lHautSans = PositionFormPx("|Bordures|").Hauteur
```

À ce stade, nous disposons des éléments pour calculer la dimension de chacune des bordures :

```
gBordure.Laterale = (lLargeAvec - lLargeSans) / 2
gBordure.Inferieure = gBordure.Laterale
gBordure.Superieure = lHautAvec - lHautSans -
gBordure.Inferieure
```

Nous n'avons plus besoin de notre copie, on la ferme (sans sauver) et on la supprime :

```
DoCmd.Close acForm, "|Bordures|", acSaveNo
DoCmd.DeleteObject acForm, "|Bordures|"
```

On est presque au bout : nous avons la dimension des bordures en pixels, il reste à les transformer en twips.

Il suffit de multiplier les données par leur nombre de twips par pixel.

Souvenez-vous, il s'agit du contenu des contrôles *ZdtNbreTwipspPixelHor* et *ZdtNbreTwipspPixelVer* de notre formulaire *fOusuisJe*.

Nous allons ouvrir, en mode caché, *fOusuisJe*.

On y puise les coefficients de conversion, on en profite pour les mémoriser dans une variable globale (nous en aurons encore besoin plus tard) et nous refermerons le formulaire :

```
DoCmd.OpenForm "fOusuisJe", , , , , acHidden
gPixel2Twip.Horizontal = Forms!
fOusuisJe.ZdtNbreTwipspPixelHor
gPixel2Twip.Vertical = Forms!
fOusuisJe.ZdtNbreTwipspPixelVer
DoCmd.Close acForm, "fOusuisJe"
gBordure.Laterale = gBordure.Laterale *
gPixel2Twip.Horizontal
gBordure.Superieure = gBordure.Superieure *
gPixel2Twip.Vertical
gBordure.Inferieure = gBordure.Inferieure *
gPixel2Twip.Vertical
```

Et voilà, nous avons un « truc » pour les marges en twips.

8. Les dimensions de l'écran

8.1. Une API : SystemParametersInfo

Si, pour vous, l'important est que « ça » fonctionne, même si on ne comprend pas, ce serait sans doute raisonnable de faire appel à une API !

Voilà pourquoi nous donnons ici les ingrédients de la recette, sans expliquer pourquoi c'est bon.

Un peu plus loin dans ce tutoriel, nous expliquerons un moyen plus « acrobatique », avec quelques contorsions qui nous permettront de progresser dans la maîtrise du code.

Vous collez ceci dans un module, par exemple dans le module *mAPI*.

```
Public Declare Function SystemParametersInfo Lib
"User32" _
Alias
"SystemParametersInfoA" (ByVal uAction As Long, _
```

```

ByVal uParam As Long, ByVal
lpvParam As RECT, _
ByVal fuWinIni As Long) As
Long
Public Const SPI_GETWORKAREA = 48

```

Ensuite, cette fonction par exemple dans *mFonctions*.

```

Public Function EcranPx() As Position
Dim lScreenRect As RECT
SystemParametersInfo SPI_GETWORKAREA, 0,
lScreenRect, 0
EcranPx.Largeur = lScreenRect.Right -
lScreenRect.Left
EcranPx.Hauteur = lScreenRect.Bottom -
lScreenRect.Top
End Function

```

Et pour constater que ça fonctionne, ouvrez la fenêtre d'exécution <Ctrl-key-G>, et saisissez :

? *EcranPx().Largeur* <Enter> => la largeur de l'écran s'affiche en pixels.

? *EcranPx().Hauteur* <Enter> => la hauteur de l'écran s'affiche en pixels. Il s'agit de la hauteur **disponible**. C'est-à-dire la distance entre le bord supérieur de l'écran et la barre des tâches si vous avez choisi de l'afficher en permanence.



8.2. Un peu de gymnastique pour le même résultat

On va se servir de la théorie développée dans le tutoriel précédent.

Avec notre formulaire *fOusuisJe* et une dose d'imagination, nous allons mesurer notre écran.

Pour comprendre la démarche, nous allons un peu torturer une copie de notre formulaire *fOuSuisJe*.

Suivez pas à pas :

- prenez une copie : clic sur *fOuSuisJe* dans la fenêtre des objets => surbrillance => <Ctrl-key-C> suivi de <Ctrl-key-V> (équivalent à Copier/Coller) => Access vous invite à donner un nom => « *Martyr* » (par exemple !);
- modifiez la propriété Style Bordure : « Aucune »;
- modifiez le code des événements : <Alt-key-F11> pour ouvrir l'éditeur, dans l'explorateur de projets cliquez sur *Form_Martyr* et modifiez le code pour arriver à ceci :



La flèche bleue indique une instruction ajoutée : le formulaire occupera la totalité de la surface libre, en l'occurrence tout l'écran puisque notre formulaire est une fenêtre indépendante.

Par contre, les flèches vertes montrent des instructions qui ont été « commentées » : elles sont maintenant neutralisées (Access les confondra avec du commentaire). Ceci est une astuce pour garder une trace de la situation initiale quand on met du code au point.

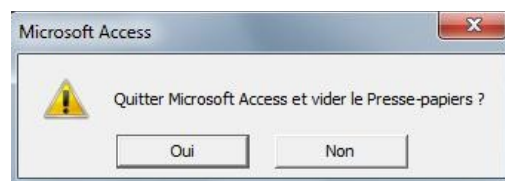
Ces modifications sont en fait : l'ouverture du formulaire à dimension maximale et incorporation dans l'événement « Sur ouverture », du code initialement prévu dans la minuterie (timer) qui a d'ailleurs disparu.

- Sauvegardez « *Martyr* » et ouvrez-le en mode formulaire.

Le contrôle *zdtLargeurTwi* affiche la largeur de l'écran et *zdtHauteurTwi* affiche la hauteur **totale** (ce qu'on veut, c'est la hauteur disponible).

Cela dépend de la résolution de votre écran, par exemple avec une résolution de 1280 x 1024, vous obtiendrez : 19 200 (twips) de largeur et 15 360 (twips) de hauteur.

Puisque vous avez créé une copie de l'objet *fOuSuisJe*, lorsque vous fermerez la session, vous aurez ce message :



Nous avons déjà la moitié de ce qu'on cherche : **la largeur**. Continuons nos basses œuvres sur *Martyr* :

- ouvrons-le à nouveau en mode création, et cette fois, remplaçons *DoCmd.Maximize* par *DoCmd.Minimize* et ouvrons en mode formulaire. Le bas de l'écran affiche ceci :



- en fait la hauteur que nous recherchons, c'est la distance entre le bord supérieur de l'écran et le

bord supérieur de Martyr (*zdtBasTwi*) + la hauteur de Martyr (*zdtHauteurTwi*). Dommage que l'affichage est tel que ces données ne sont pas visibles ;

- qu'à cela ne tienne, on peut quand même les voir ! Ouvrons la fenêtre d'exécution et demandons d'afficher la somme de ces deux valeurs :
? forms!Martyr.zdtBasTwi + forms!Martyr.zdtHauteurTwi <Enter>.

Les nombres affichés dépendent de la résolution de votre écran. Par exemple avec une résolution de 1280 x 1024, vous obtiendrez : 14 910 (twips).

Et voilà le travail !

À titre d'exercice, comparez ces valeurs à celles obtenues par la voie rapide décrite au paragraphe précédent (exprimés en pixels). Les résultats sont identiques. Cependant, un peu comme de la soupe en sachet comparée à celle préparée avec les légumes du jardin, la seconde semble meilleure, juste parce qu'on l'a cuisinée soi-même.

8.3. Une procédure pour automatiser ce processus

Dans *mfonctions*, nous ajoutons une variable globale *gEcran* de type *Position*. Nous pourrions ainsi mémoriser les mesures dans *gEcran.Largeur* et *gEcran.Hauteur*.

Avant de poursuivre, nous allons apporter, manuellement, quelques modifications dans le code de notre formulaire *fOuSuisJe*.

Nous allons donner une portée publique à la procédure *Sub Form_Timer()*. Il y a **Private** *Sub Form_Timer()*, nous modifions en **Public** *Sub Form_Timer()*. Ainsi, nous pourrions faire appel à cette procédure depuis l'extérieur du formulaire.

```
Public Sub Form_Timer()  
[...]
```

Nous allons aussi conditionner la copie dans le presse-papier. Elle ne surviendra que si le formulaire a été ouvert par l'utilisateur.

Pour détecter ce fait, nous utilisons la propriété *CodeContextObject*, elle permet de déterminer dans quel objet, du code Visual Basic est en cours d'exécution. Dans le cas où le formulaire *fOuSuisJe* a été ouvert par l'utilisateur, la propriété *CodeContextObject.name* aura pour valeur : « fOuSuisJe ». Par contre, si l'ouverture a été commandée par un processus inclus dans *fPrincipale*, nous aurons *CodeContextObject.name=fPrincipale*.

```
[...]  
If CodeContextObject.Name = Me.Name Then  
    'Commande dans le presse-papier  
    Me.zdtParam.SetFocus  
    'on sélectionne tout le texte  
    Me.zdtParam.SelStart = 0  
    Me.zdtParam.SelLength = Len(Me.zdtParam.Text)  
    'on copie dans le presse-papier  
    DoCmd.RunCommand acCmdCopy 'équivalent du  
    raccourci CTRL + C  
End If  
[...]
```

Par prudence, nous ajouterons une gestion d'erreur. Celle qui surviendrait, si pour une raison ou l'autre, la fonction « copier » n'est pas disponible à l'instant où le programme l'appelle.

```
GestionErreur:  
Select Case Err.Number  
    Case 2046 'pour une raison ou l'autre la  
        fonction "copier" n'est pas disponible  
        Resume Next  
    Case Else  
        MsgBox "Erreur non prévue dans Form_Timer"  
& vbCrLf _  
            & Err.Number & " : " &  
            Err.Description & "."  
End Select
```

Le code complet de l'événement « Sur minuterie » devient donc :

```
Public Sub Form_Timer()  
On Error GoTo GestionErreur  
'La position du coin sup gauche  
Me.zdtDroitePix = PositionFormPx(Me.Name).Droite  
Me.zdtBasPix = PositionFormPx(Me.Name).Bas  
Me.zdtDroiteTwi = Me.zdtDroitePix *  
Me.ZdtNbreTwipspixelHor  
Me.zdtBasTwi = Me.zdtBasPix *  
Me.ZdtNbreTwipspixelVer  
'La Largeur et la hauteur  
Me.zdtLargeurTwi = Me.WindowWidth  
Me.zdtHauteurTwi = Me.WindowHeight  
Me.zdtLargeurPix = Me.zdtLargeurTwi /  
Me.ZdtNbreTwipspixelHor  
Me.zdtHauteurPix = Me.zdtHauteurTwi /  
Me.ZdtNbreTwipspixelVer  
Me.zdtParam = "DoCmd.MoveSize " & Me.zdtDroiteTwi  
& " , " & Me.zdtBasTwi & " , " _  
            & Me.zdtLargeurTwi & " , "  
& Me.zdtHauteurTwi  
  
' Si exécution à partir d'un autre formulaire, on  
n'effectue pas  
            'la copie presse-  
papiers de la zone de texte  
If CodeContextObject.Name = Me.Name Then  
    'Commande dans le presse-papier  
    Me.zdtParam.SetFocus  
    'on sélectionne tout le texte  
    Me.zdtParam.SelStart = 0  
    Me.zdtParam.SelLength = Len(Me.zdtParam.Text)  
    'on copie dans le presse-papier  
    DoCmd.RunCommand acCmdCopy 'équivalent du  
    raccourci CTRL + C  
End If  
Exit Sub  
GestionErreur:  
Select Case Err.Number  
    Case 2046 'pour une raison ou l'autre la  
        fonction "copier" n'est pas disponible  
        Resume Next  
    Case Else  
        MsgBox "Erreur non prévue dans Form_Timer"  
& vbCrLf _  
            & Err.Number & " : " &  
            Err.Description & "."  
End Select  
End Sub
```

Ceci fait, traduisons en code les manœuvres que nous avons faites au § 8. B.

D'abord la largeur de l'écran.

Nous ouvrons *fOusuisJe* en mode création, pour supprimer ses bordures.

Ensuite, nous l'ouvrons et maximisons la fenêtre et dans la foulée (pour ne pas devoir attendre une seconde), nous déclenchons le code de la minuterie. C'est pour cela que nous la voulions « publique ».

Nous relevons alors la largeur et la mémorisons dans la variable *gEcran.Largeur*. Et terminons sans sauvegarder : *fOusuisJe* reste donc intact.

```
Private Sub Ecran()  
'La Largeur  
DoCmd.OpenForm "fOusuisJe", acDesign  
Forms!fOusuisJe.BorderStyle = 0  
DoCmd.OpenForm "fOusuisJe", acNormal  
DoCmd.Maximize  
Forms!fOusuisJe.Form_Timer  
gEcran.Largeur = Forms!fOusuisJe.zdtLargeurTwi  
DoCmd.Close acForm, "fOusuisJe", acSaveNo  
DoCmd.Restore
```

Et on refait la même opération, pour la hauteur.

```
DoCmd.OpenForm "fOusuisJe", acDesign  
Forms!fOusuisJe.BorderStyle = 0  
DoCmd.OpenForm "fOusuisJe", acNormal  
DoCmd.Minimize  
Forms!fOusuisJe.Form_Timer  
gEcran.Hauteur = Forms!fOusuisJe.zdtBasTwi +  
Forms!fOusuisJe.zdtHauteurTwi  
'Refermer sans sauver  
DoCmd.Close acForm, "fOusuisJe", acSaveNo
```

Avant de terminer, on rétablit la dimension des autres objets ouverts (entre autres : *fPrincipal*).

```
DoCmd.Restore  
End Sub
```

9. Rassemblons les pièces du puzzle

9.1. Le code du formulaire *fPrincipal*

Nous allons expliquer le code événement par événement.

Nous ne réexpliquerons pas ici comment on navigue entre l'affichage en mode formulaire, en mode création ou de l'éditeur de code. Cela a été décrit en détail au § 3.B du tutoriel précédent.

9.1.1. Sur ouverture (*Private Sub Form_Open*)

À l'ouverture du formulaire, nous allons déclencher les processus relatifs au calcul des bordures et des dimensions de l'écran.

La première fois que l'on ouvre le formulaire depuis que la base de données est ouverte, la variable globale *gBordureDispo* a sa valeur initiale : *False* (faux). Lors que

la procédure *CalculBordures* aura été exécutée *gBordureDispo* passera à *True* (vrai).

C'est la valeur constatée de *gBordureDispo* qui va nous servir de critère pour déclencher ou non l'appel des procédures *CalculBordures* et *Ecran*.

Ainsi, dans une même session, si le formulaire est ouvert plusieurs fois, les calculs ne seront lancés qu'une seule fois.

Remarquez cette instruction *Application.Echo False*, elle a pour effet de figer l'écran. Les actions qui suivent ne seront pas affichées. On rétablit la situation normale avec *Application.Echo False* quelques fractions de seconde plus tard. Cela pour éviter, à l'utilisateur, l'effet de flash désagréable. (Désolé, la suspension temporaire du flash ne fonctionne pas pour certains appareils placés le long des routes.)

```
Private Sub Form_Open(Cancel As Integer)  
If gBordureDispo = False Then  
Application.Echo False  
Call CalculBordures  
Call Ecran  
Application.Echo True  
End If  
End Sub
```

Pour vérifier que ça marche :

prenez en mode formulaire ;

dans la fenêtre d'exécution, saisissez : ? *gBordureDispo* ;

True s'affiche.

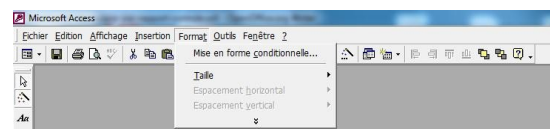
9.1.2. Sur activation (*Private Sub Form_Current*)

C'est ici que nous allons faire le nécessaire pour que le contrôle *zdtMILLESIME* s'affiche en vert dans l'enregistrement actif.

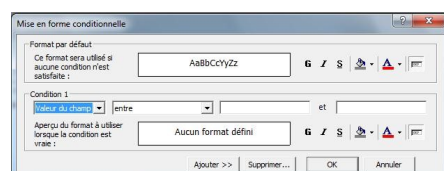
Ici aussi, on va ruser. Nous allons utiliser le format conditionnel pour colorer le contrôle uniquement dans l'enregistrement actif.

Dans *fPrincipal* ouvert en mode création, cliquez sur le contrôle *zdtMILLESIME* pour le sélectionner.

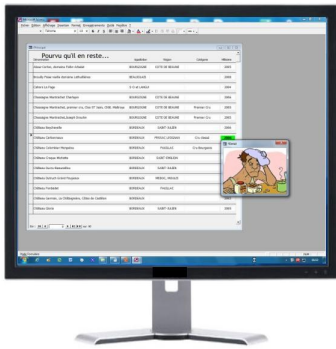
Dans la barre des menus, vous cliquez sur *Format* et ensuite *Mise en forme conditionnelle...*



Il vient :



Si vous cliquez pour dérouler le choix des conditions, vous ne trouvez pas les mots pour dire : « changer la couleur dans l'enregistrement actif ». Il y a bien « Champ activé », mais ce que nous voulons, c'est changer la couleur de *zdtMILLESIME* quel que soit le contrôle activé (lui ou un autre, ou aucun) et ce, uniquement sur la ligne qui correspond à l'enregistrement en cours, comme ceci :



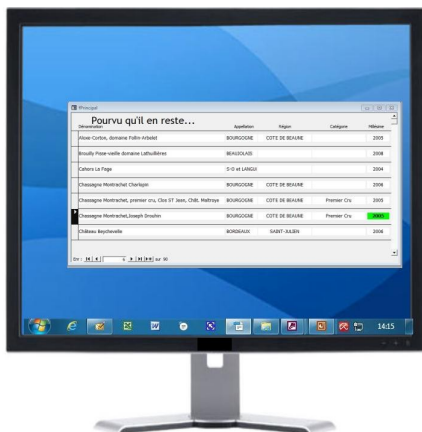
Voici une astuce pour dire « dans l'enregistrement actif ».

Dans notre formulaire, nous allons ajouter un contrôle indépendant (pas de source) que nous appelons par exemple *zdtActif*. Et nous le rendons invisible parce qu'il n'a pas d'intérêt pour l'utilisateur.

À chaque changement d'enregistrement, nous allons systématiquement copier dans ce contrôle, la valeur que nous savons unique, d'un autre contrôle. Par exemple la valeur de *zdtDENOMINATION* (*DENOMINATION* étant la clé de *tVins*, sa valeur est forcément unique).

Avec ce subterfuge le seul cas où on a égalité entre *zdtDENOMINATION* et *zdtActif*, c'est dans l'enregistrement actuellement actif !

Dès lors, nous exprimons la condition pour le formatage particulier comme ceci :



Et pour l'événement « Sur activation » nous codons ceci :

```
Private Sub Form_Current()
Me.zdtActif = Me.zdtDENOMINATION
End Sub
```

Vérifions que ça marche :

fermez *fPrincipal* en le sauvegardant ;

ouvrez-le à nouveau et naviguez entre les enregistrements pour constater le comportement du contrôle qui affiche le millésime, en vert dans le seul enregistrement actif.

9.1.3. Double-clic sur *zdtMILLESIME* (Private Sub *zdtMILLESIME_DblClick*)

Il s'agit de commander l'ouverture du formulaire *fDetail*.

Juste une précaution : on referme d'abord *fDetail* au cas où il aurait déjà été précédemment ouvert. Ceci pour provoquer le déclenchement de son événement sur ouverture que nous verrons au § suivant.

Remarquez aussi la gestion d'erreur. Vous comprendrez mieux lorsque nous passerons en revue le code de *fDetail*.

```
Private Sub zdtMILLESIME_DblClick(Cancel As Integer)
On Error GoTo GestionErreur
'Fermer le formulaire fDetail s'il est encore ouvert (appel précédent)
If CurrentProject.AllForms("fDetail").IsLoaded Then DoCmd.Close acForm, "fDetail"
'Ouvrir le formulaire fDetail
DoCmd.OpenForm "fDetail"
Select Case Err.Number
Case 2501
Resume Next
Case Else
MsgBox "Erreur inattendue dans ""Sub zdtMILLESIME_DblClick"" _
& vbLf & Err.Number & " : " & Err.Description & "."
End Select
End Sub
```

9.1.4. Sur fermeture (Private Sub *Form_Close*)

Si on ferme *fPrincipal*, *fDetail* n'a plus de raison de rester ouvert. On le referme si c'était le cas.

```
Private Sub Form_Close()
'Fermer le formulaire fDetail s'il est encore ouvert (appel précédent)
If CurrentProject.AllForms("fDetail").IsLoaded Then DoCmd.Close acForm, "fDetail"
End Sub
```

9.2. Le code du formulaire *fDetail*

9.2.1. Sur ouverture (Private Sub *Form_Open*)

Il s'agit « simplement » de calculer les paramètres de la méthode *MoveSize* de l'objet *DoCmd*.

Nous définissons une variable pour chacun de ces paramètres :

```
Private Sub Form_Open(Cancel As Integer)
Dim lDroite As Long
Dim lBas As Long
Dim lLargeur As Long
Dim lHauteur As Long
```

Nous allons nous assurer que *fDetail* a bien la propriété *Fen indépendante* = oui. C'est à cette condition que les paramètres de *doCmd.MoveSize* s'expriment par rapport au

bord supérieur gauche de l'écran.

Si ce n'est pas le cas, on affiche un message à l'utilisateur et on interrompt le processus d'ouverture de *fDetail*.

```
If Not Me.PopUp Then
    MsgBox "Le formulaire à positionner doit être
en fenêtre indépendante" & _
        vbCrLf & "(onglet Autre dans les
propriétés du formulaire)", vbInformation
    Cancel = True
    Exit Sub
End If
```

Si on fixe la valeur de *Cancel* à *True*, une erreur n° 2501 sera levée dans l'exécution du code qui avait demandé l'ouverture. Voir *Sub zdtMILLESIME_DbClick* de *fPrincipal*.

Pour la largeur et la hauteur, nous fixons les valeurs arbitraires de notre choix :

```
lLargeur = 4245
lHauteur = 3495
```

Dans un premier temps nous calculons les paramètres *Droite* et *Bas*, comme s'il y avait la place disponible à l'écran. On traduit donc en code les distances que nous avons montrées aux § 5 A et 5 B en utilisant la syntaxe montrée au § 6 :

```
'Calcul du paramètre Droite
lDroite = PositionFormPx("fPrincipal").Droite *
gPixel2Twip.Horizontal _
        + gBordure.Laterale _
        + Forms!
fPrincipal.CurrentSectionLeft _
        + Forms!
fPrincipal.zdtMILLESIME.Left
'Calcul du paramètre Bas
lBas = PositionFormPx("fPrincipal").Bas *
gPixel2Twip.Vertical _
        + gBordure.Superieure _
        + Forms!
fPrincipal.CurrentSectionTop _
        + Forms!
fPrincipal.zdtMILLESIME.Top _
        + Forms!
fPrincipal.zdtMILLESIME.Height
```

Nous vérifions ensuite qu'il y a place à l'écran et nous rectifions si nécessaire :

```
'c'est trop bas si détail est en dehors de
l'écran
If lBas + lHauteur > gEcran.Hauteur Then
    lBas = lBas - Forms!
fPrincipal.zdtMILLESIME.Height - lHauteur
End If
'c'est trop à droite si détail est en dehors de
l'écran
If lDroite + lLargeur > gEcran.Largeur Then
    lDroite = lDroite + Forms!
fPrincipal.zdtMILLESIME.Width - lLargeur
End If
```

Et on ouvre le formulaire à l'endroit précis :

```
DoCmd.MoveSize lDroite, lBas, lLargeur, lHauteur
End Sub
```

Pensez à compiler votre code.

Dans le menu de l'éditeur de code, cliquez sur Débogage et ensuite sur Compiler.

10. La preuve que ça fonctionne quelle que soit la section qui héberge le contrôle

Ouvrez le formulaire *fPrincipal* en mode création, sélectionnez le contrôle *zdtMILLESIME* et faites-le glisser dans la section *Entête du formulaire*. Passez en mode formulaire et essayez.

De même si le contrôle se trouve dans la section *Pied du formulaire*.

11. Conclusion

On y est, *fdetail* s'affiche en entier à proximité du millésime cliqué, quelle que soit la position de *fPrincipal*.

C'était un des objectifs, mais peut-être pas le principal. Mon but était en fait de vous mettre le pied à l'étrier et vous montrer qu'il n'est pas tellement difficile d'apprendre à programmer, cela vous ouvre de nouvelles perspectives pour tirer meilleur parti de votre logiciel Access.

12. La base de données

La base de données qui a servi à construire ce tutoriel se trouve ici : [Lien 41](#).

[Retrouvez l'article de Claude Leloup en ligne : Lien 42](#)

Import de données d'un serveur IBM AS/400 dans Access

Méthode pour importer des données d'un serveur IBM AS/400 dans une table Access et pouvoir ainsi faire des analyses personnalisées.

1. Introduction

Certaines entreprises possèdent l'ensemble de leurs données sur serveur AS/400. Système ô combien hermétique, mais ô combien fiable.

Mais il arrive régulièrement qu'il y ait ponctuellement des analyses complémentaires à faire ne nécessitant pas forcément l'intervention de développeurs sur l'AS/400.

Certes on peut toujours faire des queries mais ce n'est pas

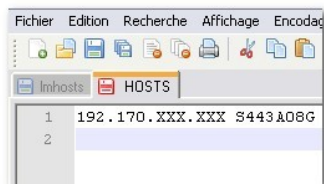
d'une convivialité folle.

Une solution parmi d'autres, est de se connecter au serveur et de rapatrier les données nécessaires dans une table Access ce qui permet de concilier les données AS/400 avec les commodités de la bureautique et les possibilités de traitement d'Access.

2. Petit rappel sur l'AS/400

L'architecture est basée sur des **bibliothèques** (que l'on peut assimiler à des répertoires), dans lesquelles sont stockés des **fichiers** (tables) avec différentes **zones** (champs). Il est défini par un nom du type 'S443A08G' ou par son adresse IP (192.170.xxx.xxx) par exemple.

Pour que la correspondance entre ces deux données soit effective, il faudra ajouter une ligne dans le fichier hosts du poste.



Ce fichier pour un poste XP Pro est présent dans : \Windows\System32\Drivers\Etc

3. Définition de l'application

Pour étayer notre propos, nous allons simuler l'importation dans Access du chiffre d'affaires de la veille fait par un vendeur donné (dont le code est « V12 »).

Sur l'AS/400 les données sont stockées dans une bibliothèque et deux fichiers.

La bibliothèque s'appelle "GESTION".

Le premier fichier appelé "VENTES1" comprend entre autres les zones suivantes :

Nom zone	Description	Type	Nb caractères
NOBON	N° du bon	Char	8
COVEN	Code vendeur	Char	4
BONDA	Année du bon	Char	2
BONDM	Mois du bon	Char	2
BONDJ	Jour du Bon	Char	2
COCLI	Code client	Char	8
MOBON	Montant du bon	Packed	11

Le deuxième fichier appelé "VENDEUR" comprend entre autres les zones suivantes :

Nom zone	Description	Type	Nb caractères
COVEN	Code vendeur	Char	4
NOVEN	Nom vendeur	Char	20

Voilà donc où sont stockées les informations dont nous avons besoin.

4. Mise en place de l'application

Pour ce faire, nous pouvons utiliser deux techniques différentes.

4.1. Solution sans ODBC et avec ADO

Cette solution a l'avantage d'être plus facile à déployer, mais nécessite plus de code VBA.

4.1.1. Prérequis

Le prérequis technique est simplement d'avoir sur le PC, le driver IBM de ClientAccess.

Pour avoir accès aux données nous allons nous servir du composant ADO (ActiveX Data Object).

Je vous recommande donc de lire les tutoriels consacrés au sujet ici : [Lien 43](#).

4.1.2. Mise en place de la table et du formulaire de paramétrage de connexion

Ces paramètres sont ceux de l'adresse du serveur et les identifiants et mot de passe du profil autorisé sur l'AS/400. En tout premier lieu et afin d'éviter des problèmes de maintenance en cas de changement d'AS/400 ou de changement d'utilisateur nous devons avoir un moyen simple de stocker ces informations.

Cela permettra de se connecter de façon automatique, lors d'opérations programmées par exemple la nuit.

Entre autres méthodes, un moyen simple consiste à créer une table "Tbl_Parametres".

Nous mettrons la propriété "Masque de saisie" du champ "Password" sur "Mot de passe".

Puis nous créons le formulaire correspondant :

Là aussi nous mettrons la propriété "Masque de saisie" de la zone de texte "Password" sur "Mot de passe".

La propriété "Ajout autorisé" du formulaire doit être à Non.

Enregistrez puis ouvrez votre formulaire pour saisir vos paramètres :



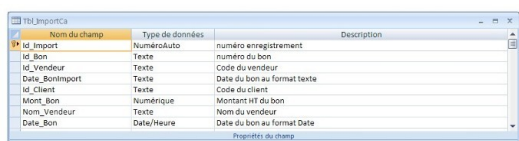
Nous avons donc nos paramètres de connexion à jour et en cas de modifications, il n'y aura pas besoin de retourner dans les pages de code.

4.1.3. Création de la structure de la table Access

Nous allons maintenant créer la table "Tbl_ImportCa" qui va nous servir à stocker les informations de l'AS/400. Ouvrons donc une nouvelle table en mode "Création" et créons les champs suivants :

Nom Zone	Description	Type	Propriété
Id_Import	Numéro d'enregistrement d'importation	Numéro auto	Clé primaire
Id_Bon	Numéro du bon	Texte	Taille champ : 10
Id_Vendeur	Code du vendeur	Char	Taille champ : 4
Date_BonImport	Date du bon au format texte	Char	Taille champ : 6
Id_Client	Code du client	Char	Taille champ : 8
Mont_Bon	Montant du bon	Numérique	Réel double
Nom_Vendeur	Nom du vendeur	Texte	Taille champ : 20
Date_Bon	Date du bon au format date	Date/heure	Format : jj/mm/aa

Ce qui nous donne dans le générateur :



Deux constatations :

- les champs de type Texte doivent avoir la même taille que sur le fichier AS/400 ;
- la création de deux champs Date :
 l'un "Date_BonImport" est destiné à rapatrier les données Année, Mois, Jour sous format Texte par concaténation ;
 l'autre "Date_Bon" à convertir le premier en format Date.

Si nous voulions faire directement l'importation dans un champ Date, nous aurions une erreur « incompatibilité de type ». Nous verrons plus tard comment faire la conversion.

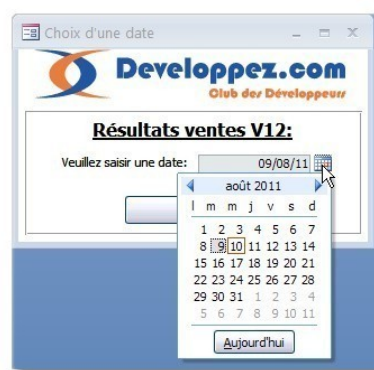
4.1.4. Création du formulaire d'accueil

Nous allons maintenant créer un formulaire d'accueil "Frm_Accueil".

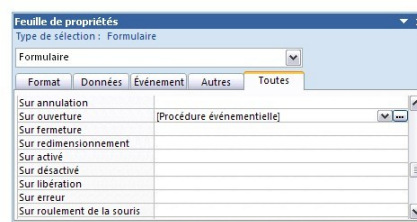
Dans ce formulaire, nous mettrons une zone de texte indépendante "Txt_DateBon" avec la propriété de format jj/mm/aa nous permettant de saisir la date à laquelle nous voulons avoir les résultats.



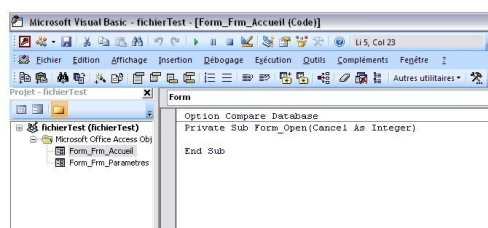
En mode formulaire nous pouvons donc saisir une date ou en choisir une en nous aidant du calendrier intégré.



Néanmoins comme nous voulons *a priori* avoir les résultats du dernier jour d'activité, et que l'entreprise travaille du lundi au samedi compris, il faut créer une fonction de remplissage automatique du champ sur l'événement "Ouverture" du formulaire. Pour cela en mode création, nous allons sur les propriétés du formulaire :



Le fait de cliquer sur le bouton en bout de ligne avec les trois petits points va nous ouvrir la fenêtre Microsoft Visual Basic sur l'événement que nous voulons renseigner.



Nous pouvons donc compléter le code ainsi :

```
Option Compare Database

Private Sub Form_Open(Cancel As Integer)
' Si la date du jour = lundi la date dans le
contrôle doit être un samedi et non un dimanche
    If Weekday(Now) = vbMonday Then
        Me.Txt_DateBon.Value = DateAdd("d", -2,
Date)
    Else
        ' Sinon la veille
        Me.Txt_DateBon.Value = DateAdd("d", -1,
Date)
        ' On pourrait également inclure les jours
fériés
    End If
End Sub
```

Enregistrons et ouvrons notre formulaire :

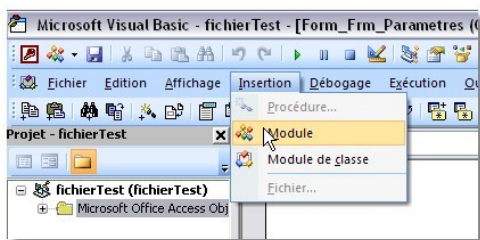


Alors que nous sommes le 10/08/11, la zone de texte affiche bien le 09/08/11. Nous aurions été le 08/08/11 (un lundi) la zone de texte aurait affiché le 06/08/11.

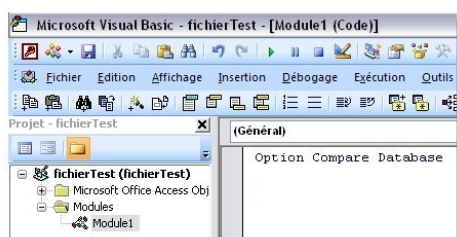
4.1.5. Module d'importation

Nous allons maintenant nous attacher au code nécessaire au remplissage de notre table.

Pour cela, ouvrons l'éditeur VBA (Alt + F11) et insérons un module.



Dans l'explorateur de projet (Ctrl + R) nous avons bien un nouveau module "Module1".



Positionnons la souris sur "Module1" et ouvrons la fenêtre de propriétés (F4).

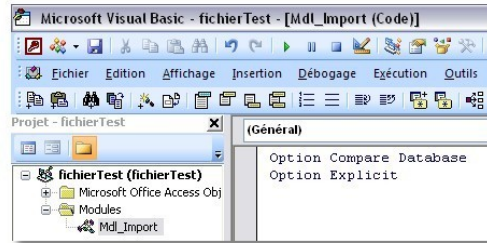
Renommons "Module1" en "Mdl_Import".

Dans l'explorateur double-cliquons sur "Mdl_Import".

Nous sommes maintenant dans la fenêtre de saisie du code correspondant au module.

Si elle n'est pas présente nous ajoutons la ligne "Option Explicit".

Cette option nous obligera à déclarer toutes nos variables, et avoir ainsi un code plus performant.



Il ne nous reste plus qu'à saisir le code de la procédure.

Procédure Import_Ca

```
Option Compare Database
Option Explicit

-----
Public Sub Import_Ca()
On Error GoTo Error_Import_Ca

'Variables de connexion ADO
    Dim CnnAs400 As ADODB.Connection
    Dim RsAs400 As ADODB.Recordset
    Dim CnnDb As New ADODB.Connection
    Dim RsDb As New ADODB.Recordset
    Dim Fld As ADODB.Field
    Dim StrNameAS400 As String          'Variable
nom de l'AS/400
    Dim StrNameUser As String          'Variable
nom utilisateur
    Dim StrPasswordUser As String      'Variable
mot de passe utilisateur
    Dim Champ1 As String, Champ2 As String,
Champ3 As String, Champ4 As String, Champ5 As
String, Champ6 As String

'Autres variables
    Dim StrDate As String              'Variable
Date de bon
    Dim strTabDelete As String
    Dim strSQL As String
    Dim i As Integer

    ' Attribue des valeurs aux variables

    ' Transforme le contrôle date du formulaire
d'accueil (par exemple 24/10/06) en texte de 6
caractères (061024)
    StrDat = Right(Form_Frm_Accueil.
[Txt_DateBon], 2) & Mid(Form_Frm_Accueil.
[Txt_DateBon], 4, 2) & Left(Form_Frm_Accueil.
[Txt_DateBon], 2)

    ' Les trois variables suivantes vont chercher
leur valeur dans la table « Paramètres »
    StrNameAS400 = Nz(DLookup("Name_AS400",
"Tbl_Parametres"))
    StrNameUser = Nz(DLookup("User",
```

```

"Tbl_Parametres"))
    StrPasswordUser = Nz(DLookup("Password",
"Tbl_Parametres"))

' Nous supprimons les données de la table
"Tbl_Import_Ca"
DoCmd.SetWarnings False 'Stoppe les
messages d'alerte
strTabDelete = "DELETE * FROM
[Tbl_ImportCa];"
DoCmd.RunSQL strTabDelete 'Rétablit les
messages d'alerte
DoCmd.SetWarnings True

' Nous lançons la connexion.
Set CnnAs400 =
CreateObject("ADODB.connection")
CnnAs400.Open "provider=IBMDA400;data
source=" & StrNameAS400 & "", StrNameUser,
StrPasswordUser

Set Cnndb = CurrentProject.Connection
Set RsAs400 = CreateObject("ADODB.recordset")
RsAs400.ActiveConnection = CnnAs400

'Nous créons la requête.
strSql = " " & _
" SELECT T01.NOBOB, T01.COVEN,
T01.BONDA||T01.BONDM|T01.BONDJ, T01.COCLI,
T01.MOBON, T02.NOVEN " & _
" FROM GESTION.VENTES1 T01 " & _
" JOIN GESTION.VENDEUR T02 " & _
" ON T01.COVEN = T02.COVEN " & _
" WHERE (T01.BONDA||T01.BONDM||
T01.BONDJ = '" & StrDate & "' AND " & _
" T01.COVEN = 'V12 ')"

'Pour avoir notre champ Date1 nous faisons
une concaténation sur l'année, le mois et le
jour.

RsAs400.Open strSql

Do Until RsAs400.EOF
i = 1
For Each fld In RsAs400.Fields
Select Case i
Case 1
Champ1 = Nz(fld.Value)
Case 2
Champ2 = Nz(fld.Value)
Case 3
Champ3 = Nz(fld.Value)
Case 4
Champ4 = Nz(fld.Value)
Case 5
Champ5 = Nz(fld.Value)
Case 6
Champ6 = Nz(fld.Value)
Case Else
End Select
i = i + 1
Next fld
If Rsdb.State = 0 Then
' Ouverture de la table et
remplissage
Rsdb.Open "[Tbl_ImportCa]", Cnndb,
adOpenKeyset, adLockOptimistic
End If
' Attribution des valeurs aux champs
correspondants

```

```

With Rsdb
.AddNew Array("N°_Bon", "Code_Ven",
"Date1", "Code_client", "CA_Bon", "Nom_Ven"), _
Array(Champ1, Champ2, Champ3,
Champ4, Champ5, Champ6)
.Update
End With
RsAs400.MoveNext
Loop

'La table est remplie et il faut maintenant
s'occuper du dernier champ qui nous intéresse
'c'est-à-dire le champ "Date_Bon". Nous
allons nous servir de la fonction Update.
DoCmd.SetWarnings False 'Stoppe les
messages d'alerte
DoCmd.RunSQL "Update Tbl_Import_Ca Set
Date_Bon " & _
"= Right([Date1],2) & '/' &
Mid([Date1],3,2) & '/' & Left([Date1],2)"
DoCmd.SetWarnings True

Exit_Import_Ca:
'Ferme la connexion et libère les variables
Rsdb.Close
RsAs400.Close
Call CloseConnection(Cnndb)
Call CloseConnection(CnnAs400)
Set CnnAs400 = Nothing
Set Cnndb = Nothing
Set RsAs400 = Nothing
Set Rsdb = Nothing
Exit Sub

Error_Import_Ca:
MsgBox "Erreur importation " & Err.Number & " "
& Err.Description
Resume Exit_Import_Ca

End Sub
-----
-----
Sub CloseConnection(Cnx As ADODB.Connection)
With Cnx
If .State = adStateOpen Then
.Close
End If
End With
End Sub

```

Attention

Quand vous saisissez les critères d'une zone « **Char** » dans une requête, il faut bien tenir compte du nombre de caractères demandés.

Dans notre exemple le code vendeur 'V12' compte trois caractères, alors que la zone en prévoit quatre.

Il faut donc dans le critère mettre un espace supplémentaire entre les deux quotes, ce qui donne: « T01.COVEN = 'V12 ' » et non « T01.COVEN = 'V12' ».

D'autre part, il arrive que les zones AS/400 Année, Mois, Jour ne soient pas de type « **Char** » mais de type « **Zoned** ».

Dans ce cas la requête ne marche pas et il faut changer le mode de concaténation.

variation requête

```
'Nous créons la requête.  
strSql = " " & _  
" SELECT T01.NO BON ,T01.CO VEN , " & _  
" Digits (T01.BONDA) Concat  
Digits (T01.BONDM) Concat Digits (T01.BONDJ) , " & _  
" T01.CO CLI ,T01.MOBON ,T02.NO VEN " & _  
" FROM GESTION.VENTES1 T01 " & _  
" JOIN GESTION.VENDEUR T02 " & _  
" ON T01.CO VEN = T02.CO VEN " & _  
" WHERE (Digits (T01.BONDA) Concat  
Digits (T01.BONDM) Concat Digits (T01.BONDJ) = ' " &  
StrDate & "' AND " & _  
" T01.CO VEN = 'V12 ' ) "
```

4.1.6. Appel de la procédure

Il ne nous reste plus qu'à appeler la procédure à partir du bouton de validation.

Procédure Import Ca

```
Private Sub Cmd_Valid_Click()  
Call Import_Ca  
DoEvents  
MsgBox "L'importation est terminée."  
End Sub
```

4.2. Solution avec ODBC

À contrario de la technique vue plus haut, celle-ci nécessite peu ou pas de code, mais est plus longue à déployer sur de nombreux postes.

Nous n'aurons pas besoin de la table et du formulaire de paramètres, par contre nous garderons notre formulaire d'accueil avec le choix de date.

4.2.1. Prérequis

Le prérequis est comme dans la solution précédente simplement d'avoir sur le PC, le driver IBM de Client Access.

Nous allons dans ce cas nous servir des liaisons ODBC. Pour la configuration des liaisons, vous pouvez vous aider du tutoriel de **LedZeppII** ici : [Lien 44](#).

4.2.2. Paramètre ODBC

Comme pour tout paramètre ODBC, allons dans le panneau de configuration, puis sur "Outils d'administration" et cliquons sur "Sources de données(ODBC)".

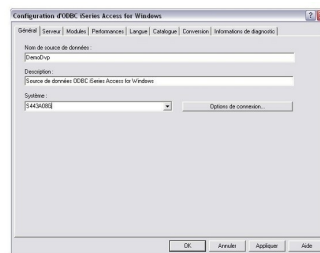
Si vous avez Vista 64 bits, Seven 64 bits ou Server 2008 64 bits avec Office 32 bits il faut passer par : C:\Windows\SysWOW 64\Odbcad32.exe.



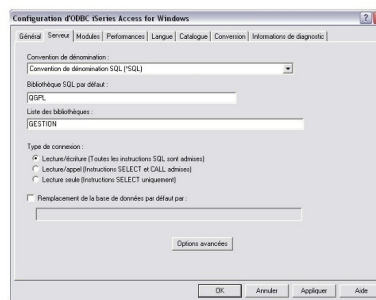
Cliquons sur "Ajouter" et choisissons le driver Client Access.



Dans l'onglet "Général", donnons un nom à notre connexion et saisissons le nom de notre système.

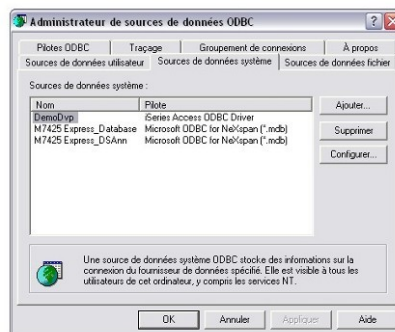


Puis dans l'onglet "Serveur", notons le nom de la ou des bibliothèques demandées.



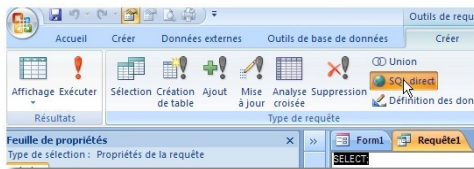
Si nous avons eu besoin de plusieurs bibliothèques, nous les aurons saisies dans la liste en les séparant par une virgule.

Validons. Notre source de données système apparaît bien dans la liste.

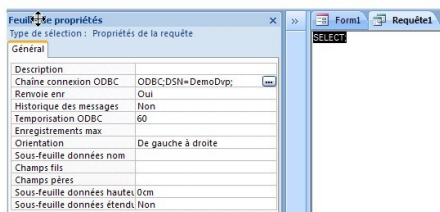


4.2.3. Requête directe

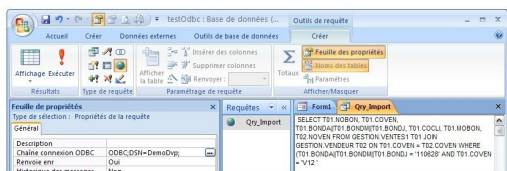
Nous allons pouvoir maintenant, envoyer dans l'AS/400 notre requête, afin d'en récupérer les données. Pour cela, passons en mode création de requête. Ne mettons aucune table source. Puis sélectionnons "SQL direct".



Dans la fenêtre de propriété, sur le paramètre "**Chaîne de connexion ODBC**" servons nous des trois petits points pour aller rechercher notre source de données machines.



Puis saisissons notre requête.



Comme vous pouvez le voir, dans la condition Where une date définie est saisie.

Dans notre exemple, cette date va être conditionnée par le formulaire d'accueil.

Nous allons donc créer une procédure de modification de la requête, afin de lui donner le critère choisi dans le formulaire.

Procédure Maj_QryImport

```
Public Sub Maj_QryImport() 'Mise à jour de la
requête Qry_Import + création des tables et
remplissage
    On Error GoTo Error_Maj_QryImport
    Dim oDb As DAO.Database
    Dim StrDat As String
    Dim oQdf As DAO.QueryDef
    Dim strSql As String

    'Attribue à la variable la valeur de la liste
déroulante du formulaire d'accueil
    StrDat = Form_Frm_Accueil.[Txt_DateBon].Value
    StrDat = Right(StrDat, 2) & Mid(StrDat, 4, 2)
    & Left(StrDat, 2)

    Set oDb = CurrentDb
    'Redéfinit la requête en y ajoutant la
variable date choisie
    Set oQdf = oDb.QueryDefs("Qry_Import")
    strSql = " " & _
        " SELECT T01.NO BON, T01.COVEN,
```

```
T01.BONDA||T01.BONDM||T01.BONDJ, T01.COCLI,
T01.MOBON, T02.NOVEN " & _
    " FROM GESTION.VENTES1 T01 " & _
    " JOIN GESTION.VENDEUR T02 " & _
    " ON T01.COVEN = T02.COVEN " & _
    " WHERE (T01.BONDA||T01.BONDM||
T01.BONDJ = ' " & StrDat & "' AND " & _
    " T01.COVEN = 'V12 ')"
```

```
'Applique la modification
oQdf.SQL = strSql
```

Exit_Maj_QryImport:

```
oDb.Close
Set oDb = Nothing
Set oQdf = Nothing
Exit Sub
```

Error_Maj_QryImport:

```
If Err.Number = 3265 Then
    MsgBox "La requête n'existe pas"
Else
```

```
    MsgBox Err.Number & " " &
```

Err.Description

```
End If
```

```
Resume Exit_Maj_QryImport
```

End Sub

Après il suffira avec l'assistant:

- soit de faire une requête de création de table à partir de "Qry_Import";
- soit de faire une requête "Ajout" pour alimenter une table déjà créée.

5. Conclusion

Voilà notre table Access est alimentée et nous pouvons maintenant créer les requêtes, formulaires et états souhaités avec toutes les possibilités offertes par Access.

Certaines extractions plus importantes peuvent être programmées la nuit, en faisant appel à une procédure lancée au démarrage de l'application.

Ceci est un exemple très simple qui peut bien sûr être complété en fonction des besoins. Par exemple :

- centraliser le fichier de paramètres sur un serveur ;
- introduire des variables pour les bibliothèques ;
- travailler sur une période avec une date de départ et d'arrivée ;
- alimenter une table vendeurs qui elle-même servira de source à une zone de liste en y joignant une variable ;
- ...

Comme vous le voyez les possibilités sont nombreuses, et vous pourrez choisir la technique qui vous convient le mieux.

Retrouvez l'article de Jean-Damien Gayot en ligne : [Lien 45](#)

Android

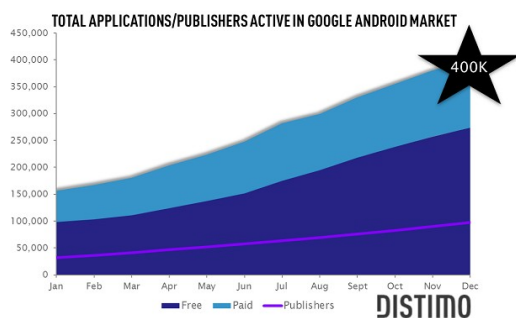
Les dernières news



Android Market : 400.000 applications actives

100.000 développeurs et succès du modèle Freemium

D'après Distimo, l'Android Market a passé la barre des 400.000 applications actives le 31 décembre. En octobre, une autre étude ([Lien 46](#)) créditait la galerie de 500.000 applications, mais prenait en compte le nombre de publications (autrement dit, elle comptait également celles qui ensuite avaient été supprimées).



Les applications gratuites représentent les deux tiers du catalogue (68%), une proportion qui ne cesse de grandir (elles étaient 60% en avril).

Distimo explique cette croissance du « gratuit » par le succès du modèle « freemium » auprès des développeurs. Un modèle économique qui associe une offre gratuite, en libre accès, à une offre « Premium », plus haut de gamme et payante (des niveaux supplémentaires dans un jeu par exemple).

Le rythme de croissance de la galerie dans son ensemble connaît également une accélération, comme le montre bien le graphique ci-dessus. La raison tiendrait au fait, d'après Distimo, que la plateforme a passé une autre barre symbolique, celle des 100.000 développeurs en activité sur Android.

Petit bémol, il y a un an, ces développeurs publiaient en moyenne 5 applications par année contre 4,1 aujourd'hui. Le bémol peut être cependant le signe que les applications prennent plus de temps à être conçues et qu'elles sont donc de meilleure qualité. On peut en tout cas l'espérer.

De son côté, Apple revendique plus de 500.000 applications dans ses communications commerciales sur l'App Store.



The App Store
There's an app for that.
Over 500,000, actually.

Every app you download from the App Store makes your iPhone do even more. And with hundreds of thousands of apps to choose from, we mean a whole lot more.



Quant au Marketplace de Windows Phone, il aurait dépassé les 50.000 applications et les 13.000 développeurs le 28 décembre : [Lien 47](#).

Commentez cette news de Gordon Fowler en ligne : [Lien 48](#).

Android : Google lance un nouveau portail Pour aider les développeurs dans la création d'interfaces utilisateur réussies

Google souhaite voir des applications Android sur sa galerie ayant des interfaces utilisateur semblables ou meilleures que celles de leur équivalent sur iOS.

Les développeurs d'applications pour Android ont été longtemps préoccupés par la fragmentation de la plateforme qui pouvait entraîner le changement de l'interface utilisateur d'une application par celle de la couche du constructeur.

Google, dans un souci de réduire cette fragmentation, avait apporté une refonte complète à son système d'exploitation avec la publication d'Android 4.0 alias Ice Cream Sandwich, qui unifie version mobile et tablette de l'OS.

Bien plus, l'éditeur qui jusqu'ici offrait une grande liberté aux constructeurs pour adapter l'OS à leurs terminaux, a contraint ceux-ci pour Android 4, à intégrer obligatoirement le thème Holo, offrant ainsi un meilleur contrôle de l'interface utilisateur pour le développeur.

Afin d'aider les développeurs à profiter de ces avancées pour la conception des applications ayant des interfaces utilisateur réussies, Google a lancé un nouveau portail « Android Design ».



Ce portail sera un endroit où les développeurs pourront en

apprendre davantage sur les principes, les bonnes pratiques, les blocs de code et les modèles pour la création de meilleures interfaces utilisateur pour Android.

« L'équipe Android User Experience s'engage à vous aider à concevoir des applications étonnantes que les gens aiment, et ce n'est que le début. Dans les prochains mois, nous allons élargir la conception pour Android avec des contenus plus approfondis », note l'éditeur.

Une bonne nouvelle donc pour les développeurs Android.

Le portail **Android Design** : [Lien 49](#)

Commentez la news de **Hinault Romaric** en ligne : [Lien 50](#)

Les dernier tutoriels et articles

Introduction à greenDroid

Dans ce tutoriel, je vais vous présenter une bibliothèque que j'ai souvent utilisée et qui est très utile pour vos développements Android, il s'agit de GreenDroid créé par Cyril Mottier : [Lien 51](#).



1. GreenDroid : à quoi ça sert ?

Cette bibliothèque facilitera la création de vos applications Android, grâce à plusieurs mécanismes, elle vous permettra :

- d'optimiser vos applications ;
- de gagner du temps sur la création d'interfaces et la gestion de plusieurs fonctionnalités de votre application ;
- de factoriser votre code.

- Package : `com.nazim.tuto.gd`.
- Activité de départ : `GDIntroActivity`.

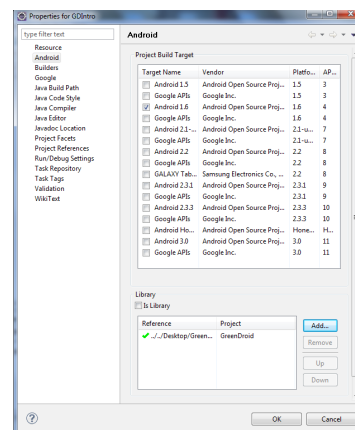
Une fois le projet créé, faites un clic droit sur votre projet, allez dans Properties puis Android et dans Library. Cliquez sur Add puis sélectionnez **GreenDroid** (si GreenDroid n'apparaît pas, c'est que l'étape "Is Library" n'est pas correctement faite).

2. GreenDroid : les fonctionnalités disponibles

Voici une introduction aux fonctionnalités de **GreenDroid**, la liste est beaucoup plus grande que ce que je présente dans cet article.

3. GreenDroid : installation

Pour commencer, vous devez cloner (**git clone <https://github.com/cyrimottier/GreenDroid.git>**) ou télécharger GreenDroid sur : **<https://github.com/cyrimottier/GreenDroid>**, puis importer le projet dans votre eclipse (File -> Import). Une fois cette étape effectuée, cliquez droit sur le projet GreenDroid puis allez dans Properties puis dans Android et vérifiez que "Is Library" est bien coché sinon faites-le.



4. GreenDroid : mise en place du projet GreenDroid

Afin de mettre en place votre projet, il faut faire quelques modifications pour intégrer **GreenDroid**.

Nous allons créer un projet qui va utiliser GreenDroid.

- Nom du projet : `GDIntro`.
- Android SDK : 1.6.
- Nom de l'application : `GD Intro`.
- Premièrement, nous allons créer une classe qui présentera l'application et qui s'appellera **GDIntroApp**. Elle contiendra une seule méthode

« **getHomeActivityClass** » qui retournera votre vue de base :

```
package com.nazim.tuto.gd;

import greendroid.app.GDApplication;

public class GDIntroApp extends GDApplication {

    @Override
    public Class getHomeActivityClass() {
        return GDIntroApp.class;
    }
}
```

et finalement, il ne reste plus qu'à modifier le manifest, pour rajouter :

- notre classe **GDIntroApp** : `android:name= »GDIntroApp »` ;
- rajouter le thème **GreenDroid** : `android:theme= »@style/Theme.GreenDroid »`.

Ce qui donnera

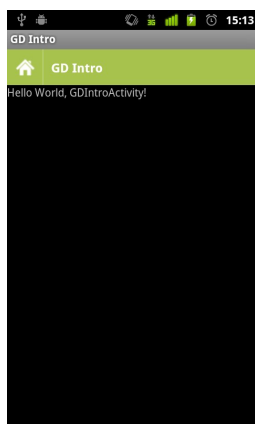
```
<?xmlversion="1.0"encoding="utf-8"?>
<manifest
xmlns:android="http://schemas.android.com/apk/res
/android"
    package="com.nazim.tuto.gd"
    android:versionCode="1"
    android:versionName="1.0">

    <application
android:icon="@drawable/icon"android:label="@string/app_name"
    android:theme="@style/Theme.GreenDroid"android:
name="GDIntroApp">
        <activity android:name=".GDIntroActivity"
            android:label="@string/app_name"
">

            <intent-filter>
                <action
android:name="android.intent.action.MAIN"/>
                <category
android:name="android.intent.category.LAUNCHER"/>
            </intent-filter>
        </activity>

    </application>
</manifest>
```

Ce qui vous donnera :



5. GreenDroid : Action Bar

L'une des fonctionnalités majeures de **GreenDroid** est **l'ActionBar**, qui correspond à la barre verte qui est sur le screen au-dessus. Nous allons l'enrichir avec plusieurs fonctionnalités.

Nous allons rajouter à notre Action Bar les boutons suivants :

- un bouton pour rafraîchir ;
- un bouton pour partager ;
- un bouton pour localiser.

Pour cela nous allons créer une petite fonction pour initialiser les différents éléments.

```
private void initActionBar() {
addActionBarItem(Type.Locate, LOCATE);
addActionBarItem(Type.Refresh, REFRESH);
addActionBarItem(Type.Share, SHARE);
}
```

et les entiers utilisés

```
private final int LOCATE = 0;
private final int REFRESH = 1;
private final int SHARE = 2;
```

sans oublier d'appeler la méthode dans le onCreate

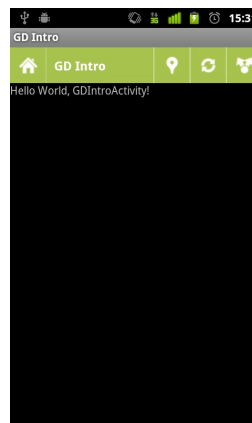
```
@Override
public void onCreate(Bundle
savedInstanceState) {
    super.onCreate(savedInstanceState);

    setActionBarContentView(R.layout.main);
    initActionBar();
}
```

Donc pour rajouter un élément, il suffit juste d'appeler `addActionBarItem`, avec comme arguments :

- le type de l'élément souhaité. Une vingtaine d'éléments sont disponibles ;
- un entier qui sera l'identifiant de votre élément quand on clique dessus.

Ce qui vous donnera :



Maintenant passons à la gestion du clic sur un élément de votre Action Bar. Pour cela il suffit de surcharger la méthode `onOptionsItemSelected`.

```

@Override
    public boolean
onHandleActionBarItemClick(ActionBarItem item,
intposition) {
    switch(item.getItemId()) {
        caseLOCATE:
            Toast.makeText(getApplicationContext(),
                "Vous avez cliqué sur le bouton
LOCATE",
                Toast.LENGTH_SHORT).show();
            break;

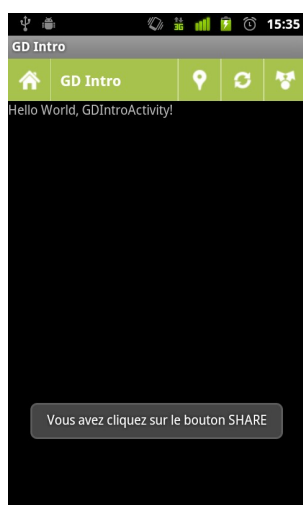
        caseREFRESH:
            Toast.makeText(getApplicationContext(),
                "Vous avez cliqué sur le bouton
REFRESH",
                Toast.LENGTH_SHORT).show();
            break;

        caseSHARE:
            Toast.makeText(getApplicationContext(),
                "Vous avez cliqué sur le bouton
SHARE",
                Toast.LENGTH_SHORT).show();
            break;
        default:
            return
super.onHandleActionBarItemClick(item, position);
    }

    returntrue;
}

```

Ce qui donnera :



Voilà pour la première fonctionnalité. Je ne pourrai pas aborder les différents cas d'utilisation. Si votre cas est spécifique, n'hésitez pas à me poser une question.

6. GreenDroid : QuickActions

Nous allons passer au **QuickActions**, vous allez comprendre à quoi cela correspond. Nous allons faire en sorte que lorsque l'utilisateur clique sur le bouton Share, cela affiche des QuickActions.

Nous allons commencer par créer une variable qui représentera notre QuickActions.

```
private QuickActionWidget quickActions;
```

Puis, nous allons créer une méthode qui initialise notre QuickActions Widget et initialise le clic listener dessus

```

private void initQuickActionBar() {
    quickActions = new QuickActionBar(this);
    quickActions.addAction(newQuickAction(th
is, R.drawable.facebook,
        "Facebook"));
    quickActions.addAction(newQuickAction(th
is, R.drawable.twitter, "Twitter"));
    quickActions.addAction(newQuickAction(th
is, R.drawable.mail, "Email"));
    quickActions.setOnQuickActionClickListener(ne
wOnQuickActionClickListener() {
        public void
onQuickActionClicked(QuickActionWidget widget,
            intposition) {
                Toast.makeText(GDIntroActivity.this,
                    "Item "+ position + " clicked",
                    Toast.LENGTH_SHORT)
                    .show();
            }
    });
}

```

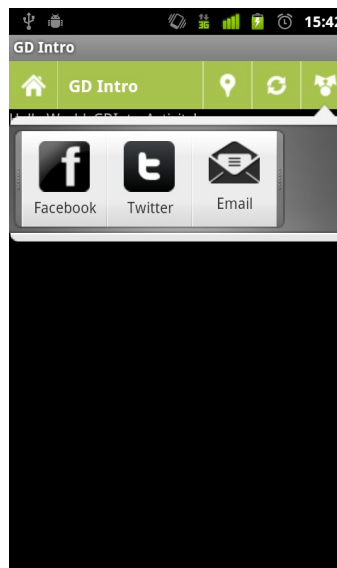
Il vous faut les trois images suivantes, à mettre dans le dossier Drawable de res : [Lien 52](#).

Nous allons modifier notre méthode **onHandleActionBarItemClick**, en cliquant sur le bouton Share

```

caseSHARE:
    quickActions.show(item.getItemView());
    Toast.makeText(getApplicationContext(),
        "Vous avez cliqué sur le bouton
SHARE",
        Toast.LENGTH_SHORT).show();
    break;

```



Sans oublier d'appeler la méthode d'initialisation dans le onCreate

```

@Override
    public void onCreate(Bundle
savedInstanceState) {
        super.onCreate(savedInstanceState);

        setContentView(R.layout.main);
    }

```

```
initActionBar();
initQuickActionBar();
}
```

Ce qui vous donnera

7. GreenDroid : surcharger le thème

Maintenant vous allez me dire **“GreenDroid c'est génial”**, mais je ne veux pas d'une barre verte dans mon application. Ne vous inquiétez pas, vous pouvez changer ça, en surchargeant le thème.

Je vais vous expliquer comment faire.

Alors la première étape est de créer votre style. Dans values, créez un fichier themes.xml. Dans ce fichier vous allez créer votre thème comme ci-dessous :

```
<?xmlversion="1.0"encoding="utf-8"?>
<resources>
  <style
name="Theme.Mytheme"parent="@style/Theme.GreenDroid.NoTitleBar">
    <item
name="gdActionBarTitleColor">#fff</item>
    <item
name="gdActionBarDividerWidth">2px</item>
    <item
name="gdActionBarBackground">@color/background</item>
  </style>
</resources>
```

Il existe plusieurs propriétés que vous pouvez surcharger de GreenDroid, ici on surcharge :

- la couleur du texte dans l'action bar ;
- la taille des séparateurs dans l'action bar ;
- la couleur de fond de l'action bar.

Vous allez sûrement remarquer le **@color**, pour celui-ci créez un fichier colors dans values qui contiendra :

```
<?xmlversion="1.0"encoding="utf-8"?>
<resources>
  <color name="background">#f00</color>
</resources>
```

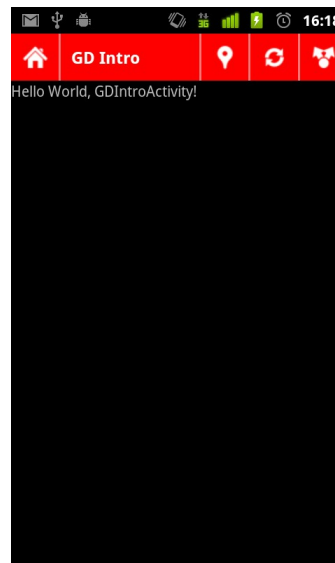
Vous pouvez à partir de maintenant prendre la bonne habitude de mettre toutes les couleurs que vous utilisez dans votre application dans ce fichier là.

Dernière étape, il faut modifier le fichier AndroidManifest pour rajouter notre thème (**android:theme= »@style/Theme.Mytheme**), ce qui donnera

```
<?xmlversion="1.0"encoding="utf-8"?>
<manifest
xmlns:android="http://schemas.android.com/apk/res/android"
package="com.nazim.tuto.gd"
android:versionCode="1"
android:versionName="1.0">
```

```
<application
android:icon="@drawable/icon"android:label="@string/app_name"
android:theme="@style/Theme.Mytheme"
android:name="GDIntroApp">
  <activity android:name=".GDIntroActivity"
android:label="@string/app_name">
    <intent-filter>
      <action
android:name="android.intent.action.MAIN"/>
      <category
android:name="android.intent.category.LAUNCHER"/>
    </intent-filter>
  </activity>
</application>
</manifest>
```

Voilà vous devriez obtenir le résultat suivant :



8. Conclusion

Voilà ce tutoriel s'arrête ici, vous pouvez télécharger le projet ici : [Lien 53](#).

Ce tutoriel fait un bref tour des fonctionnalités de **GreenDroid**. D'autres fonctionnalités sont disponibles dans cette bibliothèque. Vous trouverez plus de détails sur le site officiel : [Lien 54](#).

Parmi ces fonctionnalités :

- **SegmentedBar** ;
- **AsyncImageView** ;
- et plein d'autres fonctionnalités. N'hésitez pas à aller sur le site pour chercher si une fonctionnalité que vous souhaitez existe.

N'hésitez pas à me poser des questions et s'il y a de la demande, je ferai une suite à ce tutoriel avec les autres fonctionnalités. En attendant, j'espère que ce tutoriel vous a permis de connaître cette bibliothèque et vous a donné envie de l'utiliser. Bonne continuation à tous !

Retrouvez l'article de Nazim Benbourahla en ligne : [Lien 55](#)

Tests unitaires et tests d'IHM sur un projet Android utilisant Maven

Cet article a pour but de montrer comment créer un projet de test sur un projet Android utilisant Maven. On peut lire tout et son contraire sur la toile concernant la configuration à mettre en place.

Partant du principe que le projet fonctionne et a besoin d'être testé de manière automatique, ce tutoriel est destiné aux personnes ayant un niveau avancé sur Android.

1. Introduction

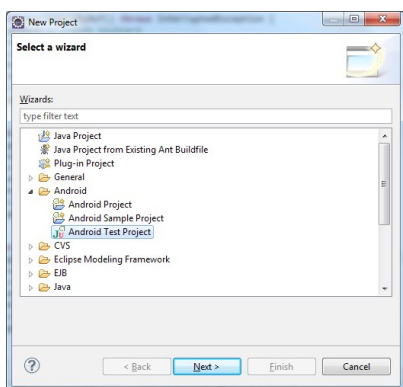
Dans ce tutoriel nous utiliserons l'EDI recommandé pour Android : Eclipse.

Supposons que vous ayez déjà un projet Android qui compile. Nous l'appellerons ici AdeoAndroidApp.

Nous utiliserons aussi Maven ([Lien 56](#)) pour gérer les dépendances (disponible en installant "Android Configurator for M2E" sur l'Eclipse Marketplace) et le déploiement, ce qui rend la manipulation plus particulière que ce qui est indiqué sur la documentation Android : [Lien 57](#).

2. Créer un projet de test

- Nous le nommerons *AdeoAndroidAppTest*.
- Choisir "Create new project in Workspace" (c'est un nouveau projet, on ne réutilise pas les sources du projet à tester).
- À l'étape suivante, choisir le projet cible. Dans notre cas, *AdeoAndroidApp*.
- Choisir ensuite la version du SDK utilisée dans le projet cible, puis valider pour quitter l'assistant.



On crée un projet de type Android Test project.

Nous avons maintenant un projet de test créé, qui cible notre projet à tester. En ouvrant le *AndroidManifest.xml* du projet, on observe :

AndroidManifest.xml

```
<instrumentation
    android:name="android.test.InstrumentationTestRunner"
    android:targetPackage="com.adeo.android.app" />
```

Preuve que le projet cible bien le bon package.

3. "Maveniser" le projet !

Il faut pour cela créer un fichier pom.xml (POM = Project Object Model) à la racine du projet.

Globalement, on peut reprendre le pom.xml du projet cible, en changeant l'artifactId du projet (AdeoAndroidAppTest) et même chose pour le nom.

Dans les dépendances, nous allons tout enlever et en renseigner trois :

- le projet cible. Maven gère la transitivité donc nul besoin de redéclarer les dépendances du projet à tester ;
- le package *android-test* qui va nous permettre d'utiliser JUnit ;
- le package *robotium-solo* qui nous permettra d'utiliser Robotium, l'outil de test d'IHM : [Lien 58](#).

Enfin, les plugins Maven (Maven Android Plugin et Compiler Plugin) et autres sont déjà renseignés dans la partie des plugins si vous avez copié/collé le *pom.xml* du projet cible, nul besoin d'y toucher.

Cela donne ce genre de fichier :

pom.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<project
  xmlns="http://maven.apache.org/POM/4.0.0"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/maven-v4_0_0.xsd">
  <modelVersion>4.0.0</modelVersion>
  <groupId>com.adeo.android</groupId>
  <artifactId>AdeoAndroidAppTest</artifactId>
  <version>1.1.2-SNAPSHOT</version>
  <packaging>apk</packaging>
  <name>AdeoAndroidAppTest</name>

  <dependencies>

    <dependency>
      <groupId>com.google.android</groupId>
      <artifactId>android-test</artifactId>
      <version>2.3.1</version>
      <scope>provided</scope>
    </dependency>
    <dependency>
      <groupId>com.adeo.android</groupId>
      <artifactId>AdeoAndroidApp</artifactId>
      <scope>provided</scope>
      <type>jar</type>
      <version>1.1.2-SNAPSHOT</version>
    </dependency>
    <dependency>
      <groupId>com.jayway.android.robotium</group
Id>
      <artifactId>robotium-solo</artifactId>
      <version>3.0</version>
    </dependency>

  </dependencies>
```

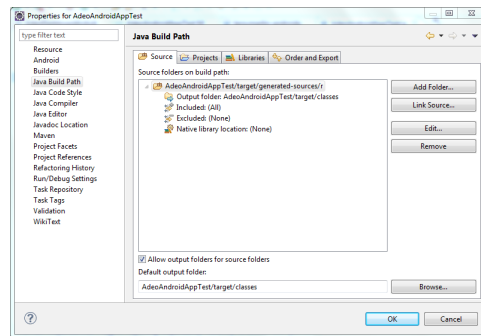
```

<build>
  <plugins>
    <plugin>
      <groupId>com.jayway.maven.plugins.android.generation2</groupId>
      <artifactId>maven-android-plugin</artifactId>
      <version>2.8.3</version>
      <configuration>
        <androidManifestFile>${project.basedir}/AndroidManifest.xml</androidManifestFile>
        <assetsDirectory>${project.basedir}/assets</assetsDirectory>
        <resourceDirectory>${project.basedir}/res</resourceDirectory>
        <sourceDirectory>${project.basedir}/src</sourceDirectory>
        <nativeLibrariesDirectory>${project.basedir}/src/main/native</nativeLibrariesDirectory>
        <deleteConflictingFiles>true</deleteConflictingFiles>
        <undeployBeforeDeploy>true</undeployBeforeDeploy>
      </configuration>
      <extensions>true</extensions>
    </plugin>

    <plugin>
      <artifactId>maven-compiler-plugin</artifactId>
      <version>2.3.2</version>
      <configuration>
        <source>1.6</source>
        <target>1.6</target>
      </configuration>
    </plugin>
  </plugins>
</build>
<properties>
</properties>
</project>

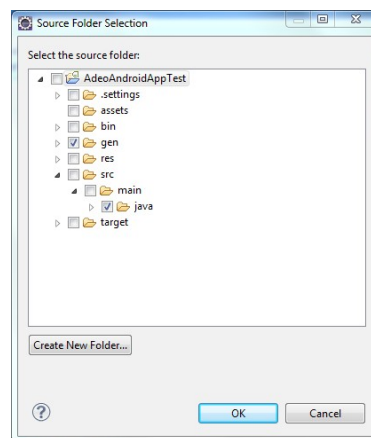
```

Aussi, nous allons éditer les chemins de build de Java afin de faire correspondre notre arborescence : **cliquez droit sur le projet de test => sélectionner Java Build Path puis supprimer les chemins déjà présents.**



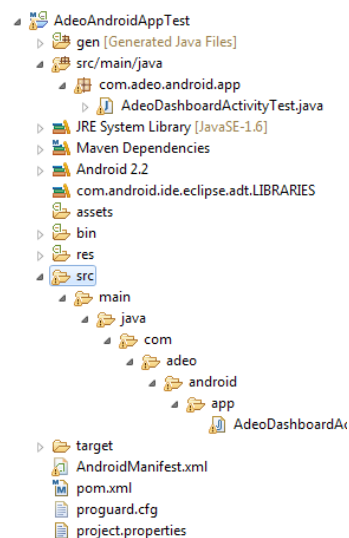
Fenêtre de chemin(s) de build

On va ajouter les dossiers `gen/` et `src/main/java`, comme sur la capture d'écran suivante :



On sélectionne les bons dossiers de build

On obtient ainsi une arborescence de ce type :



Arborescence du projet de test

À noter que les deux dépendances `android-test` et `AdeoAndroidApp` (notre projet cible) ont un scope `provided`.

De plus, on a inclus `AdeoAndroidApp` en type `jar` et non `apk`.

Après l'enregistrement du `pom.xml`, le projet n'est pas encore complètement "mavenisé". On va activer la gestion des dépendances par un **cliquez droit sur le projet de test => Maven => Enable Dependency Management**.

Cela a pour effet d'ajouter plusieurs containers, dont `Maven Dependencies`. En l'ouvrant, on observe les dépendances du projet cible (car Maven gère la transitivité) et celles que l'on a ajoutées : le projet cible lui-même, JUnit, Robotium...

De plus, le dossier `src` est devenu visible dans l'explorateur de projet Eclipse. Pour faire fonctionner les tests, j'ai réorganisé ce dossier en plaçant les dossiers du package (`com.adeo.android.app`) dans `src/main/java/` et non plus dans `src/`, ce qui donne : `src/main/java/com/adeo/android/app/`. Le dossier `test/` peut être supprimé. Nous placerons nos classes de test dans le dossier `src/main/java/com/adeo/android/app/`.

4. Créer une classe de test

Nous allons créer une classe nous permettant de tester une Activity (placée dans le répertoire `src/main/java/com/adeo/android/app/`).

- Elle hérite de la classe `ActivityInstrumentationTestCase2`.

- Elle contient forcément une méthode `setUp()` décrite ci-dessous.
- Elle peut contenir une méthode `testPreconditions()` qui sera lancée une seule fois avant la série de tests.
- Elle contient les méthodes de test.

Voici son code :

AdeoDashboardActivityTest.java

```
package com.adeo.android.app;

import
com.adeo.android.app.activity.AdeoDashboardActivi
ty;

import android.app.Instrumentation;
import
android.test.ActivityInstrumentationTestCase2;
import android.widget.TextView;

public class AdeoDashboardActivityTest extends
ActivityInstrumentationTestCase2 {

    private AdeoDashboardActivity
mAdeoDashboardActivity;
    private Instrumentation mInstrumentation;
    private TextView mTextView;

    private static final String
TARGET_PACKAGE = "com.adeo.android.app";

    @SuppressWarnings("unchecked")
    public AdeoDashboardActivityTest () {
        super(TARGET_PACKAGE,
AdeoDashboardActivity.class);
    }

    @Override
    protected void setUp() throws Exception {
        super.setUp();

        setActivityInitialTouchMode(false
);

        mAdeoDashboardActivity =
(AdeoDashboardActivity) getActivity();
        mInstrumentation =
getInstrumentation();

        mTextView = (TextView)
mAdeoDashboardActivity.findViewById(com.adeo.andr
oid.app.R.id.my_textview);
    }

    public void testPreconditions() {
        //Indiquez ici vos préconditions.
        Cette méthode est exécutée une seule fois, avant
l'exécution du premier test.
        //Par exemple :
        assertNotNull(mTextView);
    }

    public void testText() {
        String resourceString = "Ok";
        assertEquals(resourceString,
(String)mTextView.getText());
    }
}
```

Jusqu'ici, notre classe de test n'effectue que des tests basiques sur les états des composants de l'Activity et non des tests d'IHM. La dépendance *robotium-solo* que nous avons ajoutée dans le *pom.xml* va nous servir maintenant. Ainsi, on instancie un objet *Solo* dans notre méthode `setUp()`. Cet objet est un attribut de la classe de test, afin de pouvoir l'utiliser dans toutes les méthodes. Il suffit ensuite d'utiliser les méthodes de l'objet *Solo* pour effectuer les tests.

Le code de la classe utilisant Robotium serait alors :

AdeoDashboardActivityTest.java

```
package com.adeo.android.app;

import
com.adeo.android.app.activity.AdeoDashboardActivi
ty;
import com.jayway.android.robotium.solo.Solo;

import android.app.Instrumentation;
import
android.test.ActivityInstrumentationTestCase2;
import android.widget.TextView;

public class AdeoDashboardActivityTest extends
ActivityInstrumentationTestCase2 {

    private AdeoDashboardActivity
mAdeoDashboardActivity;
    private Instrumentation mInstrumentation;
    private Solo solo;
    private TextView mTextView;

    private static final String
TARGET_PACKAGE = "com.adeo.android.app";

    @SuppressWarnings("unchecked")
    public AdeoDashboardActivityTest () {
        super(TARGET_PACKAGE,
AdeoDashboardActivity.class);
    }

    @Override
    protected void setUp() throws Exception {
        super.setUp();

        setActivityInitialTouchMode(false
);

        mAdeoDashboardActivity =
(AdeoDashboardActivity) getActivity();
        mInstrumentation =
getInstrumentation();

        mTextView = (TextView)
mAdeoDashboardActivity.findViewById(com.adeo.andr
oid.app.R.id.my_textview);

        solo = new Solo(mInstrumentation,
mAdeoDashboardActivity);
    }

    public void testPreconditions() {
        //Indiquez ici vos préconditions.
        Cette méthode est exécutée une seule fois, avant
l'exécution du premier test.
        //Par exemple :
        assertNotNull(mTextView);
    }
}
```



```
public void testText() {
    String resourceString = "Ok";
    assertEquals(resourceString,
        (String)mTextView.getText());
}

public void testButtonClick() {
    //On simule un clic sur le bouton
    //0, par exemple
    solo.clickOnButton(0);
}

@Override
public void tearDown() throws Exception {
    solo.finishOpenedActivities();
}
}
```

En lançant le build, vous pourrez regarder votre application Android s'animer et faire ce que vous lui avez demandé.

Voilà pour les tests unitaires et d'IHM sur un projet Android "mavenisé" !

5. Références

- Introduction au SDK Android : [Lien 59](#)
- Importer un projet Maven dans Eclipse en 5 minutes : [Lien 60](#)
- 3T : les Tests en Trois Temps : [Lien 61](#)
- Les Tests en Trois Temps (3T) en pratique : [Lien 62](#)
- À la découverte de JUnit : [Lien 63](#)

Retrouvez l'article de Maxime Ghignet en ligne : [Lien 64](#)

Introduction à Zest

Ce tutoriel est une introduction à la boîte à outils de visualisation de graphes sous Eclipse Zest. Au travers d'un exemple simple, nous montrerons comment construire un graphe en utilisant cette boîte à outils.

1. Introduction

1.1. Qu'est-ce que Zest ?

Eclipse Zest est une boîte à outils de visualisation de graphes, basée sur SWT et Draw2D. Elle permet de construire et visualiser des graphes, soit directement par création des branches et des nœuds, soit par l'utilisation de l'abstraction JFace. Cette abstraction permet une approche par modèle, en s'affranchissant de la construction du graphe proprement dite.

Dans ce tutoriel, je vous propose d'utiliser Zest afin de représenter un graphe simple, soit directement par la construction des nœuds et branches du graphe, soit par l'utilisation de l'abstraction JFace à partir d'un modèle de données.

Ce tutoriel suppose que vous disposiez des connaissances de base sur les technologies suivantes :

- développement de plugins avec Eclipse : [Lien 65](#) ;
- utilisation de Jface : [Lien 66](#).

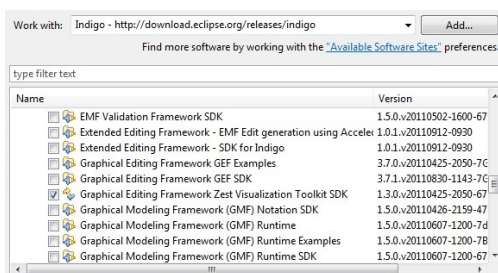
1.2. Présentation de notre fil conducteur

Notre fil conducteur dans cet article sera la représentation d'un réseau routier entre villes. Chaque ville est un nœud du graphe, et chaque route est une branche du graphe. Vous avez peut-être entendu parler de ces graphes dans le problème très connu du voyageur de commerce : [Lien 67](#). Nous n'entrerons pas dans ce tutoriel dans des considérations sur la théorie des graphes.

2. Notre premier graphe

2.1. Installation des outils

Pour installer Zest, ouvrez l'assistant d'installation de nouveaux plugins d'Eclipse et dans la liste sélectionnez « Graphical Editing Framework Zest Visualization Toolkit SDK », dans la section « Modeling ».



2.2. Composants de Zest

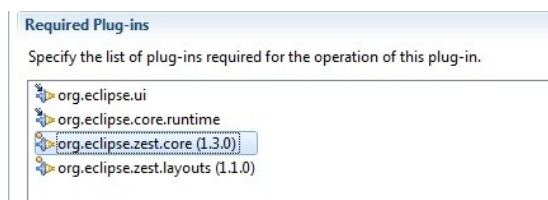
Zest est basé sur les composants suivants :

- GraphNode : un nœud du graphe ;
- GraphConnection : une branche du graphe, qui peut être orientée ou non ;
- GraphContainer : utilisé pour un graphe au sein d'un autre graphe ;
- Graph : élément de base de la boîte à outils, il contient les éléments précités.

D'autre part, Zest agence les composants du graphe en utilisant différents types de rendu, appelés « layouts ». Ces layouts seront détaillés dans un prochain article. Zest peut aussi filtrer les éléments du graphe, comme peut le faire un arbre ou une table JFace.

2.3. Création de notre premier graphe

Nous allons construire notre premier graphe. Pour cela, créez une nouvelle application Eclipse RCP, en utilisant le template « RCP application with a view ». Ajoutez dans les dépendances de l'application les plugins « org.eclipse.zest.core » et « org.eclipse.zest.layouts ».



Remplacez le contenu de la vue générée par l'assistant par le code ci-dessous. Notez que nous renommons la vue en « BasicGraphView ».

BasicGraphView.java

```
package com.abernard.zest;

import org.eclipse.swt.SWT;
import org.eclipse.swt.widgets.Composite;
import org.eclipse.ui.part.ViewPart;
import org.eclipse.zest.core.widgets.Graph;
import org.eclipse.zest.core.widgets.GraphConnection;
import org.eclipse.zest.core.widgets.GraphNode;
import org.eclipse.zest.layouts.LayoutStyles;
import org.eclipse.zest.layouts.algorithms.SpringLayoutAlgorithm;

/**
 * Cette vue contient un graphe construit
 * programmiquement, en construisant
 * 'a la main' les noeuds et les branches.
 */
```

```

*
* @author A. Bernard
*
*/
public class BasicGraphView extends ViewPart {

    /**
     * le graphe
     */
    private Graph graph;

    @Override
    public void createPartControl(Composite
parent) {
        // Le Graph contient tous les autres objets
        graph = new Graph(parent, SWT.NONE);

        // 1 :Les villes sont les noeuds du graphe
        GraphNode paris = new GraphNode(graph,
SWT.NONE, "Paris");
        GraphNode bordeaux = new GraphNode(graph,
SWT.NONE, "Bordeaux");
        GraphNode poitiers = new GraphNode(graph,
SWT.NONE, "Poitiers");
        GraphNode toulouse = new GraphNode(graph,
SWT.NONE, "Toulouse");

        // 2 :Construction des branches du graphe.
        // Chaque branche affiche la distance
        // kilometrique entre deux villes
        GraphConnection parisToulouse = new
GraphConnection(graph, SWT.NONE,
toulouse, paris);
        parisToulouse.setText("678km");

        GraphConnection parisPoitiers = new
GraphConnection(graph, SWT.NONE,
paris, poitiers);
        parisPoitiers.setText("338km");

        GraphConnection poitiersBordeaux = new
GraphConnection(graph, SWT.NONE,
bordeaux, poitiers);
        poitiersBordeaux.setText("235km");

        GraphConnection bordeauxToulouse = new
GraphConnection(graph, SWT.NONE,
toulouse, bordeaux);
        bordeauxToulouse.setText("244km");

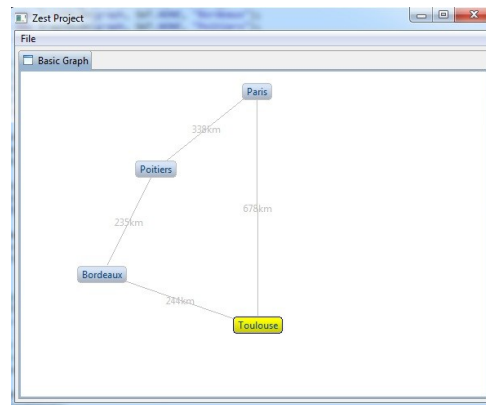
        // 3 :Definition de l'algorithm de rendu du
graphe
        // Ces layouts seront decrits dans un
prochain cours
        graph.setLayoutAlgorithm(new
SpringLayoutAlgorithm(
LayoutStyles.NO_LAYOUT_NODE_RESIZING),
true);
    }

    @Override
    public void setFocus() {
        //
    }
}

```

Les éléments de type `GraphNode` (bloc 1) sont les nœuds du graphe. Les éléments de type `GraphConnection` (bloc 2) servent à définir les branches du graphe. Enfin, on définit un layout pour le graphe (bloc 3).

Lancez l'application pour observer le résultat :



Vous pouvez cliquer sur les éléments du graphe pour les réorganiser. Voilà un graphe simple obtenu facilement et représentatif du problème. Néanmoins, la méthode de construction est lourde, et va vite devenir problématique au fur et à mesure que notre modèle va s'enrichir avec de nouveaux éléments.

Nous allons donc utiliser l'abstraction `JFace` pour représenter des modèles plus complexes dans notre graphe. C'est l'objet de la partie suivante.

3. Utilisation de l'abstraction JFace

3.1. Mise en place du modèle

Afin de construire notre graphe, nous avons besoin dans un premier temps de construire notre modèle. Celui-ci est basé sur les classes `Ville` et `Route` qui sont définies comme suit :

```

Classe Ville
package com.abernard.zest.model;

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

/**
 * Cette classe represente une ville, c'est-a-
 * dire une entite, ou un noeud, de
 * notre graphe.
 *
 * @author A. Bernard
 */
public class Ville {

    /**
     * nom de la ville
     */
    private String nom;

    /**
     * villes auxquelles est reliee cette ville
     */
    private List<Ville> connexions;

    /**
     * Constructeur
     * @param n nom de la ville

```

```

    */
    public Ville (String n) {
        this.nom = n;
        this.connexions = new ArrayList<Ville>();
    }

    /**
     * Ajouter une ville reliee a cette ville
     * @param v la ville a connecter
     */
    public void addConnexion(Ville v) {
        if (!connexions.contains(v)) {
            connexions.add(v);
        }
    }

    /**
     * Donne la liste des villes connectees a
    cette ville
     * @return les villes connectees
     */
    public List<Ville> getConnexions() {
        return this.connexions;
    }

    /**
     * Donne le nom de cette ville
     * @return le nom de cette ville
     */
    public String getNom() {
        return this.nom;
    }
}

```

Classe Route

```

package com.abernard.zest.model;

/**
 * Cette class represente une route, c'est-a-dire
    une branche du graphe.
 *
 * @author A. Bernard
 *
 */
public class Route {

    /**
     * la ville de depart de la route
     */
    private Ville source;

    /**
     * la ville d'arrivee de la route
     */
    private Ville destination;

    /**
     * la longueur de la route
     */
    private int longueur;

    /**
     * Constructeur
     * @param s la ville de depart
     * @param d la ville de destination
     * @param l la longueur de la route
     */
}

```

```

    public Route(Ville s, Ville d, int l) {
        this.source = s;
        this.destination = d;
        this.longueur = l;
    }

    /**
     * Donne la ville de depart
     * @return la ville de depart
     */
    public Ville getSource() {
        return this.source;
    }

    /**
     * Donne la ville d'arrivee
     * @return la ville d'arrivee
     */
    public Ville getDestination() {
        return this.destination;
    }

    /**
     * Donne la longueur de la route
     * @return la longueur de la route
     */
    public int getLongueur() {
        return this.longueur;
    }
}

```

La troisième classe du modèle est un singleton, et construit le modèle de deux manières différentes. La première approche permet d'accéder à la liste de toutes les villes, qui donnent à chaque fois à quelles autres villes elles sont reliées. La deuxième approche permet d'accéder à la liste des routes, qui donnent à chaque fois la ville de départ et celle de destination.

Dans la pratique, un modèle ne permet pas toujours cette double approche. Dans notre cas c'est uniquement pour explorer les possibilités de Zest.

Classe Model

```

package com.abernard.zest.model;

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

/**
 * Modele de notre graphe. Il contient toutes les
    villes et toutes les routes,
 * et permet d'accéder soit a la liste de toutes
    les villes, soit a la liste de
 * toutes les routes
 *
 * @author A. Bernard
 */
public enum Model {

    /**
     * l'instance unique de notre modele
     */
    INSTANCE;

    /**
     * toutes les villes du graphe
     */
}

```

```

*/
private List<Ville> villes;

/**
 * toutes les routes du graphe
 */
private List<Route> routes;

/**
 * Constructeur. Initialise le modele.
 */
private Model() {
villes = new ArrayList<Ville>();
routes = new ArrayList<Route>();

// Creation de toutes les villes du graphe
Ville paris = new Ville("Paris");
villes.add(paris);
Ville toulouse = new Ville("Toulouse");
villes.add(toulouse);
Ville bordeaux = new Ville("Bordeaux");
villes.add(bordeaux);
Ville poitiers = new Ville("Poitiers");
villes.add(poitiers);
Ville metz = new Ville("Metz");
villes.add(metz);

// Creation de toutes les routes
Route r1 = new Route(paris, toulouse, 678);
routes.add(r1);
paris.addConnexion(toulouse);

r1 = new Route(paris, poitiers, 338);
routes.add(r1);
poitiers.addConnexion(paris);

r1 = new Route(poitiers, bordeaux, 235);
routes.add(r1);
bordeaux.addConnexion(poitiers);

r1 = new Route(bordeaux, toulouse, 244);
routes.add(r1);
toulouse.addConnexion(bordeaux);

r1 = new Route(paris, metz, 333);
routes.add(r1);
paris.addConnexion(metz);
}

/**
 * Donne la liste de toutes les villes
 * @return la liste des villes
 */
public List<Ville> getVilles() {
return villes;
}

/**
 * Donne la liste de toutes les routes
 * @return la liste des routes
 */
public List<Route> getRoutes() {
return routes;
}
}

```

Notre modèle étant mis en place, nous pouvons maintenant créer nos graphes.

3.2. Approche orientée « nœuds »

Pour construire un graphe à partir des nœuds, Zest fournit l'interface « IGraphEntityContentProvider ». Créez la classe « EntityContentProvider » définie comme suit :

```

EntityContentProvider.java
package
    com.abernard.zest.viewer;

import java.util.List;

import org.eclipse.jface.viewers.Viewer;
import
org.eclipse.zest.core.viewers.IGraphEntityContent
Provider;

import com.abernard.zest.model.Ville;

/**
 * ContentProvider de notre graphe. Cette
implementation prend en donnees
 * d'entree la liste des villes.
 *
 * @author A. Bernard
 */
public class EntityContentProvider implements
IGraphEntityContentProvider {

    @SuppressWarnings("unchecked")
    @Override
    public Object[] getElements(Object input) {
// Cette methode definit les elements
d'entree du graphe.
// Dans notre cas ce doit etre une liste de
Villes.
return ((List<Ville>)input).toArray();
}

    @Override
    public void dispose() {
//
}

    @Override
    public void inputChanged(Viewer viewer,
Object oldInput, Object newInput) {
//
}

    @Override
    public Object[] getConnectedTo(Object entity)
{
// Pour chaque ville, on retourne la liste
des villes connectees a la
// ville courante
if (entity instanceof Ville) {
return
((Ville)entity).getConnexions().toArray();
} else {
return null;
}
}
}

```

Comme tout viewer JFace, il nous faut aussi définir un « LabelProvider ». Dans un premier temps, nos LabelProvider se contenteront d'étendre la classe « LabelProvider » de JFace. Nous verrons dans le cours

« Compléments sur Zest » que l'on peut définir bien d'autres choses grâce à des interfaces particulières. Créez donc la classe « EntityLabelProvider » :

EntityLabelProvider.java

```
package
    com.abernard.zest.viewer;

import org.eclipse.jface.viewers.LabelProvider;

import com.abernard.zest.model.Route;
import com.abernard.zest.model.Ville;

/**
 * LabelProvider du graphe. Determine le texte a
 * afficher selon le type
 * d'element.
 *
 * @author A. Bernard
 */
public class EntityLabelProvider extends
LabelProvider {

    @Override
    public String getText(Object element) {
        // On retourne la longueur de la route, ou le
        nom de la ville
        // On remarquera que le texte sur la route
        n'est jamais affiche :
        // aucun element de type Route n'est traite.
        if (element instanceof Route) {
            return ((Route)element).getLongueur() +
"km";
        } else if (element instanceof Ville) {
            return ((Ville)element).getNom();
        } else {
            return null;
        }
    }
}
```

Créez maintenant la vue « LinkGraphView », qui affichera un graphe « orienté nœuds ». Les données d'entrée du graphe sont données via la méthode « setInput », et sont donc les villes de notre modèle.

EntityGraphView.java

```
package com.abernard.zest;

import org.eclipse.swt.SWT;
import org.eclipse.swt.widgets.Composite;
import org.eclipse.ui.part.ViewPart;
import org.eclipse.zest.core.viewers.GraphViewer;
import org.eclipse.zest.layouts.LayoutStyles;
import org.eclipse.zest.layouts.algorithms.SpringLayoutAlgorithm;

import com.abernard.zest.model.Model;
import com.abernard.zest.viewer.EntityContentProvider;
import com.abernard.zest.viewer.EntityLabelProvider;

/**
 * Cette vue affiche un graphe construit a partir
 * d'un modele base sur les
 * noeuds du graphe (dans notre cas les villes).
```

```
*
 * @author A. Bernard
 */
public class EntityGraphView extends ViewPart {

    /**
     * le graphe
     */
    private GraphViewer viewer;

    @Override
    public void createPartControl(Composite
parent) {
        viewer = new GraphViewer(parent, SWT.NONE);
        // Traitement du contenu
        viewer.setContentProvider(new
EntityContentProvider());
        // Traitement de l'affichage
        viewer.setLabelProvider(new
EntityLabelProvider());

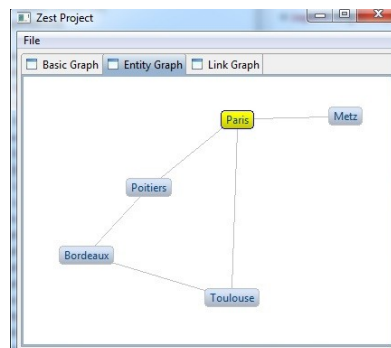
        // Donnees d'entree du graphe
        // Comme pour les autres viewers JFace, la
        methode setInput doit etre
        // appelee APRES la definition des
        ContentProvider et LabelProvider.
        viewer.setInput (Model.INSTANCE.getVilles());

        // Definition du layout
        viewer.setLayoutAlgorithm(new
SpringLayoutAlgorithm(LayoutStyles.NO_LAYOUT_NODE_RESIZING));
        viewer.applyLayout();

    }

    @Override
    public void setFocus() {
        //
    }
}
```

Le graphe est donc construit correctement, et ceci de manière très simple. On constate une chose : même si le LabelProvider fournit le texte à afficher pour des éléments de type « Route », aucun texte n'est affiché. On le verra par la suite, le fonctionnement est différent dans une approche orientée « branches ».



3.3. Approche orientée « branches »

Construisons maintenant notre graphe à partir des liens entre nœuds. Pour cela, Zest met à notre disposition l'interface « IGraphContentProvider ». Créez la classe « LinkContentProvider » comme suit :

LinkContentProvider.java

```
package
    com.abernard.zest.viewer;

import java.util.ArrayList;

import org.eclipse.jface.viewers.Viewer;
import
org.eclipse.zest.core.viewers.IGraphContentProvid
er;

import com.abernard.zest.model.Route;

/**
 * ContentProvider de notre graphe. Cette
implementation prend en donnees
 * d'entree la liste des routes.
 *
 * @author A. Bernard
 */
public class LinkContentProvider implements
IGraphContentProvider {

    @Override
    public void dispose() {
        //
    }

    @Override
    public void inputChanged(Viewer viewer,
Object oldInput, Object newInput) {
        //
    }

    @Override
    public Object getSource(Object rel) {
        // Donne pour chaque branche du graphe le
noeud source de la branche
        if (rel instanceof Route) {
            return ((Route)rel).getSource();
        } else {
            return null;
        }
    }

    @Override
    public Object getDestination(Object rel) {
        // Donne pour chaque branche du graphe le
noeud destination de la
        branche
        if (rel instanceof Route) {
            return ((Route)rel).getDestination();
        } else {
            return null;
        }
    }

    @SuppressWarnings("unchecked")
    @Override
    public Object[] getElements(Object input) {
        // Ici, les donnees d'entree du graphe sont
les routes du modele, c'est-
        // a-dire les branches du graphe.
        return ((ArrayList<Route>)input).toArray();
    }
}
```

Là aussi, nous devons définir un LabelProvider. Notez que dans ce cas présent, nous pourrions réutiliser celui que

nous avons défini dans la partie précédente. Cependant, en prévision du cours suivant, je vous invite à créer la classe « LinkLabelProvider » :

LinkLabelProvider.java

```
package
    com.abernard.zest.viewer;

import org.eclipse.jface.viewers.LabelProvider;

import com.abernard.zest.model.Route;
import com.abernard.zest.model.Ville;

/**
 * LabelProvider du graphe. Determine le texte a
afficher selon le type
 * d'element.
 *
 * @author A. Bernard
 */
public class LinkLabelProvider extends
LabelProvider {

    @Override
    public String getText(Object element) {
        // On retourne la longueur de la route, ou le
nom de la ville
        // Dans ce cas, les textes sont affiches sur
tous les elements du graphe
        if (element instanceof Route) {
            return ((Route)element).getLongueur() +
"km";
        } else if (element instanceof Ville) {
            return ((Ville)element).getNom();
        } else {
            return null;
        }
    }
}
```

Enfin, nous pouvons créer la vue qui va afficher le graphe :

LinkGraphView.java

```
package com.abernard.zest;

import org.eclipse.swt.SWT;
import org.eclipse.swt.widgets.Composite;
import org.eclipse.ui.part.ViewPart;
import org.eclipse.zest.core.viewers.GraphViewer;
import org.eclipse.zest.layouts.LayoutStyles;
import
org.eclipse.zest.layouts.algorithms.SpringLayoutA
lgorithm;

import com.abernard.zest.model.Model;
import
com.abernard.zest.viewer.LinkContentProvider;
import
com.abernard.zest.viewer.LinkLabelProvider;

/**
 * Cette vue affiche un graphe construit a partir
d'un modele base sur les
 * branches du graphe (dans notre cas les
routes).
 *
 * @author A. Bernard
 */
```

```

public class LinkGraphView extends ViewPart {

    /**
     * le graphe
     */
    private GraphViewer viewer;

    @Override
    public void createPartControl(Composite
parent) {
        viewer = new GraphViewer(parent, SWT.NONE);
        // Traitement du contenu
        viewer.setContentProvider(new
LinkContentProvider());
        // Traitement de l'affichage
        viewer.setLabelProvider(new
LinkLabelProvider());

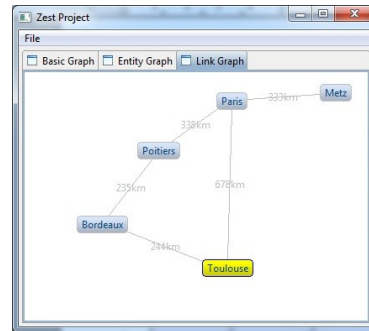
        // Donnees d'entree du graphe
        // Comme pour les autres viewers JFace, la
methode setInput doit etre
        // appelee APRES la definition des
ContentProvider et LabelProvider.
        viewer.setInput(Model.INSTANCE.getRoutes());

        // Definition du layout
        viewer.setLayoutAlgorithm(new
SpringLayoutAlgorithm(LayoutStyles.NO_LAYOU
T_NODE_RESIZING));
        viewer.applyLayout();
    }

    @Override
    public void setFocus() {
        //
    }
}

```

Dans le cas présent, le texte sur les routes est affiché, ainsi que celui sur les villes.



4. Conclusion

Dans ce tutoriel nous avons appris comment afficher un graphe, soit sans modèle de données associé, soit depuis un modèle en utilisant des concepts classiques de JFace et donc aisés à mettre en place. Notre graphe est pour l'instant basique, mais les possibilités d'amélioration sont nombreuses et ce sont ces possibilités que je vous montrerai dans la deuxième partie de ce cours sur Zest : « Compléments sur Zest ».

5. Liens utiles

Pour aller plus loin, voici quelques liens utiles :

- tutoriels sur le développement de plugins Eclipse et l'utilisation de SWT/JFace : [Lien 68](#) ;
- page officielle de Zest : [Lien 69](#).

Retrouvez l'article d'Alain Bernard en ligne : [Lien 70](#).

PGE (Plegat Graphic Engine) - Introduction

Article d'introduction sur le développement de PGE (Plegat Graphic Engine), moteur graphique basé sur JOGL (Java/OpenGL).

1. Introduction

1.1. Présentation

Cet article inaugure toute une série sur la création d'un moteur graphique pour mon application de manipulation de modèles éléments finis.

Le but n'est pas de créer un moteur de jeu multiplateforme, aux capacités graphiques avancées utilisant les dernières technologies, mais de mettre en place un moteur graphique simple.

Ceci permettra d'aborder la programmation OpenGL en Java au travers de l'utilisation de l'API JOGL. Je n'aborderai donc pas explicitement la théorie concernant OpenGL, étant donné que le net regorge d'informations et de tutoriels, sauf ponctuellement si nécessaire. Les articles seront donc essentiellement orientés programmation Java et JOGL.

1.2. But

Plegat (en plus d'être mon pseudo) est le nom du logiciel que je développe. Plegat est un pré/postprocesseur pour logiciels éléments finis. Il est codé en Java, pour des raisons de facilité de portabilité, et il utilise OpenGL pour la partie affichage graphique.

Cela fait maintenant quelques années que je travaille sur son développement, et le moteur graphique n'avait jamais été repensé depuis les premières versions. La version actuelle n'est pas assez évolutive pour correspondre à mes besoins, et devait donc être recodée. Cette réécriture du moteur servira donc de support à cette série de tutoriels.

Le but final est donc d'obtenir :

- un moteur graphique 3D évolutif ;
- permettant trois types d'affichage : fil de fer, solide et texturé ;
- contrôlable via le clavier et la souris ;
- avec une gestion précise des éléments graphiques à afficher ;
- possédant plusieurs modes de sélection à l'écran (picking, par zone...) ;
- éventuellement extensible par un système de plugins.

Afin d'avoir une idée du rendu graphique souhaité, on pourra jeter un œil aux copies d'écran des logiciels suivants :

- Gmsh : [Lien 71](#) ;
- Salome : [Lien 72](#).

ainsi qu'aux vidéos de démonstration suivantes :

- Démo Gmsh (vidéo) : [Lien 73](#) ;
- Démo Plegat (vidéo, v0.23, ancien moteur) : [Lien 74](#).

1.3. Technologies employées

Pour suivre ces articles, vous aurez besoin :

- d'un JDK Java SE. J'utilise personnellement le JDK Java 6, pour des soucis de portabilité personnels, mais toute version supérieure du JDK pourra être utilisée sans problème. Vous pouvez télécharger le JDK sur le site d'Oracle : [Lien 75](#) ;
- de l'API OpenGL JOGL. Celle-ci est disponible sur le site JogAmp ([Lien 76](#)), qui maintient et développe le binding OpenGL JOGL. Vous trouverez également sur ce site les API JOCL (binding OpenCL) et JOAL (binding OpenAL). La version actuelle de JOGL est la 2.0, mais encore en RC (Release Candidate) ;
- de votre EDI préféré. Pour ma part, ce sera NetBeans : [Lien 77](#).

1.4. Installation de JogAmp

Vous pouvez télécharger une archive incluant tous les fichiers composant l'API JogAmp sur cette page : [Lien 78](#). Choisissez l'archive correspondant à votre système d'exploitation (Win, Mac, Linux) et à votre architecture (32/64 bits). Les archives sont au format 7-zip.

Après avoir décompressé l'archive, les deux répertoires nécessaires pour la suite sont les répertoires "jar" et "lib". Le répertoire "jar" contient toutes les bibliothèques jar qui seront utilisées pour le développement de nos programmes. Le répertoire "lib" contient quant à lui les bibliothèques natives (*.dll sous Windows, *.so sous Linux) qui seront utilisées lors de l'exécution des programmes.

Il vous faut ensuite configurer votre EDI afin de pointer correctement sur ces deux répertoires. Pour cela, je vous renvoie vers la page du Wiki JogAmp ([Lien 79](#)) qui présente le processus de configuration pour les EDI suivants : Eclipse, IntelliJ IDEA, et NetBeans.

2. Mon premier cube

2.1. Tout est relatif...

... il faut donc bien définir l'espace dans lequel nous allons travailler.

Il y a quatre choses très importantes à déterminer pour voir

quelque chose à l'écran :

- l'endroit où sont nos éléments à dessiner ;
- l'endroit où l'on est ;
- l'endroit où l'on regarde ;
- où se trouve le haut.

Les éléments à dessiner seront pour le moment très simples. Nous commencerons par un cube unitaire (c'est-à-dire dont les côtés mesurent une unité), je vous fais grâce du triangle, un peu trop... basique ! Nous le placerons à l'origine de notre "univers" pour simplifier.

L'endroit où l'on est, celui où l'on regarde et le haut de la scène seront fixés dans ce premier article (ultérieurement, ils seront gérés par une caméra).

On définit tout d'abord une vision en perspective de notre scène grâce à l'instruction **gluPerspective** :

```
public void gluPerspective(double fovy,
                          double aspect,
                          double zNear,
                          double zFar)
```

fovy sera pris égal à 45°. aspect sera défini par le ratio entre la largeur de la fenêtre JOGL et sa hauteur. zNear sera égal à 0.1, et zFar à 100 (ces valeurs seront optimisées pour les prochains articles).

L'instruction JOGL permettant de définir la caméra (où l'on est, où l'on regarde, où est le haut) est **gluLookAt** :

```
public void gluLookAt(double eyeX,
                    double eyeY,
                    double eyeZ,
                    double centerX,
                    double centerY,
                    double centerZ,
                    double upX,
                    double upY,
                    double upZ)
```

Dans ce premier article, la caméra sera fixe. Les points de définition sont donnés par les paramètres suivants :

- où l'on est : (eyeX,eyeY,eyeZ)=(3,5,2) ;
- où l'on regarde : (centerX,centerY,centerZ)=(0,0,0) ;
- où est le haut : (upX,upY,upZ)=(0,0,1).

Voilà. Notre environnement est défini, il ne reste plus qu'à y dessiner...

2.2. Le repère principal

Une des choses que j'apprécie, c'est de savoir où je suis... nous allons donc dessiner un repère à l'origine de notre univers. Ce repère sera composé de trois lignes de couleurs différentes, afin de repérer quels sont les différents axes :

- l'axe X sera en rouge ;
- l'axe Y sera en vert ;
- l'axe Z sera en bleu.

Rien de compliqué pour le dessin, nous utiliserons les instructions **glColor3d** pour définir la couleur du trait, les traits étant tracés en mode **GL_LINES**.

Le dessin du repère principal sera affecté à une classe dédiée :

Classe PGEGlobalCoord

```
package pge_article_1;

import javax.media.opengl.GL2;

public class PGEGlobalCoord {

    public void draw(GL2 gl) {

        gl.glBegin(GL2.GL_LINES);

        // rouge : axe X
        glColor3d(1, 0, 0);
        glVertex3d(0, 0, 0);
        glVertex3d(1, 0, 0);

        // vert : axe Y
        glColor3d(0, 1, 0);
        glVertex3d(0, 0, 0);
        glVertex3d(0, 1, 0);

        // bleu : axe Z
        glColor3d(0, 0, 255);
        glVertex3d(0, 0, 0);
        glVertex3d(0, 0, 1);

        glEnd();
    }
}
```

2.3. Un cube à six faces

Pour le dessin du cube unitaire, une classe dédiée est également écrite. Le constructeur prend en paramètre la position dans l'espace du centre du cube, et une méthode **draw** permet de gérer l'affichage du cube. On y dessine les faces inférieures et supérieures du cube en mode **GL_LINE_LOOP**, puis les quatre arêtes joignant les deux faces en **GL_LINES**.

Pour la couleur, vous aurez droit à un joli cyan...

Classe PGEUnitCube

```
package pge_article_1;

import javax.media.opengl.GL2;

public class PGEUnitCube {

    public PGEUnitCube(double x,double y, double z) {

        this.x=x;
        this.y=y;
        this.z=z;
    }

    public void draw(GL2 gl) {

        // dessin face inférieure
        glBegin(GL2.GL_LINE_LOOP);
        glColor3d(0, 1, 1);
        glVertex3d(this.x - 0.5, this.y - 0.5,
this.z - 0.5);
        glVertex3d(this.x + 0.5, this.y - 0.5,
this.z - 0.5);
        glVertex3d(this.x + 0.5, this.y + 0.5,
```

```

this.z - 0.5);
    gl.glVertex3d(this.x - 0.5, this.y + 0.5,
this.z - 0.5);
    gl.glEnd();

    // dessin face supérieure
    gl.glBegin(GL2.GL_LINE_LOOP);
    gl.glVertex3d(this.x - 0.5, this.y - 0.5,
this.z + 0.5);
    gl.glVertex3d(this.x + 0.5, this.y - 0.5,
this.z + 0.5);
    gl.glVertex3d(this.x + 0.5, this.y + 0.5,
this.z + 0.5);
    gl.glVertex3d(this.x - 0.5, this.y + 0.5,
this.z + 0.5);
    gl.glEnd();

    // dessin des quatre arêtes manquantes
    gl.glBegin(GL2.GL_LINES);
    gl.glVertex3d(this.x - 0.5, this.y - 0.5,
this.z - 0.5);
    gl.glVertex3d(this.x - 0.5, this.y - 0.5,
this.z + 0.5);
    gl.glEnd();

    gl.glBegin(GL2.GL_LINES);
    gl.glVertex3d(this.x + 0.5, this.y - 0.5,
this.z - 0.5);
    gl.glVertex3d(this.x + 0.5, this.y - 0.5,
this.z + 0.5);
    gl.glEnd();

    gl.glBegin(GL2.GL_LINES);
    gl.glVertex3d(this.x - 0.5, this.y + 0.5,
this.z - 0.5);
    gl.glVertex3d(this.x - 0.5, this.y + 0.5,
this.z + 0.5);
    gl.glEnd();

    gl.glBegin(GL2.GL_LINES);
    gl.glVertex3d(this.x + 0.5, this.y + 0.5,
this.z - 0.5);
    gl.glVertex3d(this.x + 0.5, this.y + 0.5,
this.z + 0.5);
    gl.glEnd();
}

private double x,y,z; // position du centre
du cube
}

```

2.4. La fenêtre JOGL

Maintenant, nous avons tout pour dessiner notre cube... sauf une fenêtre JOGL. Nous allons créer une classe, que je nommerai **PgePanel**, dérivant de **GLJPanel**. Cette classe mère est choisie plutôt que **GLCanvas** car elle est plus adaptée à une utilisation en environnement Swing (GLCanvas étant plus adaptée pour AWT).

La trame de la classe **PgePanel** est la suivante :

Trame de la classe PgePanel

```

package pge_article_1;

import javax.media.opengl.GL;
import javax.media.opengl.GLAutoDrawable;
import javax.media.opengl.GLEventListener;
import javax.media.opengl.GLJPanel;
import javax.media.opengl.glu.GLU;

```

```

public class PGEPanel_article_1 extends GLJPanel
implements GLEventListener {

    public PGEPanel_article_1() {
        super();
        this.initComponents();
    }

    private void initComponents() {
        // initialisation des composants de
PgePanel
    }

    public void display(GLAutoDrawable arg0) {
        // affichage de la scène
    }

    public void reshape(GLAutoDrawable arg0, int
arg1, int arg2, int arg3, int arg4) {
        // mise à jour suite à redimensionnement
de la fenêtre
    }

    public void dispose(GLAutoDrawable glad) {
        // libération des ressources OpenGL
    }

    public void init(GLAutoDrawable glad) {
        // initialisation OpenGL
    }

    private int winAW; // largeur de la zone
d'affichage
    private int winAH; // hauteur de la zone
d'affichage
}

```

On se contentera d'ajouter un écouteur JOGL dans la méthode **initComponents()**, ainsi que la création des deux objets **PGEGlobalCoord** et **PGEUnitCube** :

```

private void initComponents() {
    this.addGLEventListener(this);

    this.cube=new PGEUnitCube(0, 0, 0);
    this.coord=new PGEGlobalCoord();
}

```

La méthode **reshape()** redéfinit les valeurs de la largeur et de la hauteur de la fenêtre JOGL :

```

public void reshape(GLAutoDrawable arg0, int
arg1, int arg2, int arg3, int arg4) {
    System.out.println("panel reshape");

    // réinitialisation des dimensions de la zone
d'affichage
    this.winAW = this.getWidth();
    this.winAH = this.getHeight();
}

```

Les méthodes **dispose()** et **init()** ne seront pas implémentées ici :

```

public void dispose(GLAutoDrawable glad) {
    System.out.println("running
"+this.getClass().getName()+" dispose method");
}

```

```
public void init(GLAutoDrawable glad) {
    System.out.println("running
    "+this.getClass().getName()+" init method");
}
```

La méthode **display()** se chargera de l'affichage de la scène JOGL :

```
public void display(GLAutoDrawable arg0) {

    GL2 gl = arg0.getGL().getGL2();
    GLU glu = new GLU();

    gl.glViewport(0, 0, this.winAW, this.winAH);

    // initialisation de la matrice de projection
    gl.glMatrixMode(GL2.GL_PROJECTION);
    gl.glLoadIdentity();

    // définition de la matrice de projection
    // vue en perspective, angle 45°, ratio
    // largeur/hauteur, plan de clipping "near" et "far"
    glu.gluPerspective(45, (double) this.winAW /
    (double) this.winAH, 0.1, 100.0);

    // position de la caméra : point (3,5,2)
    // point de visée: point (0,0,0)
    // vecteur "up": point (0,0,1)
    glu.gluLookAt(3, 5, 2, 0, 0, 0, 0, 0, 1);

    // initialisation de la matrice modèle
    // couleur de fond: noir
    gl.glMatrixMode(GL2.GL_MODELVIEW);
    gl.glLoadIdentity();
    gl.glClearColor(0.0f, 0.0f, 0.0f, 0.0f);
    gl.glClear(GL2.GL_COLOR_BUFFER_BIT);

    // dessin des axes du repère global
    this.cube.draw(gl);

    // dessine un cube unitaire à l'origine du
    // repère global
    this.coord.draw(gl);
}
```

2.5. Premier affichage de PGE

Il ne nous manque que la "main class" pour pouvoir tester notre fenêtre JOGL :

```
package pge_article_1;

import java.awt.event.WindowAdapter;
import java.awt.event.WindowEvent;
import java.util.EventListener;
import javax.media.opengl.GLCapabilities;
import javax.media.opengl.GLProfile;
import javax.swing.JFrame;

/**
 * Article1.java
 *
 * Code source du premier article.
 */
public class Main_article_1 implements
EventListener {
```

```
public static void main(String[] args) {

    // Initialisation JOGL
    GLProfile
    GLProfile
    glp=GLProfile.get(GLProfile.GL2); // profil
    Opengl Desktop 1.x à 3.0
    GLProfile.initSingleton(true);
    GLCapabilities caps=new
    GLCapabilities(glp);

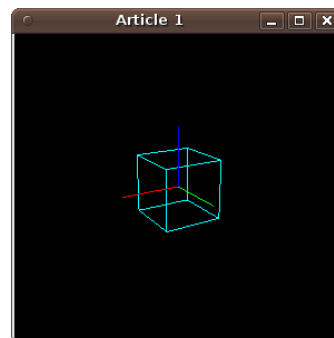
    final JFrame frame = new JFrame("Article
    1");
    PGEPanel_article_1 pane = new
    PGEPanel_article_1(caps);
    frame.getContentPane().add(pane);

    frame.addWindowListener(new
    WindowAdapter() {
        public void windowClosing(WindowEvent
    e) {
            System.out.println("closing main
    window...");
            System.exit(0);
        }
    });

    frame.setSize(300, 300);

    java.awt.EventQueue.invokeLater(new
    Runnable() {
        public void run() {
            frame.setVisible(true);
        }
    });
}
```

Il ne reste plus qu'à lancer l'exécution du programme pour obtenir notre premier cube sous PGE !



Notre premier cube !!!

3. Conclusion

Ce premier article a permis de présenter PGE, et de commencer l'implémentation du moteur graphique par l'affichage d'une scène simple.

Dans le prochain article, nous mettrons en place un système de caméra mobile, ainsi qu'une gestion du clavier et de la souris.

Les sources de cet article sont disponibles en suivant ce lien : src_PGE_article_1.zip ([Lien 80](#)).

Retrouvez l'article de Jean-Michel Borlot en ligne : [Lien 81](#).

Developpement Web

Les derniers tutoriels et articles

Usage avancé des fonctions JavaScript

Cet article est un complément à l'article sur les trois fondamentaux de JavaScript ([Lien 82](#)), il vaut mieux être déjà à l'aise avec JavaScript avant de crier au scandale en voyant ce qu'on peut en faire. Pour reprendre un bon mot de quelqu'un qui avait assisté à ma conférence sur JavaScript ([Lien 83](#)).

1. Introduction



JavaScript == la pornstar des langages de développement : souple, puissant, tu lui fais faire ce que tu veux et ça peut finir bien crade.

Admettons donc que vous ayez digéré sans problème les portées et les fonctions, passons à deux choses vraiment particulières à JavaScript :

1. Le renvoi de fonction qui permet de belles optimisations et qui ouvre la voie à des patterns que les amoureux de la théorie du langage apprécieront ;
2. Une implémentation de **classe statique**, pour reprendre le terme utilisé en PHP ou en Java.

Et enfin nous verrons une proposition d'implémentation de deux *design pattern* célèbres et particulièrement utiles en JavaScript : *Singleton* et *Factory*.

2. Classe statique

Pour rappel, en PHP et dans d'autres langages, une propriété ou une méthode statique peut être appelée sans que la classe ait été instanciée pour créer un objet. C'est généralement là que l'on range les constantes ou les fonctions utilitaires par exemple. En JavaScript, tout étant objet y compris les fonctions, cela se fait assez naturellement :

```
// constructeur
var myClass = function () {
};
myClass.staticMethod = function() {
    console.log('OK');
};
// que voit-on au niveau global ?
myClass.staticMethod(); // OK
```

Regardez la manière dont est définie `staticMethod` : on la range directement dans la fonction `myClass` ! Elle est donc directement disponible sans passer par la création d'un objet. Comparons d'ailleurs avec une définition de méthode d'objet comme on l'a vu dans les paragraphes précédents pour bien comprendre où sont disponibles ces nouvelles méthodes de classe.

```
// constructeur
var myClass = function () {
    return {
        publicMethod:function() {
            console.log('OK');
        }
    };
};
myClass.staticMethod = function() {
    console.log('OK');
};

// que voit-on au niveau global ?
myClass.publicMethod(); // Error
myClass.staticMethod(); // OK

// que voit l'instance ?
myObject = myClass();
myObject.publicMethod(); // OK
myObject.staticMethod(); // Error
```

Si vous exécutez ce code dans votre console, vous allez voir où se produisent les erreurs :

- vous ne pouvez pas accéder à `publicMethod` sans avoir d'abord instancié `myClass` ;
- l'instance de `myClass` ne contient pas `staticMethod` car celle-ci est directement disponible à partir du nom de la classe.

3. Renvoi de fonction

Une fonction peut se redéfinir elle-même quand elle s'exécute. Ça a l'air sale dit comme ça, mais nous allons voir un cas concret où cela est bien utile. Imaginez que vous construisez une minibibliothèque dont une des fonctions permet de se rattacher à un événement du DOM. Pour supporter tous les navigateurs, il y a deux méthodes aux noms distincts et aux arguments légèrement différents que vous pouvez encapsuler dans une fonction en faisant un simple `if`.

```
var onDOMEvent =
function( el, ev, callback) {
    // le monde de Microsoft
    if(document.body.attachEvent){
        el.attachEvent('on'+ev, callback);
    // le monde du W3C
    } else {
        el.addEventListener( ev, callback,
false);
    }
};
```

Cette fonction marche très bien, mais à chaque fois qu'elle

est exécutée, le test sur la condition aussi est exécuté. Si vous faites une bibliothèque, vous vous devez de ne pas ralentir les développeurs qui l'utiliseraient. Or cette fonction pourrait très bien être appelée des centaines de fois, exécutant ainsi inutilement du code. Pour optimiser cela, nous allons redéfinir la fonction à la volée lors de sa première exécution.

```
var onDOMEvent =
function ( ) {
    if(document.body.attachEvent) {
        return function(element, event, callback)
        {
            element.attachEvent('on'+ event,
callback);
        };
    } else {
        return function(element, event, callback)
        {
            element.addEventListener( event,
callback);
        };
    }
}();
```

Comme vous le voyez :

- cette fonction est auto-exécutée grâce aux deux parenthèses finales et n'a plus besoin des arguments puisqu'elle ne sera plus jamais exécutée après ;
- le if reste, mais il ne sera exécuté qu'une seule fois ;
- la fonction **renvoie des fonctions anonymes** contenant les codes spécifiques au navigateur, notez que ces fonctions attendent toujours les mêmes paramètres ;
- lorsque onDOMEvent() sera appelée, seul l'un ou l'autre corps de fonction sera exécuté, nous avons atteint notre objectif.

C'est une technique d'optimisation pas forcément évidente à intégrer mais qui donne de très bons résultats. Cela irait un peu trop loin pour cet article, mais si vous avez l'âme mathématique, cherchez donc sur le Web comment calculer la suite de Fibonacci en JavaScript, avec et sans « memoization » (Wikipedia : [Lien 84](#)). Vous pouvez également créer des fonctions spécialisées qui capturent certains paramètres pour vous éviter d'avoir à les préciser à chaque fois, technique connue sous le nom de *currying* (voir ce post de John Resig à ce sujet : [Lien 85](#)).

Autre cas concret d'école d'utilisation de cette technique. Partons du code suivant qui boucle sur un petit tableau d'objets et qui rattache l'événement onload à une fonction anonyme.

```
var queries = [ new XHR('url1'), new XHR('url2'),
new XHR('url3') ];
for(var i = 0; i < queries.length; i++) {
    queries[i].onload = function() {
        console.log( i ); // référence
    }
}
```

Observez bien la valeur de i : notre fonction anonyme crée une portée, le parseur JavaScript ne voit pas i dans cette

fonction, il remonte donc d'un niveau pour le trouver. Jusqu'ici tout va bien notre variable est bien référencée. Pourtant lorsque l'événement onload est appelé par le navigateur, nous avons une petite surprise :

```
queries[ 0 ].onload(); // 3!
queries[ 1 ].onload(); // 3!
queries[ 2 ].onload(); // 3!
```

L'interpréteur JavaScript a correctement fait son boulot : la fonction onload voit bien i (sinon nous aurions eu undefined), mais c'est une **référence** vers la variable, pas une copie de sa valeur ! Or onload n'est appelé qu'après que la boucle s'est terminée et i a été incrémentée entre-temps. Pour fixer cela, nous allons utiliser deux choses :

1. L'auto-exécution, qui va nous permettre de copier la valeur de i ;
2. Le renvoi de fonction pour que onload soit toujours une fonction.

Attention, ça peut piquer les yeux :

```
for(var i = 0; i < queries.length; i++) {
    queries[i].onload = function(i) {
        return function() {
            console.log( i ); // valeur
        };
    }(i); // exécution immédiate
}
// plus tard ...
queries[ 0 ].onload(); // 0
queries[ 1 ].onload(); // 1
queries[ 2 ].onload(); // 2
```

Essayez de suivre le chemin de l'interpréteur :

- i est donnée à la fonction anonyme auto-exécutante ;
- le paramètre de cette première fonction anonyme s'appelle aussi i : dans cette portée locale, i a pour valeur 0 (pour la première passe) ;
- la fonction renvoyée embarque toute la portée avec elle et n'a donc que la **valeur** de ce nouveau i, qui ne bougera plus.

Pour information, ce cas d'école est souvent posé lors des entretiens d'embauche si on veut vérifier que vous avez bien compris les portées.

4. Implémenter des design pattern

En combinant *namespace* (voir l'article JavaScript pour les développeurs PHP : [Lien 86](#)), portée maîtrisée, espace privé et émulation d'objets, nous allons implémenter la *design pattern Factory*. *Factory* ou *Singleton* sont très intéressants en JavaScript, notamment pour les *Widgets* (type jQuery UI) : vous pouvez vous retrouver sur des pages où vous ne savez pas si le JavaScript d'un *Widget* s'est déjà exécuté sur tel élément du DOM. Pour optimiser, vous ne voulez pas recréer systématiquement le *Widget*, mais plutôt **créer une nouvelle instance ou récupérer l'instance en cours**. Vous avez donc besoin :

1. D'interdire la création directe d'un objet ;
2. De passer par une fonction qui va faire la vérification pour vous et vous renvoyer une instance.

Commençons par créer notre espace de travail, ainsi que notre *namespace* :

```
(function(){
// création ou récupération du namespace et du
sous-namespace
MY = window.MY || {};
MY.utils = MY.utils || {};
// constructeur
MY.utils.XHR=function( url ){
    console.log( url );
};
})();
```

À ce stade, nous avons une classe accessible de l'extérieur avec `new MY.utils.XHR(url);`. C'est bien mais pas top, nous voudrions passer par une fonction qui va vérifier s'il n'existe pas déjà une instance avec ce même paramètre URL. Nous allons déclencher une erreur en cas d'appel direct (comme en PHP : [Lien 87](#)) et prévoir une fonction `getInstance(url)` qui va se rattacher au *namespace* de la fonction constructeur.

```
(function(){
// création ou récupération du namespace et du
sous-namespace
MY = window.MY || {};
MY.utils = MY.utils || {};
// constructeur
MY.utils.XHR=function( url ){
    throw new Error('please use
MY.utils.XHR.getInstance()');
};
// factory
MY.utils.XHR.getInstance = function( url ) {
};
})();
```

Enfin, nous allons introduire une variable privée qui va contenir la liste de nos instances (un objet `currentInstances` avec en index l'URL et en valeur l'instance). Nous allons également rendre privé notre vrai constructeur.

```
(function(){
// constructeur
MY.utils.XHR=function( url ){
    throw new Error('please use
MY.utils.XHR.getInstance()');
};

//constructeur privé
var XHR = function( url ){
```

```
    console.log( url );
};
// liste privée d'instances
var currentInstances = {};

// factory
MY.utils.XHR.getInstance = function( url ) {
    // déjà créé ? on renvoie l'instance
    if(currentInstances[url]) {
        return currentInstances[url];
    // on crée, on enregistre, on renvoie
    } else {
        return currentInstances[url] = new
XHR(url);
    }
};
})();
```

Telle quelle, cette implémentation permet déjà de créer une *Factory* pour des objets qui ont besoin d'être uniques mais qui n'ont qu'un seul paramètre (id d'un objet DOM, URL...). La vraie raison d'être d'une *Factory*, c'est de gérer des objets complexes à instancier, à vous donc d'étendre ses fonctionnalités. Les puristes auront remarqué que l'objet renvoyé n'était pas du type `MY.utils.XHR` et qu'on ne pouvait donc pas faire de vérification avec `instanceof` du type de l'objet. Honnêtement je ne connais pas de bon moyen de le faire, à vous de voir si c'est un manque dans votre code.

Vous voilà parés pour écrire du code un peu plus maintenable et mieux rangé et vous pourrez crâner dans les dîners en ville en racontant vos exploits de codeur JavaScript orienté objet.

5. Conclusion

- JavaScript a des concepts différents des langages majeurs et devient extrêmement important sur votre CV. Prenez le temps de l'apprendre.
- Les bibliothèques telles que jQuery ne sont pas faites pour couvrir les cas que nous venons de voir. jQuery permet d'utiliser le DOM sereinement, pas d'organiser votre code proprement.
- J'ai essayé d'être pratique, mais lire un article ne suffira jamais pour comprendre des concepts de programmation : codez !

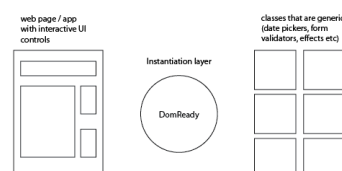
Retrouvez l'article de Jean-Pierre Vincent en ligne : [Lien 88](#).

MooTools Behavior / Comportement MooTools

Aaron Newton travaille depuis plusieurs années sur la conception d'interfaces utilisateur et apporte beaucoup de contenu et de soutien à l'équipe MooTools. Il propose ici une bibliothèque pour le framework MooTools dont le but est d'automatiser des comportements afin d'alléger le code. Cette bibliothèque s'appelle MooTools Behavior (comportement).

1. Objectif

Tous les sites Web et applications bien rédigés qui sont interactifs ont le même modèle de base :



Chaque page d'un site ou d'une application que vous construisez est ésothérique. Il peut y avoir n'importe quelle combinaison d'éléments interactifs, certains étant interactifs les uns avec les autres (par exemple, une validation de formulaire pourrait interagir avec un contrôleur Ajax pour empêcher d'envoyer un formulaire qui n'est pas valide).

Typiquement, ce code de cohésion existe dans un état de DOMReady. En gros, il faut sélectionner le formulaire et instancier telle classe avec telles options. Ce code est fragile ; si vous changez le DOM ou le code, l'état se casse facilement. Ce n'est pas réutilisable, cela fonctionne seulement pour un état spécifique de la page.

La bibliothèque proposée essaye d'extraire le code du DOMReady pour constituer quelque chose que vous écrivez seulement une fois et employez souvent. C'est rapide et facile à adapter et à étendre. Au lieu d'avoir un bloc DOMReady qui, par exemple, trouve toutes les images d'une page et les met dans une galerie, ou qui, dans un autre bloc, recherche la cible pour tous les liens sur la page et les transforme en info bulles, la bibliothèque Behavior fait une simple recherche de tous les éléments que vous avez marqués. Chaque élément est passé par un filtre. Un filtre est une fonction (avec peut-être une certaine configuration) que vous avez nommée. Chacune de ces fonctions prend l'élément, lit ses propriétés d'une façon prescrite et appelle le composant UI (interface utilisateur) approprié.

2. Pourquoi ?

En bref, au lieu d'avoir une fonction DOMReady qui trouve les éléments dans votre DOM et instancie des classes et autres, vous mettez la configuration dans le HTML lui-même et vous écrivez le code de la méthode `new Foo(...)` seulement une fois. Exemple :

JavaScript :

```
new Behavior().apply(document.body);
new Delegator().attach(document.body);
```

HTML :

```
<div data-behavior="Accordion" data-accordion-
options="
  'headers': '.toggle',
  'sections': '.target'
">
  <div class="toggle">Toggle 1</div>
  <div class="target">This area is controlled by
Toggle 1.</div>
  <div class="toggle">Toggle 2</div>
  <div class="target">This area is controlled by
Toggle 2.</div>
  <div class="toggle">Toggle 3</div>
  <div class="target">This area is controlled by
Toggle 3.</div>
</div>

<hr/>
<div id="box">
  <p>
    <a data-trigger="toggleClass" data-
toggleclass-options="
      'target': '!div#box',
```

```
      'class': 'red'
    ">Click to toggle this color to red and
back.</a>
  </p>
</div>
```

CSS :

```
.red {
  background: #DE3535;
}
a {
  display: block;
  cursor: pointer;
}
.toggle {
  cursor:pointer;
  background: #777;
  padding: 2px;
}
.target {
  padding: 4px;
}
[data-behavior=Accordion] {
  width: 300px;
  margin: 10px;
}
#box {
  border: 1px solid #000;
  margin: 10px;
  text-align: center;
  display:inline-block;
}
[data-trigger=toggleClass] {
  width: 100px;
  padding: 40px;
}
```

Donc, au lieu de ceci :

```
$$('form').each(function(form){
  new FormValidator(form, someOptions);
  new Form.Request(form, someOptions);
});
new Tips($$('.tip'));
$$('.accordion').each(function(container){
  new Accordion(container.getElements('.togglers'),
container.getElements('.section'), someOptions);
});
//etc.
```

Vous faites :

```
<form data-behavior="FormValidator FormRequest"
data-formvalidator-
options="{someOptions}">...</form>
<a data-behavior="Tip" title="I'm a
tip!">blah</a>
<div data-behavior="Accordion" data-accordion-
options="{someOptions}">...</div>
```

Pensez à cela comme à une délégation (comme dans la délégation d'événements : [Lien 89](#)) pour l'invocation de classe. Si vous employez le DOMReady pour faire votre initialisation et que vous voulez permuter de l'HTML via un appel Ajax, vous devez réappliquer cette initialisation de manière sélective pour les composants que vous mettez à jour uniquement, ce qui est souvent douloureux. Pas

avec Behavior. Vous appliquez simplement les filtres à la réponse.

Vous faites beaucoup moins de sélection dans le DOM ; vous ne lancez qu'une fois `$$('[data-behavior']')` (cependant, certains filtres peuvent lancer plusieurs sélecteurs sur eux-mêmes - comme l'accordéon trouvant ses toggles et sections).

L'initialisation du DOMReady est toujours étroitement liée aux DOM de toute façon, mais elle est également séparée de lui. Si vous changez le DOM, vous pourriez casser le JS et toujours devoir le resynchroniser. Vous ne devez presque pas faire cela ici parce que le DOM et sa configuration sont étroitement liés et sont au même endroit.

Les développeurs qui ne sont peut-être pas intéressés par l'écriture de composants n'ont pas besoin de patauger dans le JS pour l'utiliser. C'est une véritable affaire si vous travaillez avec une équipe que vous devez soutenir.

Behavior est conçu pour les applications qui mettent constamment à jour l'UI avec de nouvelles données du serveur. Ce n'est toutefois pas un remplaçant du MVC. La bibliothèque est conçue pour le développement Web qui emploie du code HTML et non des API JSON (cependant, ça peut être amusant de jouer avec). Si vous détruisez un nœud qui a un composant initialisé, il est facile de s'assurer que le composant se nettoie lui-même. La bibliothèque permet également d'empêcher une mauvaise configuration et propose une API qui facilite la lecture des valeurs de la configuration (on en reparle plus loin).

Il y a quelques autres astuces ; vous obtenez librement des spécifications de tests et des modèles parce que le code pour les créer tous les deux est dans le filtre du Behavior. Voici un exemple de ce qu'il faut pour écrire une spécification sur un composant et ÉGALEMENT sur la manière de l'instancier (ceci emploie Behavior.SpecsHelpers.js : [Lien 90](#)).

```
Behavior.addFilterTest({
  filterName: 'OverText',
  desc: 'Creates an instance of OverText',
  content: '<input data-behavior="OverText"
title="test"/>',
  returns: OverText
});
```

Ce code ci-dessus peut être employé pour valider que la partie de HTML impliquée crée, en fait, une instance d'OverText ; il peut également être employé avec Benchmark.js ([Lien 91](#)) pour voir lesquels de vos filtres sont les plus coûteux.

3. Delegator

La bibliothèque inclut également un fichier appelé Delegator qui est essentiellement la même chose que le Behavior mais pour les événements. Par exemple, disons que vous avez un modèle qui, une fois que l'on clique sur un lien, va cacher un élément parent. Plutôt que d'écrire ce code à chaque fois :

```
document.body.addEvent("click:a.hideParent",
function(e, link){
  e.preventDefault();
  link.getParent().hide();
});
```

Vous enregistrez ce modèle auprès du Delegator et vous n'avez plus qu'à faire :

```
<a data-trigger="hideParent" data-hideparent-
options ={"target": '.someSelector'}">Hide Me!
</a>
```

Il fournit essentiellement la même valeur que le Behavior, mais au moment d'un événement. L'exemple ci-dessus est assez basique, on est d'accord. Mais considérez combien de ces petites choses vous écrivez pour faire une fonctionnalité d'une application Web. Si vous pouvez les créer une fois et les configurer en ligne, vous vous épargnez beaucoup de lignes de code.

4. BehaviorAPI

Cette bibliothèque autonome facilite la lecture des propriétés des données et des éléments. Les exemples des expressions HTML évaluées sont comme suit (ils produisent tous le même résultat) :

```
<tag data-behavior="Filter1 Filter2" data-
filter1-options="{opt1: 'foo', 'opt2': 'bar',
'selector': '.selector'}"> //preferred
<tag data-behavior="Filter1 Filter2" data-
filter1-options="{opt1: 'foo', 'opt2': 'bar',
'selector': '.selector'}"> //no braces on JSON
<tag data-behavior="Filter1 Filter2" data-
filter1-options="{opt1: 'foo', 'opt2': 'bar'}"
data-filter1-selector=".selector">
<tag data-behavior="Filter1 Filter2" data-
filter1-opt1='foo' data-filter1-opt2='false'
data-filter1-selector=".selector">
```

La valeur de l'option est d'abord analysée comme un objet JSON (il est légèrement plus laxiste dans la mesure où vous ne devez pas l'envelopper dans des {} juste pour la convenance). Les valeurs définies ici sont lues afin de permettre d'exprimer des tableaux, des nombres, des booléens, etc. Les fonctions et les rappels (callbacks) ne sont généralement pas employés par Behavior.

Si vous essayez de lire une valeur qui n'est pas définie dans cet objet d'options, le code tente alors de lire le nom de la propriété directement (par exemple data-behaviorname-prop). Cette valeur est toujours une chaîne de caractères à moins que vous ayez spécifié un type. Si un type est spécifié, la valeur est lue par le parseur JSON et est validée en fonction de ce type.

Même si vous ne voulez pas utiliser toutes les fonctionnalités de la bibliothèque Behavior, cette bibliothèque peut être utile si vous aimez l'idée d'inclure de la configuration en ligne. Il y a beaucoup plus de choses dans l'API BehaviorAPI. Il vaut donc mieux parcourir attentivement la documentation : [Lien 92](#).

5. Documentation

- Behavior : [Lien 93](#)
- BehaviorAPI : [Lien 92](#)

- Element.Data : [Lien 94](#)

6. Comportements courants

Vérifiez les ressources sur les filtres fournis par l'auteur :

- [Lien 95](#)
- [Lien 96](#)
- [Lien 97](#)

7. Démonos

- Démonos MooTools Bootstrap et Documentation : [Lien 98](#)
- Clientcide : [Lien 99](#) (cliquez sur "demos" et cherchez les entrées "Delegators" et "Behaviors" à la fin du menu sur la gauche).

8. Remarques

Voici quelques remarques concernant l'implémentation. La documentation devrait être lue probablement en premier lieu parce qu'elle donne des exemples d'utilisation.

- Seulement un sélecteur à la fois ; ajouter 1000 filtres n'affecte pas les performances.
- Les nœuds peuvent avoir de nombreux filtres.
- Les nœuds peuvent avoir un nombre arbitraire d'options supportées pour chaque filtre (data-behaviorname-foo="bar").
- Les nœuds peuvent définir des options en JSON (c'est actuellement l'implémentation conseillée - data-behaviorname-options="your JSON").
- Les éléments peuvent être supprimés avec une destruction configurée ; nettoyer un élément nettoie également tous les enfants de cet élément qui appliquent le même comportement.
- Les comportements sont seulement appliqués une fois à un élément ; si vous appelez myBehavior.apply(document.body) une douzaine

de fois, les éléments avec des filtres auront ces filtres appliqués seulement une fois (on peut forcer à réappliquer le comportement).

- Les filtres sont des instances de classes qui sont appliquées à tout nombre d'éléments. Ils ont un nom unique.
- Les filtres peuvent avoir un espace de nom. Déclarez un filtre appelé Foo.Bar et référencez ses options comme suit : data-foo-bar-options="...".
- Il y a des filtres "globaux" qui sont enregistrés pour toutes les instances d'un comportement.
- Les instances de filtres obtiennent la priorité. Ceci tient compte du fait que les bibliothèques fournissent des filtres (comme ici : [Lien 95](#)). Ceci est nécessaire pour qu'une instance spécifique le réécrive sans affecter l'état global (ce modèle est dans le Form.Validator de **MooTools** et fonctionne assez bien).

9. Limitations

À cause de la recherche DOM pour la création et la destruction, vous ne pouvez pas avoir des exemples de comportements inclus l'un dans l'autre.

10. Téléchargement

Vous pouvez trouver la bibliothèque Behavior sur github ([Lien 100](#)) et aussi sur Clientcide ([Lien 101](#)) où vous trouverez également un constructeur. Celui-ci vous fournit un stock de comportements provenant du **Clientcide** et ceux dont j'ai donné l'autorisation pour le **MooTools More**. Si vous voulez prendre le constructeur par défaut, vérifiez bien de sélectionner "MooTools Bootstrap" dans le menu du haut (ou bien cliquez ici : [Lien 102](#)).

Retrouvez l'article d'Aaron Newton traduit par Vermine en ligne : [Lien 103](#)

La syntaxe d'une règle CSS

Ce second article a pour objectif de vous faire découvrir les bases du CSS : vous allez être initié à la syntaxe utilisée dans les feuilles de style.

Cela va consister à étudier en particulier ce qu'est une règle CSS. On va la décortiquer pour vous donner un aperçu de ses principales composantes : les sélecteurs, les blocs de déclarations, les déclarations, les propriétés et les valeurs.

Nous verrons ensuite quelle est l'influence de la présentation du code CSS : c'est-à-dire les effets de la casse du texte, des espaces, tabulations et autres retours à la ligne sur la bonne interprétation du code.

Puis nous aborderons les notions de regroupement des sélecteurs et de propriétés raccourcies. Je vous montrerai comment mettre des commentaires dans une feuille de style, et, enfin, j'aborderai des règles un peu particulières : les règles at-rules.

1. La syntaxe d'une règle CSS

Dans cet article nous allons étudier la structure d'une règle CSS et ses principales composantes que sont les sélecteurs et les déclarations. Nous verrons ensuite l'influence de la présentation du code CSS. Puis on abordera les notions de regroupement des sélecteurs et les propriétés raccourcies.

On verra comment mettre des commentaires dans le code CSS. Et, enfin, on étudiera succinctement les règles at-rules.

1.1. Structure d'une règle CSS

Une feuille de style est structurée en règles CSS. En voici un exemple :

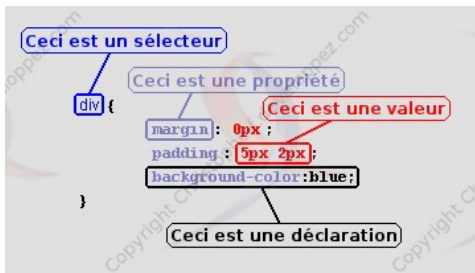
Une règle CSS :

```
div {
    margin :          0px;
    padding :         5px 2px;
    background-color : blue;
}
```

Une règle CSS se décompose en un sélecteur et une ou plusieurs déclarations appelées bloc de déclaration se trouvant entre les accolades.

Le sélecteur permet d'indiquer à quel élément la règle s'applique. Le bloc de déclaration englobe toutes les déclarations que l'on souhaite voir appliquer au sélecteur indiqué.

Toutes les règles CSS sont donc écrites sur le même modèle : un sélecteur suivi d'une accolade ouvrante ({), puis une ou plusieurs déclarations chacune subdivisée entre propriété et valeur, se terminant par un point virgule. Et enfin une accolade fermante (}).



Une règle CSS décomposée

1.1.1. Sélecteurs

Dans une règle CSS, le sélecteur correspond à la partie du code se trouvant en dehors des accolades. Il a pour rôle d'indiquer au navigateur ce qui doit être mis en forme. Vous allez indiquer le ou les éléments (X)HTML auxquels une règle CSS va s'appliquer.

Deux exemples de sélecteurs :

```
div {
    border :          1px solid blue;
    padding :         10px;
    color :           white;
    background-color : gray;
}
p.paragraphe {
    font-size :       1.6em;
    color :           red;
}
```

Dans cet exemple, la première règle va attribuer une bordure, des marges internes, la couleur blanche au texte et grise à l'arrière-plan de tous les éléments <div>.

La seconde règle déclare une taille de texte et la couleur rouge pour tous les paragraphes (balises <p>) possédant l'attribut **class** avec la valeur 'paragraphe'.

Consultez l'exemple 01 sur le principe des sélecteurs dans une règle CSS : [Lien 104](#).

Je détaillerai l'ensemble des sélecteurs disponibles dans le langage CSS dans un prochain article.

1.1.2. Déclarations et blocs de déclarations

Les déclarations sont placées à la suite du sélecteur, entre des accolades. L'ensemble des déclarations d'une règle est appelé "bloc de déclaration".

Un bloc de déclaration est composé d'une ou plusieurs déclarations. Chaque déclaration est constituée de deux éléments : une propriété et une valeur.

Exemple schématique d'une règle CSS :

```
sélecteur {
    propriété :      valeur;
    propriété :      valeur;
}
```

La propriété est un mot clé CSS permettant de modifier un point précis au niveau du rendu des éléments (X)HTML ciblés par le sélecteur. Par exemple les marges avec le mot clé **margin**, ou la taille du texte avec le mot clé **font-size**...

La valeur permet de spécifier une valeur différente que celle appliquée par défaut à la propriété en question.

Voici un exemple concret :

Un exemple de déclaration avec différentes propriétés et valeurs :

```
#contenu {
    position :        relative;
    top :             10px;
    left :            100px;
    width :           70%;
    border :          2px dashed #000000;
    padding :         12px;
    font-size :       1.2em;
    font-weight :     bold;
    background :      url('../images/logo.gif') no-repeat center;
    color :           rgb(125,200,100);
}
```

Consultez l'exemple 02 sur le principe des déclarations dans une règle CSS : [Lien 105](#).

Vous voyez là aussi que toutes les déclarations sont fondées sur le même modèle : une propriété suivie de deux-points, lui-même suivi d'une valeur (composée d'un ou plusieurs éléments) et se terminant par un point-virgule. Le point-virgule de fin est optionnel pour la dernière déclaration ; mais là encore il est conseillé de l'indiquer tout de même pour des questions de lisibilité et de maintenance du code.

1.2. Présentation du code

L'indentation d'une règle CSS, et plus généralement sa mise en forme, est soumise à des règles assez souples : comme dans le code (X)HTML il n'est pas tenu compte des caractères blancs (espaces, tabulations, retours à la ligne...) pour la bonne interprétation du code par l'agent utilisateur. La présentation du code CSS est donc à votre convenance.

Mise en forme d'une règle CSS, exemples valides mais à déconseiller :

```
#ss-err { color :
blue;
font-size : 1.4em;
background
: fuchsia
;
margin :
50px
50px
50px;
}
```

L'exemple ci-dessus est par exemple identique à celui-ci :

Mise en forme d'une règle CSS plus lisible :

```
#ss-err {
color : blue;
font-size : 1.4em;
background : fuchsia;
margin : 50px 50px 50px 50px;
}
```

Ce dernier code est bien mieux organisé que le précédent : cela lui permet d'être plus lisible et plus clair. N'hésitez donc pas à utiliser cette seconde méthode pour écrire votre code.

Ceci n'est pas la seule méthode valable pour présenter son code, il doit exister autant de manières de présenter son code que de développeurs ! Voyons-en quelques-unes des plus usitées :

Diverses manières de mettre en forme du code CSS :

```
#ss-err
{
color : blue;
font-size : 1.4em;
background:fuchsia;
margin:50px 50px 50px 50px;
}

#ss-err { color: blue;font-size:
1.4em;background: fuchsia;margin: 50px 50px 50px
50px; }
```

Dans cet exemple les accolades sont toutes deux en début de ligne. Certains vont préférer cette méthode car cela permet de mieux visualiser l'ouverture et la fermeture des accolades de chaque règle. Une autre différence de présentation consiste à ne pas aligner les valeurs entre elles, ou encore à ne pas laisser d'espace entre les deux-points.

La seconde règle de cet exemple met en œuvre une autre méthode qui fait perdre un peu de lisibilité, mais qui a l'avantage de condenser un peu le code. À vous de faire votre choix...

Cependant cette liberté n'est pas totale : vous ne pouvez, par exemple, pas insérer de retour à la ligne, de tabulation ou d'espace pour couper une propriété, ou pour séparer le nombre et l'unité de mesure utilisée...

Ce code est faux et ne fonctionnera pas :

```
#av-err {
col
or : blue;
font- size : 1.4em;
background : fuch sia;
margin : 50px 50px 50 px 50px;
}
```

Consultez l'exemple 03 sur l'indentation du code : [Lien 106](#).

Le CSS est également, sauf exception, insensible à la casse. Vous pouvez écrire vos règles en majuscules ou en minuscules, même si la convention est d'écrire en minuscules.

Voici un code en majuscules qui est totalement opérationnel :

Une règle CSS en majuscules valide :

```
DIV {
COLOR : BLUE;
FONT-SIZE : 1.3EM;
BACKGROUND : FUCHSIA;
MARGIN : 50PX;
}
```

Ce principe d'insensibilité à la casse s'applique également au sélecteur lorsque c'est le sélecteur de type qui est employé, c'est-à-dire lorsque nous ciblons une balise (X)HTML. Que vous utilisiez `div` ou `DIV` comme sélecteur ne posera aucun problème, les deux seront pris en compte de la même manière.

Par contre sont sensibles à la casse les éléments qui ne sont pas régis par le CSS : cela concerne les noms de classe et d'identifiant, les noms des polices et les adresses URI. Par exemple la classe `'header'` n'est pas la même que la classe `'HeaDer'` ou encore la classe `'HEADER'`.

Le code ci-dessous ne fonctionnera qu'avec un identifiant exactement identique dans le code (X)HTML.

Exemple de code CSS avec un nom d'identifiant en majuscules :

```
#EXCEPTION {
propriété : valeur;
}
```

Cette règle n'aura d'effet que sur le second `<div>` et non sur le premier :

Exemple de code (X)HTML démontrant la sensibilité à la casse des noms d'id :

```
<div id="exception">
La règle CSS ci-dessus ne s'appliquera pas à ce
paragraphe.
</div>

<div id="EXCEPTION">
La règle CSS ci-dessus s'appliquera à ce
paragraphe.
</div>
```

Visualisez ces trois morceaux de code dans l'exemple 04

relatif à la casse du code CSS : [Lien 107](#).

Je vous ai indiqué à l'instant que les noms de police sont sensibles à la casse comme cela ressort des spécifications CSS. Il faut cependant préciser que cette règle n'est pas appliquée de manière stricte par tous les navigateurs.

Exemple de déclarations de nom de police avec différentes casses :

```
#ex11 {
    font-family :    Arial, sans-serif;
}
#ex12 {
    font-family :    arial, sans-serif;
}
#ex13 {
    font-family :    ARiAl, sans-serif;
}
```

Ces trois règles vont toutes s'appliquer comme vous pouvez le constater dans l'exemple 05 sur l'influence de la casse sur les noms de police : [Lien 108](#).

1.3. Regroupement de sélecteurs

Il arrive que vous ayez un même bloc de déclarations à appliquer à plusieurs sélecteurs. Alors, au lieu de réécrire plusieurs fois les mêmes déclarations dans la feuille de style, vous pouvez utiliser le regroupement de sélecteurs. Pour ce faire il suffit de mettre les différents sélecteurs à la suite les uns des autres, simplement séparés par une virgule et de placer le bloc de déclaration à la suite comme nous l'avons vu plus haut. La règle s'applique alors à tous les sélecteurs indiqués.

Exemple de regroupement de sélecteurs :

```
p, h2, h3, div {
    border :          1px dashed black;
    color :           #228b22;
    background-color : orange;
}
```

Dans cet exemple, tous les paragraphes (balise `<p>`), titres de niveau 2 et 3 (balises `<h2>` et `<h3>`) et `<div>` se voient attribuer une bordure en pointillés de '1px', la même couleur de texte ('#228b22' soit la couleur verte) ainsi que la couleur orange en arrière-plan.

Il suffit qu'un seul des sélecteurs de la liste soit erroné pour invalider le regroupement de sélecteurs et donc l'ensemble de la règle. Le style ne s'appliquera alors à aucun des sélecteurs. Seul Internet Explorer dans ses versions 6 et 7 fait exception à l'application de ce principe.

Un sélecteur peut, par exemple, être erroné lorsqu'on commet une erreur dans l'orthographe d'un sélecteur de type, et plus généralement lorsqu'on écrit mal un des mots-clés utilisés dans les sélecteurs, comme le nom de l'attribut pour le sélecteur d'attribut, ou encore les noms des pseudo-éléments et pseudo-classes.

Dans l'exemple ci-dessous nous allons nous tromper dans l'orthographe du sélecteur de type `h2`, que nous allons écrire de la manière suivante : `h 2` (avec un espace entre les deux caractères). Cela va rendre invalide l'ensemble de la règle, le style ne s'appliquera donc à aucun des sélecteurs présents dans cette règle.

Voir l'exemple 06 sur le regroupement des sélecteurs : [Lien 109](#).

1.4. Commentaires css

Comme pour le langage (X)HTML, il est possible d'ajouter des commentaires dans les feuilles de style.

Un commentaire est constitué en ouverture d'une barre oblique (/) suivi d'un astérisque (*), puis du texte en commentaire, et en fermeture d'un astérisque suivi d'une barre oblique.

Un exemple de commentaire :

```
/* Ceci est un commentaire */
```

Tout le texte entre ces caractères (`/* */`) sera ignoré ; il ne sera par contre pas possible d'insérer dans le texte commenté les caractères astérisque suivi d'une barre oblique.

Exemple de commentaire non valide :

```
/*
Ceci est un commentaire non valide parce que les
caractères */ ne sont pas autorisés au sein d'un
commentaire
Voyez vous-même la coloration syntaxique de cet
exemple.
*/
```

Les commentaires sont multilignes. Il n'existe pas en CSS de commentaire de fin de ligne (Pour savoir ce qu'est un commentaire de fin de ligne consultez par exemple, la FAQ PHP : Qu'est-ce qu'un commentaire de fin de ligne ([Lien 110](#))).

Les commentaires sont bien sûr conseillés. Ils permettent par exemple d'expliquer le code, de préciser à quels éléments les règles CSS s'appliquent, de séparer les différentes parties, donc de structurer votre code, etc. Un code commenté permet de le maintenir et de le faire évoluer plus facilement. Enfin les commentaires permettent aussi de tester le code et ainsi de le déboguer. Nous allons voir concrètement comment faire avec des cas pratiques.

Comme indiqué un peu plus haut, les commentaires sont multilignes, ils peuvent donc s'étendre sur plusieurs lignes :

Un commentaire multiligne :

```
/*
    Section sur l'entête du document
    Ceci est un commentaire sur plusieurs lignes
*/
```

Ils peuvent se retrouver à de nombreux endroits dans le code, par exemple dans une règle :

Un commentaire dans une règle CSS :

```
div {
    text-indent :    2em;
    /* Ceci est un commentaire dans une règle */
    margin :         10px;
}
```

Dans une déclaration :

```
Un commentaire dans une déclaration :
div {
    border :          1px /* Ceci est un
commentaire dans une déclaration */ solid black;
}
```

Les commentaires permettent également de désactiver une ou plusieurs règles :

```
Un commentaire désactivant une règle :
/*
div {
    font-weight :      bold;
    font-style :       italic;
}
*/
```

Mais aussi de désactiver une déclaration dans une règle :

```
Un commentaire désactivant une déclaration :
li {
    padding :          0px;
    /* list-style-type : disc; */
}
```

1.5. Propriétés raccourcies

Le CSS permet pour un certain nombre de propriétés, de les regrouper en une seule et d'y spécifier toutes leurs valeurs. Cela permet d'avoir un code plus rapide à écrire et plus condensé, donc plus facile à lire, moins lourd et plus rapide à télécharger.

Les principales propriétés qui permettent une écriture abrégée sont les suivantes :

Propriété	Description
font	Concerne les propriétés que l'on peut appliquer aux polices. La propriété font permet de regrouper en une seule les propriétés font-style , font-variant , font-weight , font-size , line-height , et enfin font-family .
background	Concerne les propriétés que l'on peut appliquer à un arrière-plan. La propriété background permet de regrouper en une seule les propriétés background-color , background-image , background-repeat , background-attachment et background-position .
list-style	Concerne les propriétés que l'on peut appliquer aux listes. La propriété list-style permet de regrouper en une seule les propriétés list-style-type , list-style-position et list-style-image .
border	Concerne les propriétés que l'on peut appliquer aux bordures des boîtes, la propriété border va appliquer son style aux quatre côtés des boîtes. Elle permet de regrouper les propriétés border-width , border-style et border-color .

border-width	Concerne l'épaisseur des bordures que l'on peut appliquer aux quatre côtés d'une boîte. La propriété border-width permet de regrouper en une seule les propriétés border-top-width , border-right-width , border-bottom-width et border-left-width .
border-style	Concerne le type de style des bordures que l'on peut appliquer aux quatre côtés d'une boîte. La propriété border-style permet de regrouper en une seule les propriétés border-top-style , border-right-style , border-bottom-style et border-left-style .
border-color	Concerne la couleur des bordures que l'on peut appliquer aux quatre côtés d'une boîte. La propriété border-color permet de regrouper en une seule les propriétés border-top-color , border-right-color , border-bottom-color et border-left-color .
margin	Concerne les propriétés d'espacement que l'on peut appliquer aux quatre bords extérieurs d'une boîte. La propriété margin permet de regrouper en une seule les propriétés margin-top , margin-right , margin-bottom et margin-left .
padding	Concerne les propriétés d'espacement que l'on peut appliquer aux quatre bords intérieurs d'une boîte. La propriété padding permet de regrouper en une seule les propriétés padding-top , padding-right , padding-bottom et padding-left .
outline	Concerne les propriétés que l'on peut appliquer aux contours (autres que des bordures) appliqués aux objets visuels. La propriété outline permet de regrouper en une seule les propriétés outline-color , outline-style et outline-width .

Il ne s'agit pas ici de décrire en détail toutes ces propriétés, mais d'expliquer plutôt le principe de l'écriture abrégée.

Une propriété raccourcie permet de remplacer plusieurs propriétés par une seule qui va accepter toutes les valeurs des propriétés qu'elle remplace. Les différentes valeurs devront être indiquées soit dans un ordre précis, soit dans n'importe quel ordre voulu. De même certaines d'entre elles seront à indiquer obligatoirement, les autres seront alors facultatives.

Pour décrire les propriétés raccourcies nous allons prendre l'exemple de la propriété **font**, mais avant cela il faut expliciter la nomenclature utilisée pour déclarer les valeurs des propriétés. Enfin nous verrons le cas particulier des propriétés permettant d'appliquer leur style aux quatre côtés des boîtes.

1.5.1. Explication de la nomenclature du W3C

Le W3C a introduit une nomenclature pour décrire la manière dont les valeurs des propriétés doivent être déclarées. Cette nomenclature vous servira tout au long de ce cours, et plus encore si vous voulez vous plonger dans les spécifications du W3C.

C'est cette nomenclature que nous allons voir maintenant en prenant quelques exemples de propriétés raccourcies.

La syntaxe de la propriété font :

```
[ <'font-style'> || <'font-variant'> || <'font-weight'> ]? <'font-size'> [/ <'line-height'> ]? <'font-family'>
```

La syntaxe de la propriété margin :

```
<marge-largeur>{1,4} | inherit
```

Ces exemples nécessitent les explications suivantes :

- les crochets [] servent aux regroupements ;
- une barre | distingue deux ou plusieurs options : seule l'une d'entre elles doit survenir ;
- une barre double || distingue également deux ou plusieurs options, à la différence que l'une de ces options doit survenir et plusieurs peuvent survenir quel que soit leur ordre ;
- plusieurs mots juxtaposés signifient que tous doivent survenir dans l'ordre donné.

Il existe aussi des modificateurs :

- un point d'interrogation (?) signifie que ce qui le précède est optionnel ;
- un astérisque (*) signifie que ce qui le précède peut survenir de zéro à plusieurs fois ;
- un plus (+) signifie que ce qui le précède survient une ou plusieurs fois ;
- une paire de nombres entre accolades {A,B} signifie que ce qui précède survient au moins A fois et au plus B fois.

1.5.2. Application à une propriété : la propriété font

Pour bien comprendre ce système nous allons décortiquer la propriété raccourcie **font** qui va permettre de remplacer les propriétés **font-style**, **font-variant**, **font-weight**, **font-size**, **line-height** et **font-family**.

Après ces explications nous pouvons traduire la ligne ci-dessous :

Les propriétés font-style, font-variant et font-weight :

```
[ <'font-style'> || <'font-variant'> || <'font-weight'> ]?
```

Ceci signifie que ces trois propriétés sont regroupées (du fait des caractères '[]'), elles sont facultatives (du fait du caractère '?') et elles ne doivent pas forcément survenir toutes les trois ensemble (du fait des caractères '|') : soit on en indique une, soit deux, soit les trois et ceci dans l'ordre désiré.

La propriété font-size :

```
<'font-size'>
```

Il faut ensuite indiquer **font-size**, c'est-à-dire la taille de la police, qui est obligatoire du fait de l'absence du caractère '?'.

La propriété line-height :

```
[/ <'line-height'> ]?
```

Cette partie n'est pas obligatoire en raison de la présence du point d'interrogation, mais si nous l'indiquons, il faut séparer la valeur de **font-size** et celle de **line-height** avec la barre oblique (présence du caractère '/' entre les crochets et avant la propriété **line-height**).

La propriété font-family :

```
<'font-family'>
```

Enfin la propriété **font-family** est obligatoire du fait de l'absence du point d'interrogation.

En résumé, seul **font-size** et **font-family** sont obligatoires. Pour les autres valeurs que vous avez omises, elles conservent leur valeur initiale.

Toutes les propriétés en relation avec font :

```
p#exemple {
    font-family :      Arial, Verdana, sans-serif;
    font-size :       1.2em;
    font-weight :     bold;
    font-style :      italic;
    font-variant :    normal;
    line-height :     1.5em;
}
```

Cet exemple se réduira à ceci en utilisant les propriétés abrégées :

Les propriétés en relation avec font réunies en une seule :

```
p#exemple {
    font :             italic bold
1.2em/1.5em Arial, Verdana, sans-serif;
}
```

Vous remarquerez que la propriété **font-variant**, qui a la valeur **'normal'** dans le premier exemple, n'est pas présente dans la propriété raccourcie et ce pour la simple raison qu'elle a la valeur par défaut. Dans ce cas il n'est pas nécessaire de spécifier explicitement la valeur par défaut : une propriété absente dans la propriété raccourcie prend automatiquement sa valeur par défaut. Et cette valeur par défaut trouvera à s'appliquer même si une valeur contraire est appliquée à l'élément via l'héritage.

Voir l'exemple 07 sur les propriétés raccourcies : exemples avec la propriété font : [Lien 111](#).

1.5.3. Le cas des propriétés s'appliquant aux quatre bords d'une boîte

D'autres propriétés raccourcies s'appliquent directement aux boîtes que vont créer les balises (X)HTML. Elles sont particulières dans le sens où les propriétés peuvent avoir jusqu'à quatre valeurs différentes s'appliquant aux quatre côtés de ces boîtes. Il faut donc expliquer comment vont se répartir ces valeurs entre les quatre côtés.

Cela concerne les propriétés raccourcies **margin**, **padding**, ainsi que **border-width**, **border-style** et **border-color**.

Ces propriétés acceptent de une à quatre valeurs :

- pour la propriété **margin** : ce sont les propriétés **margin-top**, **margin-left**, **margin-bottom** et **margin-right** ;
- pour la propriété **padding** : ce sont les propriétés **padding-top**, **padding-left**, **padding-bottom** et **padding-right** ;
- pour la propriété **border-color** : ce sont des valeurs de couleur ou la valeur transparent (de une à quatre valeurs) ;
- pour la propriété **border-style** : ce sont des valeurs de style (de une à quatre valeurs) ;
- pour la propriété **border-width** : ce sont des valeurs d'épaisseur (de une à quatre valeurs).

Les valeurs sont spécifiées dans l'ordre horaire en partant de midi. Vous pouvez vous référer à la liste ci-dessus indiquant les propriétés qui sont regroupées dans la propriété raccourcie : on indique d'abord la valeur s'appliquant à **'top'** (haut), puis **'right'** (droite), puis **'bottom'** (bas) et enfin **'left'** (gauche).

Prenons l'exemple de la propriété **border-color** parce qu'elle permet de très bien distinguer les côtés en fonction des couleurs appliquées :

Indication de quatre valeurs :

```
div {  
    border-color :    red blue green yellow;  
}
```

Si vous indiquez quatre valeurs : la première vaut pour le haut, la seconde pour la droite, la troisième pour le bas et la quatrième pour la gauche.



Quatre valeurs

Indication de trois valeurs :

```
div {  
    border-color :    red blue green;  
}
```

Si vous indiquez trois valeurs : la première vaut pour le haut, la seconde pour la droite et la gauche, la troisième vaut pour le bas.

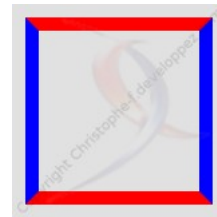


Trois valeurs

Indication de deux valeurs :

```
div {  
    border-color :    red blue;  
}
```

Si vous indiquez deux valeurs : la première vaut pour le haut et le bas, la seconde pour la droite et la gauche.



Deux valeurs

Indication d'une valeur :

```
div {  
    border-color :    red;  
}
```

Si vous indiquez une valeur : elle vaut pour les quatre côtés.



Une valeur

Consultez l'exemple 08 sur les propriétés raccourcies s'appliquant aux bordures des boîtes : [Lien 112](#).

Cette répartition dans le sens des aiguilles d'une montre ne concerne pas seulement la propriété **border-color**, mais s'applique à l'ensemble des propriétés listées ci-dessus : **margin**, **padding**, ainsi que **border-width**, **border-style** et bien sûr **border-color**.

1.6. Les règles at-rules

Les règles at-rules sont des règles spécifiques qui débutent par le caractère arobase (@) suivi d'un identifiant. Elles permettent de regrouper un ensemble de règles qui vont s'appliquer à des domaines spécifiques.

Ces règles sont au nombre de cinq :

- la règle **@import** qui permet de gérer l'importation de fichiers css ;
- la règle **@media** qui permet d'appliquer des règles css à un type de média précis ;
- la règle **@charset** qui permet de définir un encodage au fichier css ;
- la règle **@font-face** qui permet d'insérer une police externe ;
- la règle **@page** qui permet d'appliquer des styles aux médias paginés.

1.6.1. La règle @import

La règle **@import** permet, en lieu et place de la balise `<link>`, de lier des feuilles de style à un document (X)HTML. Elle peut tout autant être insérée dans le fichier (X)HTML que dans un fichier CSS.

Cette règle sera étudiée dans le prochain article.

1.6.2. La règle @media

La règle **@media** permet d'appliquer un ensemble de règles en fonction du type d'agent utilisateur qui va afficher la page. Ce peut être un navigateur (média 'screen'), l'impression de la page web (média 'print'), un synthétiseur vocal (média 'speech')...

Ainsi dans l'exemple suivant, toutes les règles déclarées à l'intérieur de la règle **@media** ne vont s'appliquer que si vous imprimez la page :

Mise en œuvre de la règle @media pour l'impression des pages :

```
@media print {
    #menu {
        display :    none;
        border :    0 none;
    }
    p {
        background-color : none;
    }
}
```

1.6.3. La règle @charset

La règle **@charset** permet de déclarer le codage qui est utilisé dans le fichier CSS extérieur. Précisez-le au début du fichier CSS externe et de la manière suivante :

La règle @charset, premier exemple :

```
@charset "UTF-8";

/* Les différentes déclarations CSS */
```

La règle @charset, second exemple :

```
@charset "ISO-8859-15";

/* Les différentes déclarations CSS */
```

Cette règle ne doit s'appliquer qu'aux fichiers CSS externes et non aux règles intégrées dans un fichier (X)HTML car elle est inutile dans ce cas (les navigateurs Safari et Google Chrome peuvent même rencontrer des soucis dans ce cas d'utilisation). Vous ne pourrez indiquer qu'une seule règle **@charset** par fichier CSS.

De même que la règle **@import**, cette règle at-rule doit être la première du document. Il ne doit y avoir ni ligne, ni aucun caractère avant, cela inclut également les

commentaires qui sont donc également interdits.

Vous pouvez retrouver la liste des codages autorisés ici : liste des codages autorisés ([Lien 113](#)).

1.6.4. La règle @font-face

La règle **@font-face** permet d'afficher une police originale sur votre site Internet. En principe pour que la police que vous avez choisie grâce à la propriété **font-family** s'affiche effectivement, il faut qu'elle soit présente sur l'ordinateur du visiteur.

Pour mettre un terme à cette limitation nous avons à notre disposition la règle at-rule **@font-face**, qui permet d'appeler une police que vous aurez au préalable placée sur le serveur et qui sera téléchargée sur le poste local (mise en cache).

Un exemple d'utilisation de @font-face :

```
@font-face {
    font-family :    nom-dappel;
    src :            url(nom-police-sur-le-
serveur.eot);
}

/* appel à cette police pour les paragraphes : */
p {
    font-family :    nom-dappel;
}
```

La propriété **font-family**, dans le cadre de la règle **@font-face**, sert à attribuer un nom quelconque à la police. Ce nom servira dans le document pour faire appel à cette police, comme vous le montre l'exemple ci-dessus dans la règle s'appliquant au paragraphe. La propriété **src** sert à indiquer le chemin vers le fichier de police se trouvant sur le serveur.

1.6.5. La règle @page

La règle **@page** est destinée aux médias paginés, et permet donc d'appliquer un style spécifique aux pages Web en mode impression. Dans cette configuration le modèle de mise en forme visuel est quelque peu modifié par la création d'un nouveau modèle de boîte : la boîte de page. Ce nouveau modèle de boîte permet de gérer les différents aspects spécifiques à une page qui sont de pouvoir déterminer la taille de la page, ses marges, les sauts de page...

Un exemple d'utilisation de la règle @page :

```
@page {
    size :            16cm 20cm;
    margin :          1cm;
}
```

Retrouvez l'article de 12monkeys en ligne : [Lien 114](#)

Introduction à HTML5

HTML5 est une évolution de la norme HTML dont le but avoué est de faciliter le développement d'interfaces utilisateur riches. HTML5 est beaucoup plus orienté applicatif que ses prédécesseurs et veut permettre de s'affranchir de plugins pour utiliser au maximum les technologies Web natives afin de construire une application riche.

1. HTML5, pourquoi ?

HTML5 est une évolution de la norme HTML regroupant un ensemble de technologies : SVG, CSS3, WebGL, File API, MathML...

Le but avoué de cette nouvelle version de HTML est de faciliter le développement d'interfaces utilisateur riches. HTML5 est beaucoup plus orienté applicatif que ses prédécesseurs et veut permettre de s'affranchir de plugins pour utiliser au maximum les technologies Web natives pour construire une application riche.

HTML5 arrive donc avec beaucoup de nouvelles capacités disponibles depuis une API vaste et variée. On pourra notamment communiquer avec des fonctions matérielles (micro, webcam, carnet d'adresses...) par l'intermédiaire d'API.

On peut facilement grâce à HTML5 visionner des vidéos sans plugin, ajouter des effets visuels aux textes, images, vidéos... On peut aussi utiliser des polices non standards, dessiner en SVG, faire de la 3D...

Voyons les choses qui me semblent être les plus intéressantes avec quelques exemples d'utilisation.

2. Nouvelle structuration

En premier lieu, HTML5 met largement à jour le système de balisage et donc la structuration d'une page. Voici à quoi pourrait ressembler une page-type utilisant certaines nouvelles balises pour améliorer la dimension sémantique du code :

```
<body>
  <header>
    <hgroup>
      <h1>Titre</h1>
      <h2>Sous-titre</h2>
    </hgroup>
  </header>

  <nav>
    <ul>
      Navigation
    </ul>
  </nav>
  <section>
    <article>
      <header>
        <h1>Titre</h1>
      </header>
      <section>
        Contenu...
      </section>
    </article>
  </article>
```

```
<header>
  <h1>Titre</h1>
</header>
<section>
  Contenu
</section>
</article>
</section>

<aside>
  Liens externes
</aside>

<figure>
  
  <figcaption>Graphique</figcaption>
</figure>

<footer>
  Copyright © Synbioz
  <time datetime="2011-12-08">2011</time>.
</footer>
</body>
```

Comme vous pouvez le voir, de nouvelles balises sémantiques sont utilisées pour regrouper les différents éléments en groupes logiques. Le principal avantage de ce balisage amélioré est de simplifier la lecture des pages par les moteurs de recherche qui seront alors en mesure de mieux classer et restituer les informations lors d'une recherche.

Les balises ajoutées l'ont aussi été pour simplifier l'écriture de la structure d'une application Web. Jusque-là on utilisait des div avec des id ou des classes pour représenter un entête, un menu, un article, une barre de navigation ou un pied de page. Pas très facile pour un moteur de recherche de faire la différence entre une pub, un menu ou l'article dans une page.

C'est donc pour pallier ce problème, faciliter et alléger le code que des balises sont apparues : header, section, article, nav, aside, footer...

L'attribut "rel", bien trop peu employé à mon goût, comporte lui aussi son lot de nouveautés :

- alternate : lien vers une version alternative de la page (version imprimable, traduction, miroir...);
- author : lien vers l'auteur du document ;
- bookmark : URL permanente pouvant servir de bookmark ;
- external : lien vers une page externe ;
- help : lien vers un document d'aide ;
- license : lien vers les copyrights du document ;
- next : prochain document dans la sélection (ex. : prochain article) ;

- prev : document précédent dans la sélection (ex. : article précédent) ;
- nofollow : lien vers un document tel qu'un lien commercial (Google utilise cet attribut pour ne pas suivre un lien) ;
- noreferrer : demande au navigateur de ne pas envoyer de référant dans les entêtes HTTP si l'utilisateur clique sur le lien ;
- prefetch : demande au navigateur de précharger la page liée ;
- search : lien vers un outil de recherche pour le document ;
- tag : permet d'affecter un tag (mot-clé) au document.

- des personnes ou entreprises ;
- des produits ;
- ...

Un exemple vaudra mieux qu'un long discours :



Vous voyez donc que sous le lien, on retrouve des étoiles représentant une note, l'auteur et la date.

Le code de cette page pourrait être, dans sa forme la plus simple :

Quelques exemples :

```
<link rel="alternate" type="application/rss+xml"
href="http://blog.com/feed"/>
<link rel="icon" href="/favicon.ico"/>
<link rel="pingback"
href="http://blog.com/xmlrpc"/>
<link rel="prefetch"
href="http://blog.com/main"/>

<a rel="archives"
href="http://blog.com/archives">anciens
articles</a>
<a rel="external"
href="http://notmysite.com">lien externe</a>
<a rel="license"
href="http://www.apache.org/licenses/LICENSE-
2.0">licence</a>
<a rel="nofollow"
href="http://notmysite.com/somewhere">pub</a>
<a rel="tag"
href="http://blog.com/category/ruby">Articles à
propos de Ruby</a>
```

```
<div>
<h1>Dragon Age</h1>
Avis rédigé par GameSpot le 3 novembre
Note: 5 sur 5
</div>
```

Vous pouvez tester vos pages utilisant des microdata via les outils Google ([Lien 115](#)) pour voir ce que donnera un résultat de recherche pointant sur votre page.

4. Les formulaires

Certainement l'une des choses les plus utilisées dans toute application Web, les formulaires ne sont pas en reste côté améliorations. Jusque-là, les contrôles disponibles et surtout les types de données associées étaient très limités. HTML5 nous apporte quelques briques manquantes qui vont faciliter la récupération et le traitement de données.

Vous devez vous dire que l'intérêt de ces nouveautés est limité aux moteurs de recherche qui voient leur travail simplifié. C'est vrai en partie mais n'oubliez pas qu'être bien référencé c'est avoir une meilleure présence sur le Web et donc plus de visiteurs. Il est également plus simple pour vous de créer une ossature HTML compréhensible ce qui vous facilitera le travail de référencement, d'accessibilité pour vos visiteurs déficients et la maintenabilité.

Maintenant les formulaires peuvent plus ou moins être typés. L'apparition de ces nouveaux types rend le code plus compréhensible et plus accessible. Ce typage permet également au navigateur de proposer des contrôles adaptés à la situation. En effet le navigateur pourra faire du préremplissage ciblé, utiliser une interface appropriée (ex. : calendrier pour les dates, colorpicker...) ou encore afficher un clavier adapté au contexte lorsqu'on utilise un téléphone.

Une structure claire, naturelle et légère est la base d'un bon site et je pense que HTML5 va largement contribuer à l'amélioration sémantique des applications développées.

Voici les nouveaux types d'input disponibles :

3. Microdata

- datetime, month, time, week...
- number ;
- search ;
- range ;
- color ;
- tel ;
- email.

Les microdata sont un autre paramètre à prendre en compte pour un meilleur référencement. Les microdata sont un balisage sémantique avec un vocabulaire personnalisé qui permet de détailler très finement un élément dans la page. Ces détails pourront être repris par le moteur de recherche pour présenter des résultats plus pertinents et détaillés. On pourra donc par exemple détailler :

Les widgets associés à ces nouveaux types sont particulièrement bien développés sur téléphone mobile où l'intérêt ergonomique est le plus perceptible.

- un avis ;
- une note ;
- un fil d'Ariane ;
- des événements ;

Des attributs ont aussi été ajoutés :

- placeholder qui permet d'avoir un texte par défaut qui disparaîtra automatiquement lors de la saisie utilisateur sans avoir la moindre ligne de JavaScript à écrire ;
- autofocus qui donnera le focus à ce champ au

chargement de page ;

- required qui définit le champ comme obligatoire ;
- pattern qui permet de définir un format qui devra être respecté pour valider le formulaire.

Voici quelques exemples d'utilisation :

```
<input type="text" required />

<input type="email" value="votre@email.com" />

<input type="date" min="2010-01-01" max="2011-12-31" value="2010-10-31"/>

<input type="range" min="0" max="50" value="10" />

<input type="search" results="10" placeholder="Recherche..." />

<input type="tel" placeholder="1234567890" pattern="^\d{10}$" />

<input type="color" placeholder="ex. #bbbbbb" />

<input type="number" step="1" min="-5" max="10" value="0" />
```

Parmi les possibilités moins souvent évoquées, les formulaires peuvent maintenant être découpés et être soumis de façons différentes.

En effet un champ peut maintenant se trouver hors du formulaire en lui-même, simplement en spécifiant l'attribut `form="id_formulaire"`.

Un même formulaire peut aussi être soumis de façon différente, en modifiant son comportement directement depuis un input de type submit.

Cela peut être très pratique par exemple dans un moteur de blog ou à l'écriture d'un article, vous pouvez soit l'enregistrer, soit en faire une prévisualisation.

Exemple :

```
<form id="new_post" action="/articles" method="post">
  <input type="submit"
  formaction="/articles/preview" formmethod="get"
  value="Aperçu" />
</form>

<input type="submit" form="new_post" value="Publier" />
```

5. Audio et vidéos natifs

Plus besoin d'intégrer un lecteur flash pour pouvoir lire du son ou de la vidéo dans une page Web. HTML5 intègre cette possibilité nativement. Une simple balise permet de lire une vidéo et d'afficher ses contrôles ! Il vous est même possible de contrôler tout cela directement en JavaScript :

```
/* Joue un son en boucle au chargement de page en
affichant ses contrôles */
<audio src="/audio/test.ogg" autoplay controls
loop>
```

```
/* Joue une vidéo en boucle au chargement de page
en affichant ses contrôles */
<video id="vid" src="/videos/test.ogv" autoplay
controls loop>
```

```
<script type="text/javascript" charset="utf-8">
  var video = document.getElementById('vid');
  video.pause();
  alert(video.src);
  /* Durée du son */
  alert(video.duration);
  /* Position actuelle */
  alert(video.currentTime);
  /* On redéfinit la position courante dans la
  lecture */
  video.currentTime=35;
  /* On reprend la lecture */
  video.play();
</script>
```

6. Canvas

Je n'ai encore pas eu l'occasion de jouer avec les éléments de type Canvas, qui permettent de dessiner point par point via une API JavaScript, mais les démos disponibles sur le Web sont vraiment impressionnantes ([Lien 116](#) et [Lien 117](#)). On imagine facilement les applications possibles. Il devient facile de dessiner en temps réel des graphiques animés. Tout cela sans Flash ni plugin, simplement de l'HTML5, un peu de CSS3 et du JavaScript !

7. API JavaScript enrichie

Tout d'abord de nouveaux sélecteurs très pratiques font leur apparition.

7.1. Trouver des éléments via l'API DOM

```
var el = document.getElementById('section1');
el.focus();

var els = document.getElementsByTagName('div');
els[0].focus();

var els =
document.getElementsByClassName('section');
els[0].focus();
```

Vous connaissiez déjà `getElementById()` qui retourne un élément du DOM. `getElementsByTagName()` permet d'obtenir un tableau d'éléments correspondant à la balise passée en paramètre. `getElementsByClassName()` est l'équivalent pour la classe passée en paramètre.

7.2. Trouver des éléments via les sélecteurs CSS (API Selectors)

```
var els = document.querySelectorAll("ul li:nth-child(odd)");
var tds = document.querySelectorAll("table.test > tr > td");
var el = document.querySelector("table.test > tr > td");
```

`querySelector()` et `querySelectorAll()` permettent respectivement de récupérer le premier élément ou tous les

éléments qui correspondent au sélecteur CSS3 passé en paramètre.

7.3. Attributs personnalisés data-*

Ces attributs que vous pouvez définir sur n'importe quelle balise permettent de définir, stocker et récupérer des données personnalisées directement dans le DOM.

```
<div id="test" data-id="12" data-name="nico" data-screen-name="bounga"></div>

<script type="text/javascript" charset="utf-8">
  // Ajout d'un attribut via JS.
  var el = document.querySelector('#test');
  el.setAttribute('data-foo', 'bar');

  var html = [];
  // On récupère toutes les données personnalisées et on les affiche
  for (var key in el.dataset) {
    html.push(key, ': ', el.dataset[key], '<br>');
  }

  el.innerHTML = html.join('');
</script>
```

Le framework Ruby on Rails dans ses dernières versions fait déjà largement usage de ces attributs pour mettre en place du JavaScript non intrusif et indépendant de la bibliothèque JavaScript utilisée.

7.4. Les listes de classes

La manipulation des classes CSS appliquées à un élément a également été largement améliorée. Plus besoin d'un framework pour connaître les classes appliquées, en ajouter, en supprimer...

```
<div id="main" class="shadow rounded"></div>

<script type="text/javascript" charset="utf-8">
  var el =
  document.querySelector('#main').classList;
  el.add('highlight');
  el.remove('shadow');
  el.toggle('highlight');

  el.contains('highlight'); // false
  el.contains('shadow'); // false
  el.classList.toString() == el.className; // true
</script>
```

7.5. Cookies on stéroïds: localStorage et sessionStorage

Un moyen bien plus puissant que les cookies de stocker des données. Le localStorage permet un stockage permanent alors que le sessionStorage permet un stockage par onglet (ou session).

```
// Enregistre le contenu d'un textarea au clic sur le bouton
saveButton.addEventListener('click', function ()
{
  window.localStorage.setItem('value', area.value);
  window.localStorage.setItem('timestamp', (new
```

```
Date()).getTime());
}, false);

// Préremplit le textarea en fonction de la valeur stockée en local
textarea.value =
window.localStorage.getItem('value');
```

7.6. Bases de données WebSQL

Encore plus puissant, il est maintenant possible d'avoir une base de données SQL côté client. On peut y stocker des données utilisables hors-ligne, relatives à l'utilisateur qui pourraient donc être chargées sans faire appel au serveur. Très pratique pour stocker des préférences utilisateur par exemple.

```
var db = window.openDatabase("ma_base", "1.0", "description", 5*1024*1024); //5MB

db.transaction(function(tx) {
  tx.executeSql("SELECT * FROM test", [], successCallback, errorCallback);
});
```

7.7. API de cache

Cette API permet de rendre disponibles certaines ressources en mode hors-ligne grâce au cache applicatif.

Il suffit de déclarer votre balise HTML en précisant l'emplacement du manifeste :

```
<html manifest="cache.appcache">
```

Le manifeste ressemblerait lui à quelque chose comme :

```
CACHE MANIFEST
# version 1.0.0

CACHE:
/html5/src/logic.js
/html5/src/style.css
/html5/src/background.png

NETWORK:
*
```

On rend donc disponibles hors-ligne les ressources listées dans la section CACHE pour tous les réseaux.

Il ne reste plus qu'à piloter via JavaScript :

```
window.applicationCache.addEventListener('updateReady', function(e) {
  if (window.applicationCache.status == window.applicationCache.UPDATEREADY) {
    window.applicationCache.swapCache();
    if (confirm('A new version of this site is available. Load it?')) {
      window.location.reload();
    }
  }
}, false);
```

7.8. Les sockets Web

HTML5 ouvre la possibilité d'avoir des communications bidirectionnelles. Le serveur et le client peuvent envoyer des données dès qu'ils en ont besoin. Il est même possible de le faire en simultané. Seules les données sont envoyées sans en-têtes ce qui réduit l'utilisation de la bande passante et améliore donc les temps de réponse.

Côté serveur il vous faudra souscrire à un service Web qui propose la gestion des Websockets ou vous en charger de votre côté avec Socket.IO-Node par exemple : [Lien 118](#).

```
var socket = new WebSocket('ws://site.org/echo');

socket.onopen = function(event) {
    socket.send('Hello, WebSocket');
};

socket.onmessage = function(event) {
    alert(event.data);
};

socket.onclose = function(event) {
    alert('closed');
};
```

Notre JavaScript d'exemple contacte le serveur et crée une WebSocket. Une fois la socket ouverte (onopen) le client envoie un message au serveur. Si un message est reçu du serveur sur cette WebSocket, le client l'affiche. Si la socket est fermée, un message sera affiché par le client.

On peut donc imaginer des outils comme des chats en temps réel, ultraréactifs et beaucoup moins gourmands en bande passante que des modèles basés sur un appel en boucle du client au serveur. Avec les WebSockets, c'est le serveur qui contacte le client pour lui dire qu'il y a des données à traiter !

7.9. Géolocalisation

Il est maintenant possible de localiser un utilisateur (s'il l'autorise) via GPS, 3G ou IP. Une fonctionnalité qui semble clairement avoir été mise en place pour les appareils mobiles en vue de créer des applications sociales avec géolocalisation.

```
if (navigator.geolocation) {
    navigator.geolocation.getCurrentPosition(function(
        position) {
        var latLng = new
        google.maps.LatLng(position.coords.latitude,
        position.coords.longitude);
        var marker = new
        google.maps.Marker({position: latLng, map: map});
        map.setCenter(latLng);
    }, errorHandler);
}
```

Avec ce simple morceau de code, sans bibliothèque externe ou appel d'une API, une carte centrée (Google Map) sur la position de l'utilisateur sera affichée.

7.10. Accès à l'historique

L'un des plus gros soucis rencontrés par les développeurs avec AJAX est la perte de l'historique navigateur. Il est aussi impossible de bookmarker un contenu chargé en AJAX. Avec l'API d'historique vous pourrez facilement pallier ces problèmes en manipulant les URL et

l'historique depuis JavaScript.

```
// Parcourir l'historique
window.history.back();
window.history.go(-1);

window.history.forward();
window.history.go(1);

window.history.length;

// Modifier l'historique
var stateObj = { foo: "bar" };

// Ajout d'un élément à l'historique,
modification de l'URL, modification du titre
// rien n'est chargé, seuls l'historique et la
barre d'URL sont impactés
history.pushState(stateObj, "titre",
"foo/bar.html");

// Ici on remplace l'élément d'historique plutôt
que d'en ajouter un
history.replaceState(stateObj, "titre 2",
"foo/bar2.html");

// Info sur l'élément courant
history.state;
```

Un événement `popstate` est envoyé à chaque fois que l'entrée courante change. Si cet élément avait été ajouté via un appel à `pushState` ou `replaceState`, alors l'événement contiendra une copie de l'état qui avait été passée en créant l'entrée. Cela peut être intéressant si à chaque changement de page vous devez remettre du contenu dans son état initial :

```
window.addEventListener('popstate',
function(event) {
    document.querySelector('h1').innerText =
event.state.title;
});
```

7.11. Gestion des fichiers

Depuis HTML5, pour peu que l'utilisateur vous accorde les droits, vous pouvez écrire dans un système de fichier virtuel cloisonné (*sandbox*). Cette nouveauté offre de nombreuses possibilités pour les applications Web. Dans notre exemple, nous écrirons des logs sur le disque du client :

```
/* On crée un fichier disponible de manière
permanente d'une taille estimée de 5Mo */
window.requestFileSystem(window.PERSISTENT,
5*1024*1024, function(fs) {
    fs.root.getFile('log.txt', {create: true},
function(fileEntry) {

    fileEntry.createWriter(function(writer) {
        writer.onwrite = function(e) { };
        writer.onerror = function(e) { };

        var bb = new BlobBuilder();
        bb.append('je log !');

        writer.write(bb.getBlob('text/plain'));
    }, errorHandler);
}
```

```

}, errorHandler);

/* Gestionnaire d'erreur d'accès au système de
fichier virtuel */
function errorHandler(e) {
    var msg = '';

    switch (e.code) {
        case FileError.QUOTA_EXCEEDED_ERR:
            msg = 'QUOTA_EXCEEDED_ERR';
            break;
        case FileError.NOT_FOUND_ERR:
            msg = 'NOT_FOUND_ERR';
            break;
        case FileError.SECURITY_ERR:
            msg = 'SECURITY_ERR';
            break;
        case FileError.INVALID_MODIFICATION_ERR:
            msg = 'INVALID_MODIFICATION_ERR';
            break;
        case FileError.INVALID_STATE_ERR:
            msg = 'INVALID_STATE_ERR';
            break;
        default:
            msg = 'Unknown Error';
            break;
    };

    console.log('Error: ' + msg);
}

```

Il est vraiment possible de gérer ce pseudosystème de fichiers comme un vrai système de fichiers. Les possibilités sont nombreuses et je vous recommande de lire cette documentation sur le sujet : [Lien 119](#).

7.12. Webworkers

Les Webworkers peuvent, en quelque sorte, être comparés au multithreading. L'API des Webworkers permet d'exécuter des scripts en tâche de fond, sans bloquer le client : [Lien 120](#). Un script s'exécutant dans un Webworker ne sera donc pas bloquant pour l'affichage de la page.

```

/* index.js: */

/* On crée un worker */
var worker = new Worker('worker.js');

/* On définit son comportement s'il reçoit un
message */
worker.addEventListener("message",
function(event) {
    alert(event.data);
}, false);

/* On envoie un message */
worker.postMessage('Nico');

/* worker.js: */

self.onmessage = function(event) {
    self.postMessage("Bonjour : " + event.data .
" !");
};

```

8. CSS3

8.1. Les nouveaux sélecteurs CSS: nth-child(even), nth-child(odd), :not, first-child ?

Un nombre impressionnant de sélecteurs ont été ajoutés et ils s'avèrent vraiment pratiques :

```

.row:nth-child(even) {
    background: #dde;
}
/* Tous les éléments impairs ayant pour classe
"row" */
.row:nth-child(odd) {
    background: white;
}

/* N'ayant pas la classe "box" */
:not(.box) {
    color: #00c;
}
/* Tous les éléments non span */
:not(span) {
    display: block;
}

/* Premier h2 si celui-ci est le premier élément
du site */
h2:first-child { }

/* Sélectionne le premier div fils des div ayant
la classe "text" */
div.text > div { }

/* Sélectionne les éléments header suivant
directement un h2 */
h2 + header { }

/* Tous les inputs de type text */
input[type="text"]{ }

```

8.2. La gestion des polices

CSS3 nous ouvre la possibilité d'utiliser des polices non standards, qu'elles soient installées ou non chez le client :

```

@font-face {
    font-family: 'LeagueGothic';
    src: url(LeagueGothic.otf);
}

@font-face {
    font-family: 'Droid Sans';
    src: url(Droid_Sans.ttf);
}

header {
    font-family: 'LeagueGothic';
}

```

8.3. Gestion de colonnes

Combien de fois avez-vous dû répartir un contenu sur plusieurs colonnes ? Une tâche fastidieuse en CSS qui peut s'avérer compliquée à mettre en œuvre de manière portable et flexible. CSS3 apporte une solution élégante :

```
-webkit-column-count: 2;
-webkit-column-rule: 1px solid #bbb;
-webkit-column-gap: 2em;
```

8.4. Maîtrise de l'opacité

En CSS3, il est possible de définir les couleurs en RGBA. On a donc en plus des habituels canaux rouge, vert et bleu accès au canal "alpha" qui permet de définir l'opacité de la couleur.

```
/* Opacité à 75 % */
color: rgba(255, 0, 0, 0.75);
background: rgba(0, 0, 255, 0.75);

/* Opacité à 50 % */
hsla(240, 100%, 70%, 0.5);
```

8.5. Bordures arrondies

Je rêvais de cette possibilité depuis des années. Étant clairement un manchot de Photoshop, je me suis longtemps demandé pourquoi il n'était pas possible d'arrondir les angles des div avec une simple ligne de CSS. Oubli réparé, fini les images de fond pour les coins de vos blocs. CSS3 intègre la gestion des angles des bordures :

```
/* Spécifique à webkit (Safari, Chrome) */
-webkit-border-radius: 3px;
-webkit-border-bottom-right-radius: 5px;
-webkit-border-bottom-left-radius: 5px;

/* Spécifique à Mozilla (Firefox) */
-moz-border-radius: 3px;
-moz-border-radius-bottomright: 5px;
-moz-border-radius-bottomleft: 5px;

/* Future norme ? */
border-radius: 3px;
border-bottom-right-radius: 5px;
border-bottom-left-radius: 5px;
```

8.6. Dégradés

Encore une fois vous allez pouvoir oublier les images utilisées en arrière-plan pour simuler un dégradé. CSS3 le gère nativement :

```
/* Dégradé linéaire du haut vers le bas, de vert à blanc */
background: -webkit-gradient(linear, left top, left bottom,
                             from(#00abeb),
                             to(white),
                             color-stop(0.5, white), color-stop(0.5, #66cc00));

/* Dégradé radial avec des valeurs absolues */
background: -webkit-gradient(radial, 430 50, 0, 430 50, 200, from(red), to(#000));
```

8.7. Fonds redimensionnables

Il était possible, jusque-là, d'ajouter une image de fond à un élément. On pouvait l'afficher une fois dans sa taille originale et également répéter le motif sur l'axe des x et/ou y. Une amélioration est apportée par CSS3 puisque maintenant il est possible de dire comment doit se comporter l'image. On peut donc la forcer à prendre toute

la taille d'un bloc, à être contenue dans le bloc, avoir sa taille originale ou encore définie à un pourcentage.

```
#logo {
  background: url(logo.gif) center center no-repeat;
  /* Valeurs possibles : auto / contain / cover / 100%
  background-size: cover;
}
```

8.8. Fonds multiples

CSS3 permet de définir plusieurs images de fond pour un même élément :

```
div {
  background: url(/images/logo.png) 10px center no-repeat,
             url(/images/stripes.png) 0 center repeat-x;
}
```

8.9. Ombrages

Vous aimez ajouter des ombres à vos textes et blocs, plus besoin d'images ! Encore une fois, CSS3 apporte une solution élégante et légère à ce cas de figure :

```
/* Ombre ajoutée à un texte utilisant potentiellement une police spécifique */
text-shadow: rgba(64, 64, 64, 0.5);

/* Ombre ajoutée à un bloc (div, etc) */
box-shadow: rgba(0, 0, 128, 0.25);
```

8.10. Transitions et transformation

Vous êtes habitué à ajouter des effets de transition aux actions qui se produisent dans votre application ? Alors je suppose que vous utilisez JavaScript pour créer l'effet visuel de transition d'un état à l'autre ! En CSS3 :

```
/*
TRANSITIONS
Les éléments #box auront un effet de transition lorsque le margin-left sera modifié
*/
#box {
  -webkit-transition: margin-left 1s ease-in-out;
}

/*
TRANSFORMATIONS
Rotation d'éléments du DOM
utile par exemple sur un :hover
*/
-webkit-transform: rotateY(45deg);
-webkit-transform: scaleX(25deg);
-webkit-transform: translate3d(0, 0, 90deg);
-webkit-transform: perspective(500px)

/*
ANIMATIONS
Fait "clignoter" tous les divs en augmentant la taille de leur police.
*/
@-webkit-keyframes pulse {
```



```
from {
  opacity: 0.0;
  font-size: 100%;
}
to {
  opacity: 1.0;
  font-size: 200%;
}
}

div {
  -webkit-animation-name: pulse;
  -webkit-animation-duration: 2s;
  -webkit-animation-iteration-count: infinite;
  -webkit-animation-timing-function: ease-in-out;
  -webkit-animation-direction: alternate;
}
```

9. Conclusion

Dans cet article, nous n'avons malheureusement pas eu le temps de faire un tour complet des nouveautés qu'offre HTML5 mais vous trouverez de nombreux sites qui traitent du sujet. Une des références incontournables est

évidemment un site à l'initiative de Google, HTML5 Rocks! ([Lien 121](#)). HTML5 Demos présente des cas concrets d'utilisation : [Lien 122](#).

Si vous comptez utiliser HTML5, il vous faut absolument jeter un œil au projet Modernizr ([Lien 123](#)) qui vous permettra de connaître les fonctionnalités supportées par le client. Cet outil vous facilitera la mise en place de solutions alternatives.

Il est indéniable que HTML5 a été pensé et taillé pour créer des applications et non plus de simples sites. HTML5 propose même un mode hors-ligne qui permet de surfer et interagir avec des parties du site sans connexion. Une fois une connexion rétablie, les données peuvent être synchronisées. Les possibilités offertes nativement sont désormais impressionnantes. Un vrai régal pour les développeurs d'applications Web. Pour les développeurs de sites vitrines c'est aussi une bonne nouvelle, les intégrations graphiques, les effets de transition et de transformation sont supportés nativement.

Retrouvez l'article de Synbioz en ligne : [Lien 124](#).

Variance en C# 4.0

Ceci est un article sur une nouveauté (fort discrète) de C# 4.0 : la covariance et la contravariance. Toujours bon à savoir pour briller en société!

1. Introduction

Une des nouveautés de .NET 4.0 est l'introduction de la covariance et contravariance.

Sous ces termes un peu ésotériques se cachent des concepts assez simples en réalité. Je vais donc tenter de démystifier tout ça avec cet article et des exemples simples.

Accrochez-vous, c'est parti !

2. Covariance

La covariance, qu'est-ce ?

Une définition générale pourrait être : conversion du plus large au plus fin. (Traduction littérale de l'article anglais de Wikipédia sur la variance : [Lien 125](#))

Nous voilà bien avancés ! Si on tente de traduire cela en langage de développeur : conversion du plus dérivé vers le moins dérivé. Mais encore. Un exemple sera plus clair.

2.1. Exemple de covariance

Ce code :

```
IEnumerable<object> test = new List<string>();
```

Ne compilera pas sous .NET 3.5 et donnera ce message d'erreur : « *Cannot implicitly convert type 'System.Collections.Generic.List<string>' to 'System.Collections.Generic.IEnumerable<object>'. An explicit conversion exists (are you missing a cast ?)* »

Pourtant `List<T>` implémente bien `IEnumerable<T>`, et `string` est bien dérivé d'`object`. Mais comme `IEnumerable<T>` n'est pas covariant, un `IEnumerable<string>` ne peut être considéré comme un `IEnumerable<object>` ! (cf. définition : du plus dérivé vers le moins dérivé).

Par contre, à partir de 4.0, pas de problème car `IEnumerable<T>` est covariant : le paramètre de sortie de type `T` peut être considéré comme un type `U` si `T` dérive de `U`.

Tentons une définition de la covariance : compatibilité d'assignation, pour une interface ou un délégué ayant un ou plusieurs paramètres génériques de sortie (voir plus bas) de type `T1, T2...` avec le même interface ou délégué ayant les paramètres génériques de sortie `U1, U2...` si `T1, T2...` dérivent de `U1, U2...`

Toutes les interfaces ne sont pas covariantes en .NET 4.0. Voici la liste de la liste de celles qui le sont :

- `IEnumerable<out T>` (T est covariant) : [Lien 126](#);
- `IEnumerator<out T>` (T est covariant) : [Lien 127](#) ;
- `IQueryable<out T>` (T est covariant) : [Lien 128](#) ;
- `IGrouping<out TKey, out TElement>` (`TKey` et `TElement` sont covariants) : [Lien 129](#) ;
- tous les délégués `Func<in T1... , out TResult>` (`TResult` est covariant).

Comme on le voit, la covariance ne concerne pas que les interfaces. Les délégués peuvent aussi être covariants :

```
public Func<string> stringfunc;  
Func<object> objectfunc = stringfunc;
```

Comme vu dans cet exemple, je peux utiliser une fonction qui renvoie `T` comme une fonction qui renvoie `U` si `T` est dérivé de `U`.

2.2. Création d'interfaces et de délégués covariants

Pour rendre une interface ou un délégué covariant, il faut ajouter le mot-clé `out` ([Lien 130](#)) au type paramètre concerné. Comme le mot-clé l'indique, celui-ci ne peut alors se trouver uniquement en paramètre de sortie.

2.2.1. Exemple d'interface covariant

```
public interface IFruitTree<out T>  
{  
    T GetFruit();  
}
```

Des classes implémentant cette interface :

```
public class Tree { }  
public class Fruit { }  
public class Orange : Fruit { }  
public class OrangeTree : Tree,  
    IFruitTree<Orange>  
{  
    public Orange GetFruit()  
    {  
        return new Orange();  
    }  
}
```

Et cette fonction :

```
public void CatchFruit(IFruitTree<Fruit>  
    fruittree)  
{  
    ///  
}
```

IFruitTree<out T> étant covariant sur T, nous pouvons passer, au lieu d'un objet implémentant **IFruitTree<Fruit>**, un objet implémentant un type plus dérivé, comme **IFruitTree<Orange>** :

```
var o = new OrangeTree();
CatchFruit(o);
```

Comme on le voit, cela simplifie les choses et permet une plus grande réutilisation des composants.

2.2.2. Exemple de délégué covariant

```
public delegate T CovariantDelegate<out T>();
void ShowCovariance()
{
    var stringdelegate = new
    CovariantDelegate<string>(ReturnString);
    CovariantDelegate<object> objectdelegate =
    stringdelegate;
    object result = objectdelegate.Invoke();
}
public string ReturnString()
{
    return "Covariance";
}
```

Comme déjà dit, un paramètre en **out** ne peut être qu'un paramètre de sortie. Par exemple :

```
public interface IsInhabited<out T>
{
    T DoSomething(T param);
}
```

Donnera une erreur de compilation :

Invalid variance: The type parameter 'T' must be contravariantly valid on 'Co_Contravariant.IsInhabited <T>. DoSomething (T)'. 'T' is covariant.

Pourquoi uniquement en paramètre de sortie ? Pour tenter d'expliquer simplement, si T est uniquement en sortie, cela ne pose pas de problème, car il n'y a pas de problème d'assigner à un type T (*object*, par exemple), le résultat d'une méthode qui renvoie un type U plus dérivé (*string*, par exemple).

Par contre, si on acceptait <T> en entrée, on pourrait dès lors se retrouver dans un cas comme celui-ci :

```
public Func<Exception, int> OKFunc;
public Func<object, int> ProblemsAhead;
void Main()
{
    OKFunc = CountStackTrace;
    ProblemsAhead = OKFunc;
    ...
    var something = ProblemsAhead(new object());
}
public int CountStackTrace(Exception e)
{
    return e.StackTrace.Count();
}
```

3. Contravariance

Le pendant de la covariance est la contravariance. Pour reprendre le même type de définition : conversion du plus fin au plus large. Traduction : du moins dérivé au plus dérivé.

Ça a l'air moins naturel à première vue, mais c'est tout à fait logique comme on va le constater :

3.1. Exemple de contravariance

Imaginons que nous ayons ceci :

```
public class Animal {}
public class Dog : Animal {}
```

Ainsi qu'un comparateur entre Animal qui implémente **IEqualityComparer<T>** (qui est contravariant) :

```
public class CompareAnimals :
    IEqualityComparer<Animal>
{
    public bool Equals(Animal x, Animal y)
    {
        .
    }
    public int GetHashCode(Animal obj)
    {
        .
    }
}
```

Si nous avons une liste de Dog, ceci est valide :

```
var dogs = new List<Dog>();
var distinct = dogs.Distinct(new
    CompareAnimals());
```

Alors que la signature de Distinct dans ce cas est : **Distinct<Dog>(IEqualityComparer<Dog> Comparer)**.

Nous passons donc un type moins dérivé (*IEqualityComparer<Animal>*) à la place d'un plus dérivé (*IEqualityComparer<Dog>*). Ça peut paraître paradoxal mais revenons à une définition plus affinée : conversion des paramètres d'entrées du moins dérivé vers le plus dérivé.

En effet : la covariance est uniquement sur les paramètres de sortie et la contravariance est uniquement sur les paramètres d'entrées. Distinct va appeler la méthode **Equal(Animal x, Animal y)** en lui passant deux instances de Dog alors qu'elle attend deux instances d'Animal. Mais, vu que Dog dérive d'Animal, ça ne pose pas de problème ! Ce qui semble contre nature est en fait tout à fait logique.

Tentons maintenant une définition de la contravariance : compatibilité d'assignation, pour une interface ou un délégué ayant un ou plusieurs paramètres génériques d'entrée de type **T1,T2...** avec le même interface ou délégué ayant les paramètres génériques d'entrée **U1,U2...** si **U1,U2...** dérivent de **T1,T2...**

Voilà la liste des interfaces contravariants de .NET 4.0 :

- **IComparer<in T>** (T est contravariant) : [Lien 131](#);

- `IEqualityComparer<in T>` (T est contravariant) : [Lien 132](#) ;
- `IComparable<in T>` (T est contravariant) : [Lien 133](#) ;
- tous les délégués `Action<in T>` , `Action<in T1...>` (T, T1... sont contravariants) ;
- tous les délégués `Func<in T1..., out TResult>` (T, T1... sont contravariants).

Comme pour la covariance, les délégués peuvent aussi être contravariants :

```
public Action<object> OKAction;
void ShowCovariance()
{
    OKAction = new Action<object>((o) =>
    o.ToString());
    DoSomethingWithAction(OKAction);
}
static void DoSomethingWithAction(Action<string>
action)
{
    action("Contravariance");
}
```

3.2. Création d'interfaces et de délégués contravariants

Pour créer une interface ou un délégué contravariant, il faut utiliser le mot-clé **in** ([Lien 134](#)). Comme le nom l'indique aussi, les paramètres en in ne peuvent être que des paramètres d'entrées.

3.2.1. Exemple d'interface contravariant

```
public interface ICompareWeigth<in T> where
T:Animal
{
    int Weight { get; set; }

    bool SameWeight(T animal);
}

public class CompareAnimalWeight :
ICompareWeigth<Animal>
{
    public int CompareWeight (Animal animal1,
Animal animal2)
    {
        // do something;
    }
}
```

Si on a cette fonction :

```
public int CompareDogWeight (ICompareWeigth<Dog>
comparison, Dog dog1, Dog dog2)
{
    return comparison.CompareWeight (dog1, dog2);
}
```

Ceci est valide :

```
CompareDogWeight (new CompareAnimalWeight (), new
Dog (), new Dog ());
```

3.2.2. Exemple de délégué contravariant

```
public delegate void ContravariantDelegate<in
T>(T param);
```

```
public void DoThing(object o){}
public void
TestContravariance(ContravariantDelegate<string>
del)
{
    del.Invoke("Contravariance");
}
void ShowContravariance()
{
    ContravariantDelegate<object> OKDelegate =
DoThing;
    TestContravariance(OKDelegate);
}
```

4. Remarques

- La covariance et contravariance ne concernent que les paramètres de type générique des délégués (depuis .NET 2.0) et des interfaces ainsi que les types tableaux ([]).

Notons que, pour les types tableaux, c'est à déconseiller car on peut alors arriver à ce genre de choses :

```
string[] stringarray = new[]{"Quelle est la
réponse ?"};
object[] objectarray = stringarray;
objectarray[0] = 42;
```

Qui compilera bien, mais il y aura une `ArrayTypeMismatchException` à l'exécution.

- La variance dans les génériques n'est supportée que par les types référence, et pas les types valeurs :

```
IEnumerable<int> intlist = new List<int>();
IEnumerable<object> objectlist = intlist; // Ne
compile pas
```

- Le support de la variance n'est implémenté qu'à partir de la version 5 de Silverlight.
- Le support de la variance n'est pas implémenté sous Windows Phone 7.
- Il n'est pas possible d'avoir un paramètre à la fois covariant et contravariant. Par contre, il est possible d'avoir un (ou plusieurs) paramètre(s) covariant(s), et un (ou plusieurs) contravariant(s). Dans le framework, les classes `Func<in T..., out TResult>` le sont. Un exemple :

```
Func<object, string> func = (o) => o.ToString();
Func<string, object> CoContra = func;
```

5. Conclusion

Voilà, j'espère avoir démystifié ces deux noms barbares.

Le fait de rendre ses interfaces ou délégués covariants et/ou contravariants permet de les réutiliser plus facilement, je suis donc du même avis que Jeffrey Richter, qui recommande donc d'utiliser **in** et **out** quand c'est possible afin de permettre une réutilisation dans plus de scénarios.

Retrouvez l'article d'Olivier Matis en ligne : [Lien 135](#)

Création d'un job avec l'ETL Talend Open Studio qui lance une MACRO Excel

Ce tutoriel va nous présenter comment réaliser un job, avec le fameux ETL Talend Open Studio, qui nous permettra de lancer une MACRO Excel.

1. Introduction

L'article suivant présente comment réaliser pas à pas un job Talend qui va créer un fichier Excel simple rempli à partir d'un fichier texte. Ce fichier résultat sera ensuite mis en forme grâce à une MACRO Excel.

Avant de commencer, je tiens à préciser qu'on n'abordera pas dans ce tutoriel le détail de création de job sous Talend vu que ça a été discuté dans cet autre tutoriel ([Lien 136](#)), ni la création de la MACRO Excel puisque ce n'est pas le but.

2. La source de données

La source de données est un simple fichier texte.

Voici un aperçu de ce dernier :

ID;NOM;RANG
1; ELHASSAK; 4
2; ESBAISS; 2
3; FRINDOU; 2
4; BAIDOUNE; 1
5; BENBARI; 4
6; STAMBOULI; 4

Le job qu'on va créer lira les données de ce fichier puis les enregistrera dans un fichier Excel.

3. La destination de données

La destination de données est un fichier Excel normal, sauf que ce dernier sera mis en forme avec une MACRO.

Il ressemblera à ça avant la mise en forme :

	A	B	C
1	id	Nom	Rang
2	ID1	NOM1	RANG1
3	ID2	NOM2	RANG2
4	ID3	NOM3	RANG3
5	ID4	NOM4	RANG4
6	ID5	NOM5	RANG5
7	ID6	NOM6	RANG6
8			

et à ça, après la mise en forme, c'est-à-dire après l'exécution de la MACRO :

	A	B	C
1	ID	NOM	RANG
2	ID1	NOM1	RANG1
3	ID2	NOM2	RANG2
4	ID3	NOM3	RANG3
5	ID4	NOM4	RANG4
6	ID5	NOM5	RANG5
7	ID6	NOM6	RANG6
8			

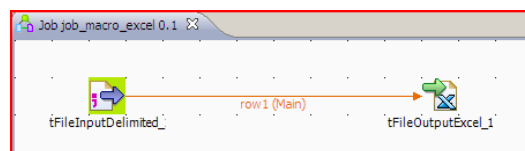
4. Job Talend Open Studio

La création du job TOS va se dérouler suivant ces étapes :

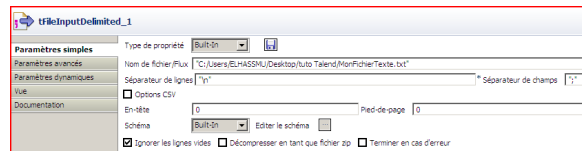
- création du job qui va remplir le fichier Excel ;
- mise en forme du fichier Excel en lançant la MACRO.

4.1. Création du job qui va remplir le fichier Excel

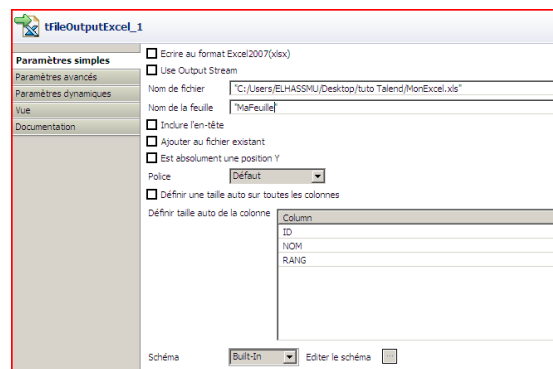
La création du job de remplissage du fichier Excel est ce qu'il y a de plus simple. Après création d'un job, on glisse un "tFileInputDelimited" dans le designer, puis un "tFileOutputExcel" et on lie le tout avec un lien de type "Main".



- Après, on accède aux propriétés du composant "tFileInputDelimited" pour spécifier ses paramètres (schéma, nom du fichier, séparateur de lignes...), on se retrouve finalement avec ça :



- Ceci fait, on passe aux paramètres du "tFileOutputExcel", pour obtenir cela :



- Notre job est fin prêt à l'emploi. On lance l'exécution, notre fichier Excel est bien rempli, il ne nous reste plus qu'à le mettre en forme.

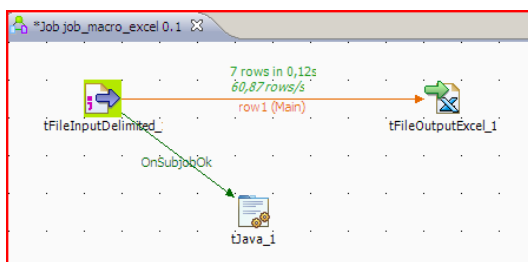
4.2. Mise en forme du fichier Excel en lançant la MACRO

Maintenant, on passe aux choses sérieuses en mettant en forme notre fichier Excel.

Pour cela, voici la MACRO qui a été utilisée. Elle a été enregistrée sur un fichier "MonVBScript.vbs". On ne va pas s'arrêter sur son contenu car ce n'est pas pertinent pour notre tutoriel :

```
Dim objXL
SET objXL = CreateObject("Excel.Application")
WITH objXL
    .Workbooks.Open
    ("C:\Users\ELHASSMU\Desktop\tuto
Talend\MonExcel.xls")
    .Range("A1:C1").Select
    .Selection.Font.Bold = True
    .Range("A2:A7").Select
    .Selection.Font.Bold = True
    .Range("B2:B7").Select
    .Selection.Font.Italic = True
    .Range("A1:C1").Select
    .Range("A2:C7").Select
    .Columns("C:C").ColumnWidth = 21.43
    .Columns("B:B").ColumnWidth = 16.29
    .Range("A1:C7").Select
    .Selection.Font.Size = 12
    .Selection.Font.Size = 14
    .Selection.Font.Size = 16
    .Range("A1:C1").Select
    .Range("A2:C7").Select
    .Columns("C:C").Select
    .Application.Quit
END WITH
SET objXL = Nothing
```

On modifie notre job en ajoutant un troisième composant : un "tJava". Comme l'exécution de la MACRO devrait se faire normalement après la fin du remplissage du fichier, on va alors lier le "tFileInputDelimited" avec le "tJava" par un lien "OnSubJobOk", ce qui nous donne ceci :

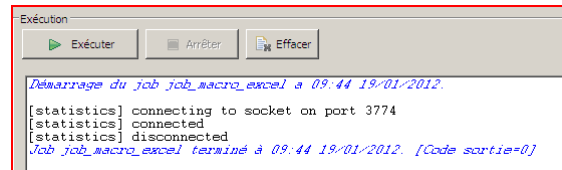


Voilà ! C'était ça l'astuce : lancer notre VBScript avec du

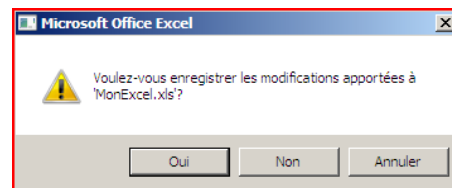
code JAVA qui ressemblerait à ça :

```
Runtime.getRuntime().exec("cmd /c start
C:\\Users\\ELHASSMU\\Desktop\\tuto
Talend\\MonVBScript.vbs");
```

Finalement, on lance l'exécution de notre job. Si tout se passe bien, notre console affichera le message suivant :



Et une petite fenêtre s'ouvrira pour nous demander si on veut ou non sauvegarder les changements apportés au fichier Excel :



On clique sur "Oui", bien sûr, puis on ouvre notre fichier Excel pour voir que la MACRO s'est bel et bien exécutée et que la mise en forme est en place :

	A	B	C
1	ID	NOM	RANG
2	1	ELHASSAK	4
3	2	ESBAISS	2
4	3	FRINDOU	2
5	4	BAIDOUNE	1
6	5	BENBARI	4
7	6	STAMBOUL	4

5. Conclusion

Merci d'avoir lu tout ce document :) . J'espère qu'il a pu aider quelques-uns de ses lecteurs, au moins pour les débutants sur TOS. Ceci dit, pour les gens qui souhaiteraient apprendre encore plus, je leur conseille d'aller sur le site officiel de Talend où il y'a plusieurs autres tutos très intéressants, sans oublier le forum sur Developpez où vous pouvez poser toutes vos questions !

Retrouvez l'article de Mustapha El Hassak en ligne : [Lien 137](#).

Liens

- Lien 01 : <http://man.developpez.com/man3/printf.3.php>
- Lien 02 : <http://qt.developpez.com/doc/4.7/qtglobal/#qdebug>
- Lien 03 : <http://qt.developpez.com/doc/4.7/qtglobal/#qwarning>
- Lien 04 : <http://qt.developpez.com/doc/4.7/qtglobal/#qcritical>
- Lien 05 : <http://qt.developpez.com/doc/4.7/qtglobal/#qfatal>
- Lien 06 : <http://qt.developpez.com/doc/4.7/qtglobal/#qinstallmsgshandler>
- Lien 07 : <http://qt.developpez.com/doc/latest/qobject/#dumpobjecttree>
- Lien 08 : <http://qt.developpez.com/doc/4.7/qobject/#dumpobjectinfo>
- Lien 09 : <http://man.developpez.com/man3/assert.3.php>
- Lien 10 : http://qt.developpez.com/doc/4.7/qtglobal/#q_assert
- Lien 11 : http://qt.developpez.com/doc/4.7/qtglobal/#q_assert_x
- Lien 12 : <http://websvn.kde.org/trunk/KDE/kdesdk/scripts/kde-devel-gdb>
- Lien 13 : <http://hiko-seijuro.developpez.com/articles/ddd/>
- Lien 14 : <http://www.codeblocks.org/>
- Lien 15 : <http://developer.qt.nokia.com/wiki/QtVSAaddin>
- Lien 16 : <http://loulou.developpez.com/tutoriels/cpp/debogueur-visual-studio/>
- Lien 17 : <https://gitorious.org/massif-visualizer>
- Lien 18 : http://qtunderground.org/wiki/Qt_Environment_Variables
- Lien 19 : <http://qt.developpez.com/doc/4.7/qscriptengine/debugger/>
- Lien 20 : <http://qt.developpez.com/doc/latest/qscriptengine/debugger/>
- Lien 21 : <http://qt.developpez.com/doc/latest/qwebinspector/>
- Lien 22 : http://alexandre-laurent.developpez.com/share_windows/DVP/Article/QtDD/Documents%20and%20Settings/HP_Propriétaire/Local%20Settings/Temp/www.kdab.com/gammaray
- Lien 23 : <http://qt.developpez.com/tutoriels/remi-achard/qt-test/>
- Lien 24 : http://get.qt.nokia.com/videos/DevDays2011/TechnicalSessions/DevDays2011_-_Effective_Debugging_And_Profiling_For_Qt_And_Qt_Quick.pdf
- Lien 25 : <http://developer.qt.nokia.com/videos/watch/the-last-mile-effective-debugging-and-profiling-for-qt-and-qt-quick>
- Lien 26 : <http://qt.developpez.com/doc/latest/index/>
- Lien 27 : <http://alexandre-laurent.developpez.com/tutoriels/qt/debogage-applications/>
- Lien 28 : <http://qt-labs.developpez.com/compilation/qmake-et-au-dela/>
- Lien 29 : <http://qt-labs.developpez.com/compilation/qmake-et-au-dela-redux/>
- Lien 30 : <http://lists.qt.nokia.com/pipermail/qt5-feedback/2011-June/000494.html>
- Lien 31 : <http://chaos.troll.no/%7Edmolkent/qbs-0.1/>
- Lien 32 : <http://aegis.sourceforge.net/auug97.pdf>
- Lien 33 : <http://gamesfromwithin.com/the-quest-for-the-perfect-build-system>
- Lien 34 : <http://blog.developpez.com/dourouc05/p10726/qt/adiieu-qmake-bienvenue-qbs/gitorious.org/qt-labs/qbs>
- Lien 35 : <http://blog.developpez.com/dourouc05/p10726/qt/adiieu-qmake-bienvenue-qbs/>
- Lien 36 : <http://claudeleloup.developpez.com/tutoriels/access/positionner-formulaire/>
- Lien 37 : <http://argyronet.developpez.com/office/access/setformtoright/>
- Lien 38 : <http://www.developpez.net/forums/d724265/logiciels/microsoft-office/access/contribuez/positionner-formulaire-sous-contrôle/#post4201158>
- Lien 39 : <http://arkham46.developpez.com/articles/office/officeweb/>
- Lien 40 : <http://argyronet.developpez.com/office/vba/convention/%20>
- Lien 41 : <http://claudeleloup.developpez.com/tutoriels/access/positionner-formulaire-par-rapport-contrôle-autre-formulaire/PositionnerFormulaire2.mdb>
- Lien 42 : <http://claudeleloup.developpez.com/tutoriels/access/positionner-formulaire-par-rapport-contrôle-autre-formulaire/>
- Lien 43 : <http://access.developpez.com/cours/?page=dataaccess#ado>
- Lien 44 : <http://ledzeppii.developpez.com/odbc-access/>
- Lien 45 : <http://jdgayot.developpez.com/tutoriels/access/importAs400/>
- Lien 46 : <http://www.developpez.com/actu/38274>
- Lien 47 : <http://windowsphone.developpez.com/actu/40215/>
- Lien 48 : <http://www.developpez.net/forums/d1171243/java/general-java/java-mobiles/android/android-market-400-000-applications-actives-succes-modele-freemium/>
- Lien 49 : <http://developer.android.com/design/index.html>
- Lien 50 : <http://www.developpez.net/forums/d1174412/java/general-java/java-mobiles/android/android-google-lance-nouveau-portail/>
- Lien 51 : <http://android.cyrilmottier.com/>
- Lien 52 : <http://www.tutos-android.com/wp-content/uploads/2011/05/drawable.zip>
- Lien 53 : <http://www.tutos-android.com/wp-content/uploads/2011/05/GDIntro.zip>
- Lien 54 : <http://android.cyrilmottier.com/>
- Lien 55 : <http://mbenbourahla.developpez.com/tutoriels/java/introduction-a-greendroid/>
- Lien 56 : <http://code.google.com/p/maven-android-plugin/>
- Lien 57 : http://developer.android.com/guide/topics/testing/testing_android.html
- Lien 58 : <http://code.google.com/p/robotium/>
- Lien 59 : <http://ydisanto.developpez.com/tutoriels/android/debuter/>
- Lien 60 : <http://thierry-leriche-dessirier.developpez.com/tutoriels/java/importer-projet-maven-dans-eclipse-5-min/>
- Lien 61 : <http://thierry-leriche-dessirier.developpez.com/tutoriels/java/methode-3t/>
- Lien 62 : <http://thierry-leriche-dessirier.developpez.com/tutoriels/java/3t-en-pratique/>
- Lien 63 : <http://thierry.leriche-dessirier.com/article/intro-junit.htm>
- Lien 64 : <http://maximeghignet.developpez.com/tutoriels/android/tests-unitaires-et-tests-ihm-sur-projet-android-utilisant-maven/>
- Lien 65 : <http://eclipse.developpez.com/cours/?page=platform-cat#plugin-dev>
- Lien 66 : <http://eclipse.developpez.com/cours/?page=platform-cat#ihm-dev>
- Lien 67 : http://fr.wikipedia.org/wiki/Problème_du_voyageur_de_commerce
- Lien 68 : <http://eclipse.developpez.com/cours/?page=platform-cat>
- Lien 69 : <http://www.eclipse.org/gef/zest/>
- Lien 70 : <http://alain-bernard.developpez.com/tutoriels/eclipse/intro-zest/>
- Lien 71 : <http://www.geuz.org/gmsh/#Screenshots>
- Lien 72 : <http://www.salome-platform.org/about/screenshots>
- Lien 73 : <http://vimeo.com/3287369>

Lien 74 : <http://vimeo.com/3082069>
Lien 75 : <http://www.oracle.com/technetwork/java/javase/downloads/index.html>
Lien 76 : <http://jogamp.org/>
Lien 77 : <http://netbeans.org/>
Lien 78 : <http://jogamp.org/deployment/jogamp-current/archive/>
Lien 79 : http://jogamp.org/wiki/index.php/Setting_up_a_JogAmp_project_in_your_favorite_IDE
Lien 80 : ftp://ftp-developpez.com/plegat/tutoriels/jogl/pge-intro/src_PGE_article_1.zip
Lien 81 : <http://plegat.developpez.com/tutoriels/jogl/pge-intro/>
Lien 82 : <http://jpvincent.developpez.com/tutoriels/javascript/trois-fondamentaux-javascript/>
Lien 83 : <http://www.slideshare.net/jpvincent/javascript-fondamentaux-et-oop>
Lien 84 : <http://fr.wikipedia.org/wiki/Mémoization>
Lien 85 : <http://ejohn.org/blog/partial-functions-in-javascript/>
Lien 86 : <http://jpvincent.developpez.com/tutoriels/javascript/javascript-orienté-objet-syntaxe-base-classes-js-intention-developpeurs-php/>
Lien 87 : <http://www.php.net/manual/fr/language.oop5.patterns.php#language.oop5.patterns.singleton>
Lien 88 : <http://jpvincent.developpez.com/tutoriels/javascript/usage-avance-fonctions-javascript/>
Lien 89 : <http://mootools.net/blog/2011/03/28/events-with-mootools-element-class-delegation-and-pseudos/>
Lien 90 : <https://github.com/anutron/behavior/blob/master/Tests/Specs/Behavior/Behavior.Specs.Helpers.js>
Lien 91 : <http://benchmarkjs.com/>
Lien 92 : <https://github.com/anutron/behavior/blob/master/Docs/BehaviorAPI.md>
Lien 93 : <https://github.com/anutron/behavior/blob/master/Docs/Behavior.md>
Lien 94 : <https://github.com/anutron/behavior/blob/master/Docs/Element.Data.md>
Lien 95 : <https://github.com/anutron/more-behaviors>
Lien 96 : <https://github.com/anutron/clientcide>
Lien 97 : <https://github.com/anutron/mootools-bootstrap>
Lien 98 : <http://anutron.github.com/mootools-bootstrap/>
Lien 99 : <http://dev.clientcide.com/>
Lien 100 : <https://github.com/anutron/behavior>
Lien 101 : <http://dev.clientcide.com/>
Lien 102 : <http://dev.clientcide.com/?version=MooTools%20Bootstrap>
Lien 103 : <http://javascript.developpez.com/tutoriels/mootools/mootools-behavior/>
Lien 104 : <http://christophe-f.developpez.com/tutoriels/css/cours-css/syntaxe-regle-css/exemples/ex01-principe-des-selecteurs-dans-regle-css.html>
Lien 105 : <http://christophe-f.developpez.com/tutoriels/css/cours-css/syntaxe-regle-css/exemples/ex02-principe-des-declarations-dans-regle-css.html>
Lien 106 : <http://christophe-f.developpez.com/tutoriels/css/cours-css/syntaxe-regle-css/exemples/ex03-indentation-code.html>
Lien 107 : <http://christophe-f.developpez.com/tutoriels/css/cours-css/syntaxe-regle-css/exemples/ex04-casse-css.html>
Lien 108 : <http://christophe-f.developpez.com/tutoriels/css/cours-css/syntaxe-regle-css/exemples/ex05-casse-nom-police.html>
Lien 109 : <http://christophe-f.developpez.com/tutoriels/css/cours-css/syntaxe-regle-css/exemples/ex06-regroupement-selecteurs.html>
Lien 110 : http://php.developpez.com/faq/langage/index.php?page=syntaxe#bases_commentaires
Lien 111 : <http://christophe-f.developpez.com/tutoriels/css/cours-css/syntaxe-regle-css/exemples/ex07-proprietes-raccourcies-font.html>
Lien 112 : <http://christophe-f.developpez.com/tutoriels/css/cours-css/syntaxe-regle-css/exemples/ex08-proprietes-raccourcies-bord-boite.html>
Lien 113 : <http://www.iana.org/assignments/character-sets>
Lien 114 : <http://christophe-f.developpez.com/tutoriels/css/cours-css/syntaxe-regle-css/>
Lien 115 : <http://www.google.com/webmasters/tools/richsnippets>
Lien 116 : http://www.youtube.com/watch?v=W4FbF8GKChk&feature=player_embedded
Lien 117 : <http://9elements.com/io/projects/html5/canvas/>
Lien 118 : <http://socket.io/>
Lien 119 : <http://www.html5rocks.com/en/tutorials/file/filesystem/>
Lien 120 : <http://www.whatwg.org/specs/web-apps/current-work/complete/workers.html>
Lien 121 : <http://www.html5rocks.com/en/>
Lien 122 : <http://html5demos.com/>
Lien 123 : <http://www.modernizr.com/>
Lien 124 : <http://synbioz.developpez.com/tutoriels/xhtml/introduction-html5/>
Lien 125 : http://en.wikipedia.org/wiki/Covariance_and_contravariance_%28computer_science%29
Lien 126 : <http://msdn.microsoft.com/fr-fr/library/9eekhta0.aspx>
Lien 127 : <http://msdn.microsoft.com/fr-fr/library/78dfe2yb.aspx>
Lien 128 : <http://msdn.microsoft.com/fr-fr/library/bb351562.aspx>
Lien 129 : <http://msdn.microsoft.com/fr-fr/library/bb344977.aspx>
Lien 130 : <http://msdn.microsoft.com/en-us/library/dd469487.aspx>
Lien 131 : <http://msdn.microsoft.com/fr-fr/library/8ehhxeaf.aspx>
Lien 132 : <http://msdn.microsoft.com/fr-fr/library/ms132151.aspx>
Lien 133 : <http://msdn.microsoft.com/fr-fr/library/4d7sx9hd.aspx>
Lien 134 : <http://msdn.microsoft.com/en-us/library/dd469484.aspx>
Lien 135 : <http://guruumeditation.developpez.com/articles/dotnet/variance/>
Lien 136 : <http://haskouse.developpez.com/tutoriels/etl/talend-open-studio/creation-job/>
Lien 137 : <http://haskouse.developpez.com/tutoriels/etl/talend-open-studio/talend-excel-macro/>